

IEEE Std 1003.1™, 2003 Edition

The Open Group Technical Standard
Base Specifications, Issue 6

Includes IEEE Std 1003.1™-2001 and IEEE Std 1003.1™-2001/Cor 1-2002

Information Technology — Portable Operating System Interface (POSIX®)

System Interfaces

Sponsor

Portable Applications Standards Committee
of the
IEEE Computer Society

and

The Open Group



THE *Open* GROUP

[This page intentionally left blank]

Abstract

This standard is simultaneously ISO/IEC 9945:2002, IEEE Std 1003.1-2001, and forms the core of the Single UNIX Specification, Version 3.

The IEEE Std 1003.1, 2003 Edition includes IEEE Std 1003.1-2001/Cor 1-2002 incorporated into IEEE Std 1003.1-2001 (base document). The Corrigendum addresses problems discovered since the approval of IEEE Std 1003.1-2001. These changes are mainly due to resolving integration issues raised by the merger of the base documents that were incorporated into IEEE Std 1003.1-2001, which is the single common revision to IEEE Std 1003.1TM-1996, IEEE Std 1003.2TM-1992, ISO/IEC 9945-1:1996, ISO/IEC 9945-2:1993, and the Base Specifications of The Open Group Single UNIX[®] Specification, Version 2.

This standard defines a standard operating system interface and environment, including a command interpreter (or “shell”), and common utility programs to support applications portability at the source code level. This standard is intended to be used by both applications developers and system implementors and comprises four major components (each in an associated volume):

- General terms, concepts, and interfaces common to all volumes of this standard, including utility conventions and C-language header definitions, are included in the Base Definitions volume.
- Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume.
- Definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs are included in the Shell and Utilities volume.
- Extended rationale that did not fit well into the rest of the document structure, which contains historical information concerning the contents of this standard and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume.

The following areas are outside the scope of this standard:

- Graphics interfaces
- Database management system interfaces
- Record I/O considerations
- Object or binary code portability
- System configuration and resource availability

This standard describes the external characteristics and facilities that are of importance to applications developers, rather than the internal construction techniques employed to achieve these capabilities. Special emphasis is placed on those functions and facilities that are needed in a wide variety of commercial applications.

Keywords

application program interface (API), argument, asynchronous, basic regular expression (BRE), batch job, batch system, built-in utility, byte, child, command language interpreter, CPU, extended regular expression (ERE), FIFO, file access control mechanism, input/output (I/O), job control, network, portable operating system interface (POSIX[®]), parent, shell, stream, string, synchronous, system, thread, X/Open System Interface (XSI)

Copyright © 2001-2003 by the Institute of Electrical and Electronics Engineers, Inc. and The Open Group. All rights reserved. This printing is by the International Organization for Standardization with special permission of the Institute of Electrical and Electronics Engineers, Inc. and The Open Group. Published in Switzerland.

System Interfaces, Issue 6

Published 31 March 2003 by the Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, U.S.A.
ISBN: 0-7381-3436-8 PDF 0-7381-3564-X/SS95078 CD-ROM 0-7381-3563-1/SE95078
Printed in the United States of America by the IEEE.

Published 31 March 2003 by The Open Group
Apex Plaza, Forbury Road, Reading, Berkshire RG1 1AX, U.K.
Document Number: C032
ISBN: 1-931624-24-0
Printed in the U.K. by The Open Group.

All rights reserved. No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without prior written permission from both the IEEE and The Open Group.

Portions of this standard are derived with permission from copyrighted material owned by Hewlett-Packard Company, International Business Machines Corporation, Novell Inc., The Open Software Foundation, and Sun Microsystems, Inc.

Permissions

Authorization to photocopy portions of this standard for internal or personal use is granted provided that the appropriate fee is paid to the Copyright Clearance Center or the equivalent body outside of the U.S. Permission to make multiple copies for educational purposes in the U.S. requires agreement and a license fee to be paid to the Copyright Clearance Center.

Beyond these provisions, permission to reproduce all or any part of this standard must be with the consent of both copyright holders and may be subject to a license fee. Both copyright holders will need to be satisfied that the other has granted permission. Requests to the copyright holders should be sent by email to austin-group-permissions@opengroup.org.

Feedback

This standard has been prepared by the Austin Group. Feedback relating to the material contained in this standard may be submitted using the Austin Group web site at <http://www.opengroup.org/austin/defectform.html>.

IEEE

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property, or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied "AS IS".

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of the IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with the IEEE.¹ Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, U.S.A.

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE Standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

A patent holder has filed a statement of assurance that it will grant licenses under these rights without compensation or under reasonable rates and non-discriminatory, reasonable terms and conditions to all applicants desiring to obtain such licenses. The IEEE makes no representation as to the reasonableness of rates and/or terms and conditions of the license agreements offered by patent holders. Further information may be obtained from the IEEE Standards Department.

Authorization to photocopy portions of any individual standard for internal or personal use is granted in the U.S. by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to the Copyright Clearance Center.² Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center. To arrange for payment of the licensing fee, please contact:

Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923, U.S.A., Tel.: +1 978 750 8400

Amendments, corrigenda, and interpretations for this standard, or information about the IEEE standards development process, may be found at <http://standards.ieee.org>.

Full catalog and ordering information on all IEEE publications is available from the IEEE Online Catalog & Store at <http://shop.ieee.org/store>.

1. For this standard, please send comments via the Austin Group as requested on page iii.

2. Please refer to the special provisions for this standard on page iii concerning permissions from both copyright holders and arrangements to cover photocopying and reproduction across the world, as well as by commercial organizations wishing to license the material for use in product documentation.

The Open Group

The Open Group, a vendor and technology-neutral consortium, is committed to delivering greater business efficiency by bringing together buyers and suppliers of information technology to lower the time, cost, and risks associated with integrating new technology across the enterprise.

The Open Group's mission is to offer all organizations concerned with open information infrastructures a forum to share knowledge, integrate open initiatives, and certify approved products and processes in a manner in which they continue to trust our impartiality.

In the global eCommerce world of today, no single economic entity can achieve independence while still ensuring interoperability. The assurance that products will interoperate with each other across differing systems and platforms is essential to the success of eCommerce and business workflow. The Open Group, with its proven testing and certification program, is the international guarantor of interoperability in the new century.

The Open Group provides opportunities to exchange information and shape the future of IT. The Open Group's members include some of the largest and most influential organizations in the world. The flexible structure of The Open Groups membership allows for almost any organization, no matter what their size, to join and have a voice in shaping the future of the IT world.

More information is available on The Open Group web site at <http://www.opengroup.org>.

The Open Group has over 15 years' experience in developing and operating certification programs and has extensive experience developing and facilitating industry adoption of test suites used to validate conformance to an open standard or specification. The Open Group portfolio of test suites includes the *Westwood* family of tests for this standard and the associated certification program for Version 3 of the Single UNIX Specification, as well tests for CDE, CORBA, Motif, Linux, LDAP, POSIX.1, POSIX.2, POSIX Realtime, Sockets, UNIX, XPG4, XNFS, XTI, and X11. The Open Group test tools are essential for proper development and maintenance of standards-based products, ensuring conformance of products to industry-standard APIs, applications portability, and interoperability. In-depth testing identifies defects at the earliest possible point in the development cycle, saving costs in development and quality assurance.

More information is available at <http://www.opengroup.org/testing>.

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical and Product Standards and Guides, but which also includes white papers, technical studies, branding and testing documentation, and business titles. Full details and a catalog are available at <http://www.opengroup.org/pubs>.

As with all *live* documents, Technical Standards and Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards compatible and those which are not:

- A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.
- A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published at <http://www.opengroup.org/corrigenda>.

Full catalog and ordering information on all Open Group publications is available at <http://www.opengroup.org/pubs>.

Contents

Chapter 1	Introduction.....	1
1.1	Scope.....	1
1.2	Conformance	1
1.3	Normative References	1
1.4	Change History	1
1.5	Terminology	1
1.6	Definitions	3
1.7	Relationship to Other Formal Standards	3
1.8	Portability	3
1.8.1	Codes	3
1.9	Format of Entries.....	11
Chapter 2	General Information	13
2.1	Use and Implementation of Functions	13
2.2	The Compilation Environment	13
2.2.1	POSIX.1 Symbols	13
2.2.1.1	The <code>_POSIX_C_SOURCE</code> Feature Test Macro	14
2.2.1.2	The <code>_XOPEN_SOURCE</code> Feature Test Macro.....	14
2.2.2	The Name Space.....	14
2.3	Error Numbers.....	21
2.3.1	Additional Error Numbers.....	28
2.4	Signal Concepts.....	28
2.4.1	Signal Generation and Delivery.....	28
2.4.2	Realtime Signal Generation and Delivery	29
2.4.3	Signal Actions	30
2.4.4	Signal Effects on Other Functions	34
2.5	Standard I/O Streams.....	34
2.5.1	Interaction of File Descriptors and Standard I/O Streams.....	35
2.5.2	Stream Orientation and Encoding Rules	36
2.6	STREAMS	38
2.6.1	Accessing STREAMS.....	39
2.7	XSI Interprocess Communication	39
2.7.1	IPC General Description.....	40
2.8	Realtime	41
2.8.1	Realtime Signals.....	41
2.8.2	Asynchronous I/O	41
2.8.3	Memory Management	43
2.8.3.1	Memory Locking.....	43
2.8.3.2	Memory Mapped Files.....	43
2.8.3.3	Memory Protection.....	43
2.8.3.4	Typed Memory Objects	44
2.8.4	Process Scheduling	44

2.8.5	Clocks and Timers	48
2.9	Threads.....	50
2.9.1	Thread-Safety.....	50
2.9.2	Thread IDs.....	51
2.9.3	Thread Mutexes.....	51
2.9.4	Thread Scheduling.....	52
2.9.5	Thread Cancellation	54
2.9.5.1	Cancelability States	54
2.9.5.2	Cancellation Points.....	55
2.9.5.3	Thread Cancellation Cleanup Handlers	57
2.9.5.4	Async-Cancel Safety.....	57
2.9.6	Thread Read-Write Locks.....	58
2.9.7	Thread Interactions with Regular File Operations	58
2.10	Sockets.....	58
2.10.1	Address Families.....	58
2.10.2	Addressing	59
2.10.3	Protocols	59
2.10.4	Routing.....	59
2.10.5	Interfaces.....	59
2.10.6	Socket Types.....	59
2.10.7	Socket I/O Mode.....	60
2.10.8	Socket Owner.....	60
2.10.9	Socket Queue Limits	60
2.10.10	Pending Error.....	60
2.10.11	Socket Receive Queue.....	61
2.10.12	Socket Out-of-Band Data State	61
2.10.13	Connection Indication Queue	62
2.10.14	Signals	62
2.10.15	Asynchronous Errors	62
2.10.16	Use of Options.....	63
2.10.17	Use of Sockets for Local UNIX Connections.....	66
2.10.17.1	Headers	66
2.10.18	Use of Sockets over Internet Protocols.....	66
2.10.19	Use of Sockets over Internet Protocols Based on IPv4.....	67
2.10.19.1	Headers	67
2.10.20	Use of Sockets over Internet Protocols Based on IPv6.....	67
2.10.20.1	Addressing	67
2.10.20.2	Compatibility with IPv4.....	68
2.10.20.3	Interface Identification.....	68
2.10.20.4	Options.....	69
2.10.20.5	Headers	70
2.11	Tracing.....	70
2.11.1	Tracing Data Definitions.....	71
2.11.1.1	Structures.....	71
2.11.1.2	Trace Stream Attributes.....	75
2.11.2	Trace Event Type Definitions	76
2.11.2.1	System Trace Event Type Definitions.....	76
2.11.2.2	User Trace Event Type Definitions.....	79

Contents

	2.11.3	Trace Functions.....	79
	2.12	Data Types.....	80
Chapter	3	System Interfaces	83
		Index.....	1669

List of Tables

2-1	Value of Level for Socket Options.....	63
2-2	Socket-Level Options.....	64
2-3	Trace Option: System Trace Events.....	77
2-4	Trace and Trace Event Filter Options: System Trace Events.....	78
2-5	Trace and Trace Log Options: System Trace Events.....	78
2-6	Trace, Trace Log, and Trace Event Filter Options: System Trace Events	79
2-7	Trace Option: User Trace Event	79

Foreword

Structure of the Standard

This standard was originally developed by the Austin Group, a joint working group of members of the IEEE, members of The Open Group, and members of ISO/IEC Joint Technical Committee 1, as one of the four volumes of IEEE Std 1003.1-2001. The standard was approved by ISO and IEC and published in four parts, correlating to the original volumes.

A mapping of the parts to the volumes is shown below:

ISO/IEC 9945 Part	IEEE Std 1003.1 Volume	Description
9945-1	Base Definitions	Includes general terms, concepts, and interfaces common to all parts of ISO/IEC 9945, including utility conventions and C-language header definitions.
9945-2	System Interfaces	Includes definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery.
9945-3	Shell and Utilities	Includes definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs.
9945-4	Rationale	Includes extended rationale that did not fit well into the rest of the document structure, containing historical information concerning the contents of ISO/IEC 9945 and why features were included or discarded by the standard developers.

All four parts comprise the entire standard, and are intended to be used together to accommodate significant internal referencing among them. POSIX-conforming systems are required to support all four parts.

Introduction

Note: This introduction is not part of IEEE Std 1003.1-2001, Standard for Information Technology — Portable Operating System Interface (POSIX).

This standard has been jointly developed by the IEEE and The Open Group. It is simultaneously an IEEE Standard, an ISO/IEC Standard, and an Open Group Technical Standard.

The Austin Group

This standard was developed, and is maintained, by a joint working group of members of the IEEE Portable Applications Standards Committee, members of The Open Group, and members of ISO/IEC Joint Technical Committee 1. This joint working group is known as the Austin Group.³ The Austin Group arose out of discussions amongst the parties which started in early 1998, leading to an initial meeting and formation of the group in September 1998. The purpose of the Austin Group has been to revise, combine, and update the following standards: ISO/IEC 9945-1, ISO/IEC 9945-2, IEEE Std 1003.1, IEEE Std 1003.2, and the Base Specifications of The Open Group Single UNIX Specification.

After two initial meetings, an agreement was signed in July 1999 between The Open Group and the Institute of Electrical and Electronics Engineers (IEEE), Inc., to formalize the project with the first draft of the revised specifications being made available at the same time. Under this agreement, The Open Group and IEEE agreed to share joint copyright of the resulting work. The Open Group has provided the chair and secretariat for the Austin Group.

The base document for the revision was The Open Group's Base volumes of its Single UNIX Specification, Version 2. These were selected since they were a superset of the existing POSIX.1 and POSIX.2 specifications and had some organizational aspects that would benefit the audience for the new revision.

The approach to specification development has been one of “write once, adopt everywhere”, with the deliverables being a set of specifications that carry the IEEE POSIX designation, The Open Group's Technical Standard designation, and an ISO/IEC designation. This set of specifications forms the core of the Single UNIX Specification, Version 3.

This unique development has combined both the industry-led efforts and the formal standardization activities into a single initiative, and included a wide spectrum of participants. The Austin Group continues as the maintenance body for this document.

Anyone wishing to participate in the Austin Group should contact the chair with their request. There are no fees for participation or membership. You may participate as an observer or as a contributor. You do not have to attend face-to-face meetings to participate; electronic participation is most welcome. For more information on the Austin Group and how to participate, see <http://www.opengroup.org/austin>.

3. The Austin Group is named after the location of the inaugural meeting held at the IBM facility in Austin, Texas in September 1998.

Background

The developers of this standard represent a cross section of hardware manufacturers, vendors of operating systems and other software development tools, software designers, consultants, academics, authors, applications programmers, and others.

Conceptually, this standard describes a set of fundamental services needed for the efficient construction of application programs. Access to these services has been provided by defining an interface, using the C programming language, a command interpreter, and common utility programs that establish standard semantics and syntax. Since this interface enables application writers to write portable applications—it was developed with that goal in mind—it has been designated POSIX,⁴ an acronym for Portable Operating System Interface.

Although originated to refer to the original IEEE Std 1003.1-1988, the name POSIX more correctly refers to a *family* of related standards: IEEE Std 1003.*n* and the parts of ISO/IEC 9945. In earlier editions of the IEEE standard, the term POSIX was used as a synonym for IEEE Std 1003.1-1988. A preferred term, POSIX.1, emerged. This maintained the advantages of readability of the symbol “POSIX” without being ambiguous with the POSIX family of standards.

Audience

The intended audience for this standard is all persons concerned with an industry-wide standard operating system based on the UNIX system. This includes at least four groups of people:

1. Persons buying hardware and software systems
2. Persons managing companies that are deciding on future corporate computing directions
3. Persons implementing operating systems, and especially
4. Persons developing applications where portability is an objective

Purpose

Several principles guided the development of this standard:

- Application-Oriented

The basic goal was to promote portability of application programs across UNIX system environments by developing a clear, consistent, and unambiguous standard for the interface specification of a portable operating system based on the UNIX system documentation. This standard codifies the common, existing definition of the UNIX system.

- Interface, Not Implementation

This standard defines an interface, not an implementation. No distinction is made between library functions and system calls; both are referred to as functions. No details of the implementation of any function are given (although historical practice is sometimes indicated in the RATIONALE section). Symbolic names are given for constants (such as signals and error numbers) rather than numbers.

4. The name POSIX was suggested by Richard Stallman. It is expected to be pronounced *pahz-icks*, as in *positive*, not *poh-six*, or other variations. The pronunciation has been published in an attempt to promulgate a standardized way of referring to a standard operating system interface.

- Source, Not Object, Portability

This standard has been written so that a program written and translated for execution on one conforming implementation may also be translated for execution on another conforming implementation. This standard does not guarantee that executable (object or binary) code will execute under a different conforming implementation than that for which it was translated, even if the underlying hardware is identical.

- The C Language

The system interfaces and header definitions are written in terms of the standard C language as specified in the ISO C standard.

- No Superuser, No System Administration

There was no intention to specify all aspects of an operating system. System administration facilities and functions are excluded from this standard, and functions usable only by the superuser have not been included. Still, an implementation of the standard interface may also implement features not in this standard. This standard is also not concerned with hardware constraints or system maintenance.

- Minimal Interface, Minimally Defined

In keeping with the historical design principles of the UNIX system, the mandatory core facilities of this standard have been kept as minimal as possible. Additional capabilities have been added as optional extensions.

- Broadly Implementable

The developers of this standard endeavored to make all specified functions implementable across a wide range of existing and potential systems, including:

1. All of the current major systems that are ultimately derived from the original UNIX system code (Version 7 or later)
2. Compatible systems that are not derived from the original UNIX system code
3. Emulations hosted on entirely different operating systems
4. Networked systems
5. Distributed systems
6. Systems running on a broad range of hardware

No direct references to this goal appear in this standard, but some results of it are mentioned in the Rationale (Informative) volume.

- Minimal Changes to Historical Implementations

When the original version of IEEE Std 1003.1 was published, there were no known historical implementations that did not have to change. However, there was a broad consensus on a set of functions, types, definitions, and concepts that formed an interface that was common to most historical implementations.

The adoption of the 1988 and 1990 IEEE system interface standards, the 1992 IEEE shell and utilities standard, the various Open Group (formerly X/Open) specifications, and the subsequent revisions and addenda to all of them have consolidated this consensus, and this revision reflects the significantly increased level of consensus arrived at since the original versions. The earlier standards and their modifications specified a number of areas where consensus had not been reached before, and these are now reflected in this revision. The authors of the original versions tried, as much as possible, to follow the principles below

when creating new specifications:

1. By standardizing an interface like one in an historical implementation; for example, directories
2. By specifying an interface that is readily implementable in terms of, and backwards-compatible with, historical implementations, such as the extended *tar* format defined in the *pax* utility
3. By specifying an interface that, when added to an historical implementation, will not conflict with it; for example, the *sigaction()* function

This revision tries to minimize the number of changes required to implementations which conform to the earlier versions of the approved standards to bring them into conformance with the current standard. Specifically, the scope of this work excluded doing any “new” work, but rather collecting into a single document what had been spread across a number of documents, and presenting it in what had been proven in practice to be a more effective way. Some changes to prior conforming implementations were unavoidable, primarily as a consequence of resolving conflicts found in prior revisions, or which became apparent when bringing the various pieces together.

However, since it references the 1999 version of the ISO C standard, and no longer supports “Common Usage C”, there are a number of unavoidable changes. Applications portability is similarly affected.

This standard is specifically not a codification of a particular vendor’s product.

It should be noted that implementations will have different kinds of extensions. Some will reflect “historical usage” and will be preserved for execution of pre-existing applications. These functions should be considered “obsolescent” and the standard functions used for new applications. Some extensions will represent functions beyond the scope of this standard. These need to be used with careful management to be able to adapt to future extensions of this standard and/or port to implementations that provide these services in a different manner.

- Minimal Changes to Existing Application Code

A goal of this standard was to minimize additional work for the developers of applications. However, because every known historical implementation will have to change at least slightly to conform, some applications will have to change.

This Standard

This standard defines the Portable Operating System Interface (POSIX) requirements and consists of the following volumes:

- Base Definitions
- Shell and Utilities
- System Interfaces (this volume)
- Rationale (Informative)

This Volume

The System Interfaces volume describes the interfaces offered to application programs by POSIX-conformant systems. Readers are expected to be experienced C language programmers, and to be familiar with the Base Definitions volume.

This volume is structured as follows:

- Chapter 1 explains the status of this volume and its relationship to other formal standards.
- Chapter 2 contains important concepts, terms, and caveats relating to the rest of this volume.
- Chapter 3 defines the functional interfaces to the POSIX-conformant system.

Comprehensive references are available in the index.

Typographical Conventions

The following typographical conventions are used throughout this standard. In the text, this standard is referred to as IEEE Std 1003.1-2001, which is technically identical to The Open Group Base Specifications, Issue 6.

The typographical conventions listed here are for ease of reading only. Editorial inconsistencies in the use of typography are unintentional and have no normative meaning in this standard.

Reference	Example	Notes
C-Language Data Structure	aiocb	
C-Language Data Structure Member	<i>aio_lio_opcode</i>	
C-Language Data Type	long	
C-Language External Variable	<i>errno</i>	
C-Language Function	<i>system()</i>	
C-Language Function Argument	<i>arg1</i>	
C-Language Function Family	<i>exec</i>	
C-Language Header	<sys/stat.h>	
C-Language Keyword	return	
C-Language Macro with Argument	<i>assert()</i>	
C-Language Macro with No Argument	INET_ADDRSTRLEN	
C-Language Preprocessing Directive	#define	
Commands within a Utility	a, c	
Conversion Specification, Specifier/Modifier Character	<i>%A, g, E</i>	1
Environment Variable	<i>PATH</i>	
Error Number	[EINTR]	
Example Output	Hello, World	
Filename	/tmp	
Literal Character	<i>'c', '\r', '\'</i>	2
Literal String	<i>"abcde"</i>	2
Optional Items in Utility Syntax	[]	
Parameter	<directory pathname>	
Special Character	<newline>	3
Symbolic Constant	_POSIX_VDISABLE	
Symbolic Limit, Configuration Value	{LINE_MAX}	4
Syntax	#include <sys/stat.h>	

Reference	Example	Notes
User Input and Example Code	<code>echo Hello, World</code>	5
Utility Name	<code>awk</code>	
Utility Operand	<code>file_name</code>	
Utility Option	<code>-c</code>	
Utility Option with Option-Argument	<code>-w width</code>	

Notes:

1. Conversion specifications, specifier characters, and modifier characters are used primarily in date-related functions and utilities and the *fprintf* and *scanf* formatting functions.
2. Unless otherwise noted, the quotes shall not be used as input or output. When used in a list item, the quotes are omitted. For literal characters, `'\'` (or any of the other sequences such as `''`) is the same as the C constant `'\\'` (or `'\''`).
3. The style selected for some of the special characters, such as `<newline>`, matches the form of the input given to the *localedef* utility. Generally, the characters selected for this special treatment are those that are not visually distinct, such as the control characters `<tab>` or `<newline>`.
4. Names surrounded by braces represent symbolic limits or configuration values which may be declared in appropriate headers by means of the C `#define` construct.
5. Brackets shown in this font, "[]", are part of the syntax and do *not* indicate optional items. In syntax the `'|'` symbol is used to separate alternatives, and ellipses ("...") are used to show that additional arguments are optional.

Shading is used to identify extensions and options; see Section 1.8.1 (on page 3).

Footnotes and notes within the body of the normative text are for information only (informative).

Informative sections (such as Rationale, Change History, Application Usage, and so on) are denoted by continuous shading bars in the margins.

Ranges of values are indicated with parentheses or brackets as follows:

- (a,b) means the range of all values from a to b , including neither a nor b .
- $[a,b]$ means the range of all values from a to b , including a and b .
- $[a,b)$ means the range of all values from a to b , including a , but not b .
- $(a,b]$ means the range of all values from a to b , including b , but not a .

Notes:

1. Symbolic limits are used in this volume instead of fixed values for portability. The values of most of these constants are defined in the Base Definitions volume, `<limits.h>` or `<unistd.h>`.
2. The values of errors are defined in the Base Definitions volume, `<errno.h>`.

Participants

IEEE Std 1003.1-2001 was prepared by the Austin Group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society, The Open Group, and ISO/SC22 WG15.

The Austin Group

At the time of approval, the membership of the Austin Group was as follows:

Andrew Josey, Chair

Donald W. Cragun, Organizational Representative, IEEE PASC

Nicholas Stoughton, Organizational Representative, ISO/SC22 WG15

Mark Brown, Organizational Representative, The Open Group

Cathy Hughes, Technical Editor

Austin Group Technical Reviewers

Peter Anvin

Bouazza Bachar

Theodore P. Baker

Walter Briscoe

Mark Brown

Dave Butenhof

Geoff Clare

Donald W. Cragun

Lee Damico

Ulrich Drepper

Paul Eggert

Joanna Farley

Clive D.W. Feather

Andrew Gollan

Michael Gonzalez

Joseph M. Gwinn

Jon Hitchcock

Yvette Ho Sang

Cathy Hughes

Lowell G. Johnson

Andrew Josey

Michael Kavanaugh

David Korn

Marc Aurele La France

Jim Meyering

Gary Miller

Finnbarr P. Murphy

Joseph S. Myers

Sandra O'Donnell

Frank Prindle

Curtis Royster Jr.

Glen Seeds

Keld Jorn Simonsen

Raja Srinivasan

Nicholas Stoughton

Donn S. Terry

Fred Tydeman

Peter Van Der Veen

James Youngman

Jim Zepeda

Jason Zions

Austin Group Working Group Members

Harold C. Adams
Peter Anvin
Pierre-Jean Arcos
Jay Ashford
Bouazza Bachar
Theodore P. Baker
Robert Barned
Joel Berman
David J. Blackwood
Shirley Bockstahler-Brandt
James Bottomley
Walter Briscoe
Andries Brouwer
Mark Brown
Eric W. Burger
Alan Burns
Andries Brouwer
Dave Butenhof
Keith Chow
Geoff Clare
Donald W. Cragun
Lee Damico
Juan Antonio De La Puente
Ming De Zhou
Steven J. Dovich
Richard P. Draves
Ulrich Drepper
Paul Eggert
Philip H. Enslow
Joanna Farley
Clive D.W. Feather
Pete Forman
Mark Funkenhauser
Lois Goldthwaite
Andrew Gollan

Michael Gonzalez
Karen D. Gordon
Joseph M. Gwinn
Steven A. Haaser
Charles E. Hammons
Chris J. Harding
Barry Hedquist
Vincent E. Henley
Karl Heubaum
Jon Hitchcock
Yvette Ho Sang
Niklas Holsti
Thomas Hosmer
Cathy Hughes
Jim D. Isaak
Lowell G. Johnson
Michael B. Jones
Andrew Josey
Michael J. Karels
Michael Kavanaugh
David Korn
Steven Kramer
Thomas M. Kurihara
Marc Aurele La France
C. Douglass Locke
Nick Maclaren
Roger J. Martin
Craig H. Meyer
Jim Meyering
Gary Miller
Finnbarr P. Murphy
Joseph S. Myers
John Napier
Peter E. Obermayer
James T. Oblinger

Sandra O'Donnell
Frank Prindle
Francois Riche
John D. Riley
Andrew K. Roach
Helmut Roth
Jaideep Roy
Curtis Royster Jr.
Stephen C. Schwarm
Glen Seeds
Richard Seibel
David L. Shroods Jr.
W. Olin Sibert
Keld Jorn Simonsen
Curtis Smith
Raja Srinivasan
Nicholas Stoughton
Marc J. Teller
Donn S. Terry
Fred Tydeman
Mark-Rene Uchida
Scott A. Valcourt
Peter Van Der Veen
Michael W. Vannier
Eric Vought
Frederick N. Webb
Paul A.T. Wolfgang
Garrett A. Wollman
James Youngman
Oren Yuen
Janusz Zalewski
Jim Zepeda
Jason Zions

The Open Group

When The Open Group approved the Base Specifications, Issue 6 on 12 September 2001, the membership of The Open Group Base Working Group was as follows:

Andrew Josey, Chair

Finnbarr P. Murphy, Vice-Chair

Mark Brown, Austin Group Liaison

Cathy Hughes, Technical Editor

Base Working Group Members

Bouazza Bachar

Mark Brown

Dave Butenhof

Donald W. Cragun

Larry Dwyer

Joanna Farley

Andrew Gollan

Karen D. Gordon

Gary Miller

Finnbarr P. Murphy

Frank Prindle

Andrew K. Roach

Curtis Royster Jr.

Nicholas Stoughton

Kenjiro Tsuji

Participants

IEEE

When the IEEE Standards Board approved IEEE Std 1003.1-2001 on 6 December 2001, the membership of the committees was as follows:

Portable Applications Standards Committee (PASC)

Lowell G. Johnson, Chair
Joseph M. Gwinn, Vice-Chair
Jay Ashford, Functional Chair
Andrew Josey, Functional Chair
Curtis Royster Jr., Functional Chair
Nicholas Stoughton, Secretary

Balloting Committee

The following members of the balloting committee voted on IEEE Std 1003.1-2001. Balloters may have voted for approval, disapproval, or abstention:

Harold C. Adams	Steven A. Haaser	Frank Prindle
Pierre-Jean Arcos	Charles E. Hammons	Francois Riche
Jay Ashford	Chris J. Harding	John D. Riley
Theodore P. Baker	Barry Hedquist	Andrew K. Roach
Robert Barned	Vincent E. Henley	Helmut Roth
David J. Blackwood	Karl Heubaum	Jaideep Roy
Shirley Bockstahler-Brandt	Niklas Holsti	Curtis Royster Jr.
James Bottomley	Thomas Hosmer	Stephen C. Schwarm
Mark Brown	Jim D. Isaak	Richard Seibel
Eric W. Burger	Lowell G. Johnson	David L. Shroads Jr.
Alan Burns	Michael B. Jones	W. Olin Sibert
Dave Butenhof	Andrew Josey	Keld Jorn Simonsen
Keith Chow	Michael J. Karels	Nicholas Stoughton
Donald W. Cragun	Steven Kramer	Donn S. Terry
Juan Antonio De La Puente	Thomas M. Kurihara	Mark-Rene Uchida
Ming De Zhou	C. Douglass Locke	Scott A. Valcourt
Steven J. Dovich	Roger J. Martin	Michael W. Vannier
Richard P. Draves	Craig H. Meyer	Frederick N. Webb
Philip H. Enslow	Finnbarr P. Murphy	Paul A.T. Wolfgang
Michael Gonzalez	John Napier	Oren Yuen
Karen D. Gordon	Peter E. Obermayer	Janusz Zalewski
Joseph M. Gwinn	James T. Oblinger	

The following organizational representative voted on this standard:

Andrew Josey, X/Open Company Ltd.

IEEE-SA Standards Board

When the IEEE-SA Standards Board approved IEEE Std 1003.1-2001 on 6 December 2001, it had the following membership:

Donald N. Heirman, Chair
James T. Carlo, Vice-Chair
Judith Gorman, Secretary

Satish K. Aggarwal
Mark D. Bowman
Gary R. Engmann
Harold E. Epstein
H. Landis Floyd
Jay Forster*
Howard M. Frazier
Ruben D. Garzon

James H. Gurney
Richard J. Holleman
Lowell G. Johnson
Robert J. Kennelly
Joseph L. Koepfinger*
Peter H. Lips
L. Bruce McClung
Daleep C. Mohla

James W. Moore
Robert F. Munzner
Ronald C. Petersen
Gerald H. Peterson
John B. Posey
Gary S. Robinson
Akio Tojo
Donald W. Zipse

Also included are the following non-voting IEEE-SA Standards Board liaisons:

Alan Cookson, NIST Representative
Donald R. Volzka, TAB Representative
Yvette Ho Sang, **Don Messina**, **Savoula Amanatidis**, IEEE Project Editors

* Member Emeritus

Participants

IEEE Std 1003.1-2001/Cor 1-2002 was prepared by the Austin Group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society, The Open Group, and ISO/IEC JTC 1/SC22/WG15.

The Austin Group

At the time of approval, the membership of the Austin Group was as follows:

Andrew Josey, Chair

Donald W. Cragun, Organizational Representative, IEEE PASC

Nicholas Stoughton, Organizational Representative, ISO/IEC JTC 1/SC22/WG15

Mark Brown, Organizational Representative, The Open Group

Cathy Fox, Technical Editor

Austin Group Technical Reviewers

Theodore P. Baker

Julian Blake

Andries Brouwer

Mark Brown

Dave Butenhof

Geoff Clare

Donald W. Cragun

Ken Dawson

Ulrich Drepper

Larry Dwyer

Paul Eggert

Joanna Farley

Clive D.W. Feather

Cathy Fox

Mark Funkenhauser

Lois Goldthwaite

Andrew Gollan

Michael Gonzalez

Bruno Haible

Ben Harris

Jon Hitchcock

Andreas Jaeger

Andrew Josey

Jonathan Lennox

Nick Maclaren

Jack McCann

Wilhelm Mueller

Joseph S. Myers

Frank Prindle

Kenneth Raeburn

Tim Robbins

Glen Seeds

Matthew Seitz

Keld Jorn Simonsen

Nicholas Stoughton

Alexander Terekhov

Donn S. Terry

Mike Wilson

Garrett A. Wollman

Mark Ziegast

Austin Group Working Group Members

Harold C. Adams
Alejandro Alonso
Jay Ashford
Theodore P. Baker
David J. Blackwood
Julian Blake
Mitchell Bonnett
Andries Brouwer
Mark Brown
Eric W. Burger
Alan Burns
Dave Butenhof
Keith Chow
Geoff Clare
Luis Cordova
Donald W. Cragun
Dragan Cvetkovic
Lee Damico
Ken Dawson
Jeroen Dekkers
Juan Antonio De La Puente
Steven J. Dovich
Ulrich Drepper
Dr. Sourav Dutta
Larry Dwyer
Paul Eggert
Joanna Farley

Clive D.W. Feather
Yaacov Fenster
Cathy Fox
Mark Funkenhauser
Lois Goldthwaite
Andrew Gollan
Michael Gonzalez
Karen D. Gordon
Scott Gudgel
Joseph M. Gwinn
Steven A. Haaser
Bruno Haible
Charles E. Hammons
Bryan Harold
Ben Harris
Barry Hedquist
Karl Heubaum
Jon Hitchcock
Andreas Jaeger
Andrew Josey
Kenneth Lang
Pi-Cheng Law
Jonathan Lennox
Nick Maclaren
Roger J. Martin
Jack McCann
George Miao

Wilhelm Mueller
Finnbarr P. Murphy
Joseph S. Myers
Alexey Neyman
Charles Ngethe
Peter Petrov
Frank Prindle
Vikram Punj
Kenneth Raeburn
Francois Riche
Tim Robbins
Curtis Royster Jr.
Diane Schleicher
Gil Shultz
Stephen C. Schwarm
Glen Seeds
Matthew Seitz
Keld Jorn Simonsen
Doug Stevenson
Nicholas Stoughton
Alexander Terekhov
Donn S. Terry
Mike Wilson
Garrett A. Wollman
Oren Yuen
Mark Ziegast

Participants

The Open Group

When The Open Group approved the Base Specifications, Issue 6, Technical Corrigendum 1 on 7 February 2003, the membership of The Open Group Base Working Group was as follows:

Andrew Josey, Chair

Finnbarr P. Murphy, Vice-Chair

Mark Brown, Austin Group Liaison

Cathy Fox, Technical Editor

Base Working Group Members

Mark Brown

Dave Butenhof

Donald W. Cragun

Larry Dwyer

Ulrich Drepper

Joanna Farley

Andrew Gollan

Finnbarr P. Murphy

Frank Prindle

Andrew K. Roach

Curtis Royster Jr.

Nicholas Stoughton

Kenjiro Tsuji

IEEE

When the IEEE Standards Board approved IEEE Std 1003.1-2001/Cor 1-2002 on 11 December 2002, the membership of the committees was as follows:

Portable Applications Standards Committee (PASC)

Lowell G. Johnson, Chair
Joseph M. Gwinn, Vice-Chair
Jay Ashford, Functional Chair
Andrew Josey, Functional Chair
Curtis Royster Jr., Functional Chair
Nicholas Stoughton, Secretary

Balloting Committee

The following members of the balloting committee voted on IEEE Std 1003.1-2001/Cor 1-2002. Balloters may have voted for approval, disapproval, or abstention:

Alejandro Alonso	Michael Gonzalez	Charles Ngethe
Jay Ashford	Scott Gudgel	Peter Petrov
David J. Blackwood	Charles E. Hammons	Frank Prindle
Julian Blake	Bryan Harold	Vikram Punj
Mitchell Bonnett	Barry Hedquist	Francois Riche
Mark Brown	Karl Heubaum	Curtis Royster Jr.
Dave Butenhof	Lowell G. Johnson	Diane Schleicher
Keith Chow	Andrew Josey	Stephen C. Schwarm
Luis Cordova	Kenneth Lang	Gil Shultz
Donald W. Cragun	Pi-Cheng Law	Nicholas Stoughton
Steven J. Dovich	George Miao	Donn S. Terry
Dr. Sourav Dutta	Roger J. Martin	Oren Yuen
Yaacov Fenster	Finnbarr P. Murphy	Juan A. de la Puente

Participants

IEEE-SA Standards Board

When the IEEE-SA Standards Board approved IEEE Std 1003.1-2001/Cor 1-2002 on 11 December 2002, the membership was as follows:

James T. Carlo, Chair

James H. Gurney, Vice-Chair

Judith Gorman, Secretary

Sid Bennett

H. Stephen Berger

Clyde R. Camp

Richard DeBlasio

Harold E. Epstein

Julian Forster*

Howard M. Frazier

Toshio Fukuda

Arnold M. Greenspan

Raymond Hapeman

Donald M. Heirman

Richard H. Hulett

Lowell G. Johnson

Joseph L. Koepfinger*

Peter H. Lips

Nader Mehravari

Daleep C. Mohla

William J. Moylan

Malcolm V. Thaden

Geoffrey O. Thompson

Howard L. Wolfman

Don Wright

Also included are the following non-voting IEEE-SA Standards Board liaisons:

Alan Cookson, NIST Representative

Satish K. Aggarwal, NRC Representative

Savoula Amanatidis, IEEE Standards Managing Editor

* Member Emeritus

Trademarks

The following information is given for the convenience of users of this standard and does not constitute endorsement of these products by The Open Group or the IEEE. There may be other products mentioned in the text that might be covered by trademark protection and readers are advised to verify them independently.

1003.1™ is a trademark of the Institute of Electrical and Electronic Engineers, Inc.

AIX® is a registered trademark of IBM Corporation.

AT&T® is a registered trademark of AT&T in the U.S.A. and other countries.

BSD™ is a trademark of the University of California, Berkeley, U.S.A.

Hewlett-Packard®, HP®, and HP-UX® are registered trademarks of Hewlett-Packard Company.

IBM® is a registered trademark of International Business Machines Corporation.

The Open Group and Boundaryless Information Flow are trademarks and UNIX is a registered trademark of The Open Group in the United States and other countries. All other trademarks are the property of their respective owners.

POSIX® is a registered trademark of the Institute of Electrical and Electronic Engineers, Inc.

Sun® and Sun Microsystems® are registered trademarks of Sun Microsystems, Inc.

/usr/group® is a registered trademark of UniForum, the International Network of UNIX System Users.

Acknowledgements

The contributions of the following organizations to the development of IEEE Std 1003.1-2001 are gratefully acknowledged:

- AT&T for permission to reproduce portions of its copyrighted System V Interface Definition (SVID) and material from the UNIX System V Release 2.0 documentation.
- The SC22 WG14 Committees.

This standard was prepared by the Austin Group, a joint working group of the IEEE, The Open Group, and ISO SC22 WG15.

Referenced Documents

Normative References

Normative references for this standard are defined in the Base Definitions volume.

Informative References

The following documents are referenced in this standard:

1984 /usr/group Standard

/usr/group Standards Committee, Santa Clara, CA, UniForum 1984.

Almasi and Gottlieb

George S. Almasi and Allan Gottlieb, *Highly Parallel Computing*, The Benjamin/Cummings Publishing Company, Inc., 1989, ISBN: 0-8053-0177-1.

ANSI C

American National Standard for Information Systems: Standard X3.159-1989, Programming Language C.

ANSI X3.226-1994

American National Standard for Information Systems: Standard X3.226-1994, Programming Language Common LISP.

Brawer

Steven Brawer, *Introduction to Parallel Programming*, Academic Press, 1989, ISBN: 0-12-128470-0.

DeRemer and Pennello Article

DeRemer, Frank and Pennello, Thomas J., *Efficient Computation of LALR(1) Look-Ahead Sets*, SigPlan Notices, Volume 15, No. 8, August 1979.

Draft ANSI X3J11.1

IEEE Floating Point draft report of ANSI X3J11.1 (NCEG).

FIPS 151-1

Federal Information Procurement Standard (FIPS) 151-1. Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].

FIPS 151-2

Federal Information Procurement Standards (FIPS) 151-2, Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].

HP-UX Manual

Hewlett-Packard HP-UX Release 9.0 Reference Manual, Third Edition, August 1992.

IEC 60559:1989

IEC 60559:1989, Binary Floating-Point Arithmetic for Microprocessor Systems (previously designated IEC 559:1989).

IEEE Std 754-1985

IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic.

IEEE Std 854-1987

IEEE Std 854-1987, IEEE Standard for Radix-Independent Floating-Point Arithmetic.

Referenced Documents

IEEE Std 1003.9-1992

IEEE Std 1003.9-1992, IEEE Standard for Information Technology — POSIX FORTRAN 77 Language Interfaces — Part 1: Binding for System Application Program Interface API.

IETF RFC 791

Internet Protocol, Version 4 (IPv4), September 1981.

IETF RFC 819

The Domain Naming Convention for Internet User Applications, Z. Su, J. Postel, August 1982.

IETF RFC 822

Standard for the Format of ARPA Internet Text Messages, D.H. Crocker, August 1982.

IETF RFC 919

Broadcasting Internet Datagrams, J. Mogul, October 1984.

IETF RFC 920

Domain Requirements, J. Postel, J. Reynolds, October 1984.

IETF RFC 921

Domain Name System Implementation Schedule, J. Postel, October 1984.

IETF RFC 922

Broadcasting Internet Datagrams in the Presence of Subnets, J. Mogul, October 1984.

IETF RFC 1034

Domain Names — Concepts and Facilities, P. Mockapetris, November 1987.

IETF RFC 1035

Domain Names — Implementation and Specification, P. Mockapetris, November 1987.

IETF RFC 1123

Requirements for Internet Hosts — Application and Support, R. Braden, October 1989.

IETF RFC 1886

DNS Extensions to Support Internet Protocol, Version 6 (IPv6), C. Huitema, S. Thomson, December 1995.

IETF RFC 2045

Multipurpose Internet Mail Extensions (MIME), Part 1: Format of Internet Message Bodies, N. Freed, N. Borenstein, November 1996.

IETF RFC 2181

Clarifications to the DNS Specification, R. Elz, R. Bush, July 1997.

IETF RFC 2373

Internet Protocol, Version 6 (IPv6) Addressing Architecture, S. Deering, R. Hinden, July 1998.

IETF RFC 2460

Internet Protocol, Version 6 (IPv6), S. Deering, R. Hinden, December 1998.

Internationalisation Guide

Guide, July 1993, Internationalisation Guide, Version 2 (ISBN: 1-859120-02-4, G304), published by The Open Group.

ISO C (1990)

ISO/IEC 9899:1990, Programming Languages — C, including Amendment 1:1995 (E), C Integrity (Multibyte Support Extensions (MSE) for ISO C).

ISO 2375:1985

ISO 2375:1985, Data Processing — Procedure for Registration of Escape Sequences.

ISO 8652:1987

ISO 8652:1987, Programming Languages — Ada (technically identical to ANSI standard 1815A-1983).

ISO/IEC 1539:1990

ISO/IEC 1539:1990, Information Technology — Programming Languages — Fortran (technically identical to the ANSI X3.9-1978 standard [FORTRAN 77]).

ISO/IEC 4873:1991

ISO/IEC 4873:1991, Information Technology — ISO 8-bit Code for Information Interchange — Structure and Rules for Implementation.

ISO/IEC 6429:1992

ISO/IEC 6429:1992, Information Technology — Control Functions for Coded Character Sets.

ISO/IEC 6937:1994

ISO/IEC 6937:1994, Information Technology — Coded Character Set for Text Communication — Latin Alphabet.

ISO/IEC 8802-3:1996

ISO/IEC 8802-3:1996, Information Technology — Telecommunications and Information Exchange Between Systems — Local and Metropolitan Area Networks — Specific Requirements — Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications.

ISO/IEC 8859

ISO/IEC 8859, Information Technology — 8-Bit Single-Byte Coded Graphic Character Sets:

Part 1: Latin Alphabet No. 1

Part 2: Latin Alphabet No. 2

Part 3: Latin Alphabet No. 3

Part 4: Latin Alphabet No. 4

Part 5: Latin/Cyrillic Alphabet

Part 6: Latin/Arabic Alphabet

Part 7: Latin/Greek Alphabet

Part 8: Latin/Hebrew Alphabet

Part 9: Latin Alphabet No. 5

Part 10: Latin Alphabet No. 6

Part 13: Latin Alphabet No. 7

Part 14: Latin Alphabet No. 8

Part 15: Latin Alphabet No. 9

ISO POSIX-1:1996

ISO/IEC 9945-1:1996, Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language] (identical to ANSI/IEEE Std 1003.1-1996). Incorporating ANSI/IEEE Stds 1003.1-1990, 1003.1b-1993, 1003.1c-1995, and 1003.1i-1995.

ISO POSIX-2:1993

ISO/IEC 9945-2:1993, Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities (identical to ANSI/IEEE Std 1003.2-1992, as amended by ANSI/IEEE Std 1003.2a-1992).

Referenced Documents

Issue 1

X/Open Portability Guide, July 1985 (ISBN: 0-444-87839-4).

Issue 2

X/Open Portability Guide, January 1987:

- Volume 1: XVS Commands and Utilities (ISBN: 0-444-70174-5)
- Volume 2: XVS System Calls and Libraries (ISBN: 0-444-70175-3)

Issue 3

X/Open Specification, 1988, 1989, February 1992:

- Commands and Utilities, Issue 3 (ISBN: 1-872630-36-7, C211); this specification was formerly X/Open Portability Guide, Issue 3, Volume 1, January 1989, XSI Commands and Utilities (ISBN: 0-13-685835-X, XO/XPG/89/002)
- System Interfaces and Headers, Issue 3 (ISBN: 1-872630-37-5, C212); this specification was formerly X/Open Portability Guide, Issue 3, Volume 2, January 1989, XSI System Interface and Headers (ISBN: 0-13-685843-0, XO/XPG/89/003)
- Curses Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapters 9 to 14 inclusive; this specification was formerly X/Open Portability Guide, Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)
- Headers Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapter 19, Cpio and Tar Headers; this specification was formerly X/Open Portability Guide Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)

Issue 4

CAE Specification, July 1992, published by The Open Group:

- System Interface Definitions (XBD), Issue 4 (ISBN: 1-872630-46-4, C204)
- Commands and Utilities (XCU), Issue 4 (ISBN: 1-872630-48-0, C203)
- System Interfaces and Headers (XSH), Issue 4 (ISBN: 1-872630-47-2, C202)

Issue 4, Version 2

CAE Specification, August 1994, published by The Open Group:

- System Interface Definitions (XBD), Issue 4, Version 2 (ISBN: 1-85912-036-9, C434)
- Commands and Utilities (XCU), Issue 4, Version 2 (ISBN: 1-85912-034-2, C436)
- System Interfaces and Headers (XSH), Issue 4, Version 2 (ISBN: 1-85912-037-7, C435)

Issue 5

Technical Standard, February 1997, published by The Open Group:

- System Interface Definitions (XBD), Issue 5 (ISBN: 1-85912-186-1, C605)
- Commands and Utilities (XCU), Issue 5 (ISBN: 1-85912-191-8, C604)
- System Interfaces and Headers (XSH), Issue 5 (ISBN: 1-85912-181-0, C606)

Knuth Article

Knuth, Donald E., *On the Translation of Languages from Left to Right*, Information and Control, Volume 8, No. 6, October 1965.

KornShell

Bolsky, Morris I. and Korn, David G., *The New KornShell Command and Programming Language*, March 1995, Prentice Hall.

MSE Working Draft

Working draft of ISO/IEC 9899:1990/Add3:Draft, Addendum 3 — Multibyte Support Extensions (MSE) as documented in the ISO Working Paper SC22/WG14/N205 dated 31 March 1992.

POSIX.0: 1995

IEEE Std 1003.0-1995, IEEE Guide to the POSIX Open System Environment (OSE) (identical to ISO/IEC TR 14252).

POSIX.1: 1988

IEEE Std 1003.1-1988, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language].

POSIX.1: 1990

IEEE Std 1003.1-1990, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language].

POSIX.1a

P1003.1a, Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — (C Language) Amendment.

POSIX.1d: 1999

IEEE Std 1003.1d-1999, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 4: Additional Realtime Extensions [C Language].

POSIX.1g: 2000

IEEE Std 1003.1g-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 6: Protocol-Independent Interfaces (PII).

POSIX.1j: 2000

IEEE Std 1003.1j-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 5: Advanced Realtime Extensions [C Language].

POSIX.1q: 2000

IEEE Std 1003.1q-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 7: Tracing [C Language].

POSIX.2b

P1003.2b, Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment.

POSIX.2d:-1994

IEEE Std 1003.2d-1994, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment 1: Batch Environment.

Referenced Documents

POSIX.13:-1998

IEEE Std 1003.13:1998, IEEE Standard for Information Technology — Standardized Application Environment Profile (AEP) — POSIX Realtime Application Support.

Sarwate Article

Sarwate, Dilip V., *Computation of Cyclic Redundancy Checks via Table Lookup*, Communications of the ACM, Volume 30, No. 8, August 1988.

Sprunt, Sha, and Lehoczky

Sprunt, B., Sha, L., and Lehoczky, J.P., *Aperiodic Task Scheduling for Hard Real-Time Systems*, The Journal of Real-Time Systems, Volume 1, 1989, Pages 27-60.

SVID, Issue 1

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 1; Morristown, NJ, UNIX Press, 1985.

SVID, Issue 2

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 2; Morristown, NJ, UNIX Press, 1986.

SVID, Issue 3

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 3; Morristown, NJ, UNIX Press, 1989.

The AWK Programming Language

Aho, Alfred V., Kernighan, Brian W., and Weinberger, Peter J., *The AWK Programming Language*, Reading, MA, Addison-Wesley 1988.

UNIX Programmer's Manual

American Telephone and Telegraph Company, *UNIX Time-Sharing System: UNIX Programmer's Manual*, 7th Edition, Murray Hill, NJ, Bell Telephone Laboratories, January 1979.

XNS, Issue 4

CAE Specification, August 1994, Networking Services, Issue 4 (ISBN: 1-85912-049-0, C438), published by The Open Group.

XNS, Issue 5

CAE Specification, February 1997, Networking Services, Issue 5 (ISBN: 1-85912-165-9, C523), published by The Open Group.

XNS, Issue 5.2

Technical Standard, January 2000, Networking Services (XNS), Issue 5.2 (ISBN: 1-85912-241-8, C808), published by The Open Group.

X/Open Curses, Issue 4, Version 2

CAE Specification, May 1996, X/Open Curses, Issue 4, Version 2 (ISBN: 1-85912-171-3, C610), published by The Open Group.

Yacc

Yacc: Yet Another Compiler Compiler, Stephen C. Johnson, 1978.

Source Documents

Parts of the following documents were used to create the base documents for this standard:

AIX 3.2 Manual

AIX Version 3.2 For RISC System/6000, Technical Reference: Base Operating System and Extensions, 1990, 1992 (Part No. SC23-2382-00).

OSF/1

OSF/1 Programmer's Reference, Release 1.2 (ISBN: 0-13-020579-6).

OSF AES

Application Environment Specification (AES) Operating System Programming Interfaces Volume, Revision A (ISBN: 0-13-043522-8).

System V Release 2.0

- UNIX System V Release 2.0 Programmer's Reference Manual (April 1984 - Issue 2).
- UNIX System V Release 2.0 Programming Guide (April 1984 - Issue 2).

System V Release 4.2

Operating System API Reference, UNIX SVR4.2 (1992) (ISBN: 0-13-017658-3).

Introduction

1.1 Scope

The scope of IEEE Std 1003.1-2001 is described in the Base Definitions volume of IEEE Std 1003.1-2001.

1.2 Conformance

Conformance requirements for IEEE Std 1003.1-2001 are defined in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 2, Conformance.

1.3 Normative References

Normative references for IEEE Std 1003.1-2001 are defined in the Base Definitions volume of IEEE Std 1003.1-2001.

1.4 Change History

Change history is described in the Rationale (Informative) volume of IEEE Std 1003.1-2001, and in the CHANGE HISTORY section of reference pages.

1.5 Terminology

This section appears in the Base Definitions volume of IEEE Std 1003.1-2001, but is repeated here for convenience:

For the purposes of IEEE Std 1003.1-2001, the following terminology definitions apply:

can

Describes a permissible optional feature or behavior available to the user or application. The feature or behavior is mandatory for an implementation that conforms to IEEE Std 1003.1-2001. An application can rely on the existence of the feature or behavior.

implementation-defined

Describes a value or behavior that is not defined by IEEE Std 1003.1-2001 but is selected by an implementor. The value or behavior may vary among implementations that conform to IEEE Std 1003.1-2001. An application should not rely on the existence of the value or behavior. An application that relies on such a value or behavior cannot be assured to be portable across conforming implementations.

The implementor shall document such a value or behavior so that it can be used correctly by an application.

legacy

Describes a feature or behavior that is being retained for compatibility with older applications, but which has limitations which make it inappropriate for developing portable

33 applications. New applications should use alternative means of obtaining equivalent
34 functionality.

35 **may**

36 Describes a feature or behavior that is optional for an implementation that conforms to
37 IEEE Std 1003.1-2001. An application should not rely on the existence of the feature or
38 behavior. An application that relies on such a feature or behavior cannot be assured to be
39 portable across conforming implementations.

40 To avoid ambiguity, the opposite of *may* is expressed as *need not*, instead of *may not*.

41 **shall**

42 For an implementation that conforms to IEEE Std 1003.1-2001, describes a feature or
43 behavior that is mandatory. An application can rely on the existence of the feature or
44 behavior.

45 For an application or user, describes a behavior that is mandatory.

46 **should**

47 For an implementation that conforms to IEEE Std 1003.1-2001, describes a feature or
48 behavior that is recommended but not mandatory. An application should not rely on the
49 existence of the feature or behavior. An application that relies on such a feature or behavior
50 cannot be assured to be portable across conforming implementations.

51 For an application, describes a feature or behavior that is recommended programming
52 practice for optimum portability.

53 **undefined**

54 Describes the nature of a value or behavior not defined by IEEE Std 1003.1-2001 which
55 results from use of an invalid program construct or invalid data input.

56 The value or behavior may vary among implementations that conform to
57 IEEE Std 1003.1-2001. An application should not rely on the existence or validity of the
58 value or behavior. An application that relies on any particular value or behavior cannot be
59 assured to be portable across conforming implementations.

60 **unspecified**

61 Describes the nature of a value or behavior not specified by IEEE Std 1003.1-2001 which
62 results from use of a valid program construct or valid data input.

63 The value or behavior may vary among implementations that conform to
64 IEEE Std 1003.1-2001. An application should not rely on the existence or validity of the
65 value or behavior. An application that relies on any particular value or behavior cannot be
66 assured to be portable across conforming implementations.

67 1.6 Definitions

68 Concepts and definitions are defined in the Base Definitions volume of IEEE Std 1003.1-2001.

69 1.7 Relationship to Other Formal Standards

70 Great care has been taken to ensure that this volume of IEEE Std 1003.1-2001 is fully aligned with
71 the following standards:

72 ISO C (1999)

73 ISO/IEC 9899: 1999, Programming Languages — C.

74 Parts of the ISO/IEC 9899: 1999 standard (hereinafter referred to as the ISO C standard) are
75 referenced to describe requirements also mandated by this volume of IEEE Std 1003.1-2001.
76 Some functions and headers included within this volume of IEEE Std 1003.1-2001 have a version
77 in the ISO C standard; in this case CX markings are added as appropriate to show where the
78 ISO C standard has been extended (see Section 1.8.1). Any conflict between this volume of
79 IEEE Std 1003.1-2001 and the ISO C standard is unintentional.

80 This volume of IEEE Std 1003.1-2001 also allows, but does not require, mathematics functions to
81 support IEEE Std 754-1985 and IEEE Std 854-1987.

82 1.8 Portability

83 Some of the utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 and functions in
84 the System Interfaces volume of IEEE Std 1003.1-2001 describe functionality that might not be
85 fully portable to systems meeting the requirements for POSIX conformance (see the Base
86 Definitions volume of IEEE Std 1003.1-2001, Chapter 2, Conformance).

87 Where optional, enhanced, or reduced functionality is specified, the text is shaded and a code in
88 the margin identifies the nature of the option, extension, or warning (see Section 1.8.1). For
89 maximum portability, an application should avoid such functionality.

90 1.8.1 Codes

91 Margin codes and their meanings are listed in the Base Definitions volume of
92 IEEE Std 1003.1-2001, but are repeated here for convenience:

93 ADV **Advisory Information**

94 The functionality described is optional. The functionality described is also an extension to the
95 ISO C standard.

96 Where applicable, functions are marked with the ADV margin legend in the SYNOPSIS section.
97 Where additional semantics apply to a function, the material is identified by use of the ADV
98 margin legend.

99 AIO **Asynchronous Input and Output**

100 The functionality described is optional. The functionality described is also an extension to the
101 ISO C standard.

102 Where applicable, functions are marked with the AIO margin legend in the SYNOPSIS section.
103 Where additional semantics apply to a function, the material is identified by use of the AIO
104 margin legend.

105 BAR **Barriers**

106 The functionality described is optional. The functionality described is also an extension to the

107		ISO C standard.
108		Where applicable, functions are marked with the BAR margin legend in the SYNOPSIS section.
109		Where additional semantics apply to a function, the material is identified by use of the BAR
110		margin legend.
111	BE	Batch Environment Services and Utilities
112		The functionality described is optional.
113		Where applicable, utilities are marked with the BE margin legend in the SYNOPSIS section.
114		Where additional semantics apply to a utility, the material is identified by use of the BE margin
115		legend.
116	CD	C-Language Development Utilities
117		The functionality described is optional.
118		Where applicable, utilities are marked with the CD margin legend in the SYNOPSIS section.
119		Where additional semantics apply to a utility, the material is identified by use of the CD margin
120		legend.
121	CPT	Process CPU-Time Clocks
122		The functionality described is optional. The functionality described is also an extension to the
123		ISO C standard.
124		Where applicable, functions are marked with the CPT margin legend in the SYNOPSIS section.
125		Where additional semantics apply to a function, the material is identified by use of the CPT
126		margin legend.
127	CS	Clock Selection
128		The functionality described is optional. The functionality described is also an extension to the
129		ISO C standard.
130		Where applicable, functions are marked with the CS margin legend in the SYNOPSIS section.
131		Where additional semantics apply to a function, the material is identified by use of the CS
132		margin legend.
133	CX	Extension to the ISO C standard
134		The functionality described is an extension to the ISO C standard. Application writers may make
135		use of an extension as it is supported on all IEEE Std 1003.1-2001-conforming systems.
136		With each function or header from the ISO C standard, a statement to the effect that “any
137		conflict is unintentional” is included. That is intended to refer to a direct conflict.
138		IEEE Std 1003.1-2001 acts in part as a profile of the ISO C standard, and it may choose to further
139		constrain behaviors allowed to vary by the ISO C standard. Such limitations are not considered
140		conflicts.
141		Where additional semantics apply to a function or header, the material is identified by use of the
142		CX margin legend.
143	FD	FORTRAN Development Utilities
144		The functionality described is optional.
145		Where applicable, utilities are marked with the FD margin legend in the SYNOPSIS section.
146		Where additional semantics apply to a utility, the material is identified by use of the FD margin
147		legend.
148	FR	FORTRAN Runtime Utilities
149		The functionality described is optional.

150 Where applicable, utilities are marked with the FR margin legend in the SYNOPSIS section.
 151 Where additional semantics apply to a utility, the material is identified by use of the FR margin
 152 legend.

153 FSC **File Synchronization**

154 The functionality described is optional. The functionality described is also an extension to the
 155 ISO C standard.

156 Where applicable, functions are marked with the FSC margin legend in the SYNOPSIS section.
 157 Where additional semantics apply to a function, the material is identified by use of the FSC
 158 margin legend.

159 IP6 **IPV6**

160 The functionality described is optional. The functionality described is also an extension to the
 161 ISO C standard.

162 Where applicable, functions are marked with the IP6 margin legend in the SYNOPSIS section.
 163 Where additional semantics apply to a function, the material is identified by use of the IP6
 164 margin legend.

165 MC1 **Advisory Information and either Memory Mapped Files or Shared Memory Objects**

166 The functionality described is optional. The functionality described is also an extension to the
 167 ISO C standard.

168 This is a shorthand notation for combinations of multiple option codes.

169 Where applicable, functions are marked with the MC1 margin legend in the SYNOPSIS section.
 170 Where additional semantics apply to a function, the material is identified by use of the MC1
 171 margin legend.

172 Refer to the Base Definitions volume of IEEE Std 1003.1-2001, Section 1.5.2, Margin Code
 173 Notation.

174 MC2 **Memory Mapped Files, Shared Memory Objects, or Memory Protection**

175 The functionality described is optional. The functionality described is also an extension to the
 176 ISO C standard.

177 This is a shorthand notation for combinations of multiple option codes.

178 Where applicable, functions are marked with the MC2 margin legend in the SYNOPSIS section.
 179 Where additional semantics apply to a function, the material is identified by use of the MC2
 180 margin legend.

181 Refer to the Base Definitions volume of IEEE Std 1003.1-2001, Section 1.5.2, Margin Code
 182 Notation.

183 MC3 **Memory Mapped Files, Shared Memory Objects, or Typed Memory Objects**

184 The functionality described is optional. The functionality described is also an extension to the
 185 ISO C standard.

186 This is a shorthand notation for combinations of multiple option codes.

187 Where applicable, functions are marked with the MC3 margin legend in the SYNOPSIS section.
 188 Where additional semantics apply to a function, the material is identified by use of the MC3
 189 margin legend.

190 Refer to the Base Definitions volume of IEEE Std 1003.1-2001, Section 1.5.2, Margin Code
 191 Notation.

192 MF **Memory Mapped Files**

193 The functionality described is optional. The functionality described is also an extension to the

194 ISO C standard.

195 Where applicable, functions are marked with the MF margin legend in the SYNOPSIS section.
196 Where additional semantics apply to a function, the material is identified by use of the MF
197 margin legend.

198 ML **Process Memory Locking**
199 The functionality described is optional. The functionality described is also an extension to the
200 ISO C standard.

201 Where applicable, functions are marked with the ML margin legend in the SYNOPSIS section.
202 Where additional semantics apply to a function, the material is identified by use of the ML
203 margin legend.

204 MLR **Range Memory Locking**
205 The functionality described is optional. The functionality described is also an extension to the
206 ISO C standard.

207 Where applicable, functions are marked with the MLR margin legend in the SYNOPSIS section.
208 Where additional semantics apply to a function, the material is identified by use of the MLR
209 margin legend.

210 MON **Monotonic Clock**
211 The functionality described is optional. The functionality described is also an extension to the
212 ISO C standard.

213 Where applicable, functions are marked with the MON margin legend in the SYNOPSIS section.
214 Where additional semantics apply to a function, the material is identified by use of the MON
215 margin legend.

216 MPR **Memory Protection**
217 The functionality described is optional. The functionality described is also an extension to the
218 ISO C standard.

219 Where applicable, functions are marked with the MPR margin legend in the SYNOPSIS section.
220 Where additional semantics apply to a function, the material is identified by use of the MPR
221 margin legend.

222 MSG **Message Passing**
223 The functionality described is optional. The functionality described is also an extension to the
224 ISO C standard.

225 Where applicable, functions are marked with the MSG margin legend in the SYNOPSIS section.
226 Where additional semantics apply to a function, the material is identified by use of the MSG
227 margin legend.

228 MX **IEC 60559 Floating-Point Option**
229 The functionality described is optional. The functionality described is also an extension to the
230 ISO C standard.

231 Where applicable, functions are marked with the MX margin legend in the SYNOPSIS section.
232 Where additional semantics apply to a function, the material is identified by use of the MX
233 margin legend.

234 OB **Obsolescent**
235 The functionality described may be withdrawn in a future version of this volume of
236 IEEE Std 1003.1-2001. Strictly Conforming POSIX Applications and Strictly Conforming XSI
237 Applications shall not use obsolescent features.

- 238 Where applicable, the material is identified by use of the OB margin legend.
- 239 OF **Output Format Incompletely Specified**
 240 The functionality described is an XSI extension. The format of the output produced by the utility
 241 is not fully specified. It is therefore not possible to post-process this output in a consistent
 242 fashion. Typical problems include unknown length of strings and unspecified field delimiters.
- 243 Where applicable, the material is identified by use of the OF margin legend.
- 244 OH **Optional Header**
 245 In the SYNOPSIS section of some interfaces in the System Interfaces volume of
 246 IEEE Std 1003.1-2001 an included header is marked as in the following example:
- 247 OH

```
#include <sys/types.h>  

  248 #include <grp.h>  

  249 struct group *getgrnam(const char *name);
```
- 250 The OH margin legend indicates that the marked header is not required on XSI-conformant
 251 systems.
- 252 PIO **Prioritized Input and Output**
 253 The functionality described is optional. The functionality described is also an extension to the
 254 ISO C standard.
- 255 Where applicable, functions are marked with the PIO margin legend in the SYNOPSIS section.
 256 Where additional semantics apply to a function, the material is identified by use of the PIO
 257 margin legend.
- 258 PS **Process Scheduling**
 259 The functionality described is optional. The functionality described is also an extension to the
 260 ISO C standard.
- 261 Where applicable, functions are marked with the PS margin legend in the SYNOPSIS section.
 262 Where additional semantics apply to a function, the material is identified by use of the PS
 263 margin legend.
- 264 RS **Raw Sockets**
 265 The functionality described is optional. The functionality described is also an extension to the
 266 ISO C standard.
- 267 Where applicable, functions are marked with the RS margin legend in the SYNOPSIS section.
 268 Where additional semantics apply to a function, the material is identified by use of the RS
 269 margin legend.
- 270 RTS **Realtime Signals Extension**
 271 The functionality described is optional. The functionality described is also an extension to the
 272 ISO C standard.
- 273 Where applicable, functions are marked with the RTS margin legend in the SYNOPSIS section.
 274 Where additional semantics apply to a function, the material is identified by use of the RTS
 275 margin legend.
- 276 SD **Software Development Utilities**
 277 The functionality described is optional.
- 278 Where applicable, utilities are marked with the SD margin legend in the SYNOPSIS section.
 279 Where additional semantics apply to a utility, the material is identified by use of the SD margin
 280 legend.

281	SEM	Semaphores
282		The functionality described is optional. The functionality described is also an extension to the
283		ISO C standard.
284		Where applicable, functions are marked with the SEM margin legend in the SYNOPSIS section.
285		Where additional semantics apply to a function, the material is identified by use of the SEM
286		margin legend.
287	SHM	Shared Memory Objects
288		The functionality described is optional. The functionality described is also an extension to the
289		ISO C standard.
290		Where applicable, functions are marked with the SHM margin legend in the SYNOPSIS section.
291		Where additional semantics apply to a function, the material is identified by use of the SHM
292		margin legend.
293	SIO	Synchronized Input and Output
294		The functionality described is optional. The functionality described is also an extension to the
295		ISO C standard.
296		Where applicable, functions are marked with the SIO margin legend in the SYNOPSIS section.
297		Where additional semantics apply to a function, the material is identified by use of the SIO
298		margin legend.
299	SPI	Spin Locks
300		The functionality described is optional. The functionality described is also an extension to the
301		ISO C standard.
302		Where applicable, functions are marked with the SPI margin legend in the SYNOPSIS section.
303		Where additional semantics apply to a function, the material is identified by use of the SPI
304		margin legend.
305	SPN	Spawn
306		The functionality described is optional. The functionality described is also an extension to the
307		ISO C standard.
308		Where applicable, functions are marked with the SPN margin legend in the SYNOPSIS section.
309		Where additional semantics apply to a function, the material is identified by use of the SPN
310		margin legend.
311	SS	Process Sporadic Server
312		The functionality described is optional. The functionality described is also an extension to the
313		ISO C standard.
314		Where applicable, functions are marked with the SS margin legend in the SYNOPSIS section.
315		Where additional semantics apply to a function, the material is identified by use of the SS
316		margin legend.
317	TCT	Thread CPU-Time Clocks
318		The functionality described is optional. The functionality described is also an extension to the
319		ISO C standard.
320		Where applicable, functions are marked with the TCT margin legend in the SYNOPSIS section.
321		Where additional semantics apply to a function, the material is identified by use of the TCT
322		margin legend.
323	TEF	Trace Event Filter
324		The functionality described is optional. The functionality described is also an extension to the
325		ISO C standard.

326 Where applicable, functions are marked with the TEF margin legend in the SYNOPSIS section.
327 Where additional semantics apply to a function, the material is identified by use of the TEF
328 margin legend.

329 THR **Threads**

330 The functionality described is optional. The functionality described is also an extension to the
331 ISO C standard.

332 Where applicable, functions are marked with the THR margin legend in the SYNOPSIS section.
333 Where additional semantics apply to a function, the material is identified by use of the THR
334 margin legend.

335 TMO **Timeouts**

336 The functionality described is optional. The functionality described is also an extension to the
337 ISO C standard.

338 Where applicable, functions are marked with the TMO margin legend in the SYNOPSIS section.
339 Where additional semantics apply to a function, the material is identified by use of the TMO
340 margin legend.

341 TMR **Timers**

342 The functionality described is optional. The functionality described is also an extension to the
343 ISO C standard.

344 Where applicable, functions are marked with the TMR margin legend in the SYNOPSIS section.
345 Where additional semantics apply to a function, the material is identified by use of the TMR
346 margin legend.

347 TPI **Thread Priority Inheritance**

348 The functionality described is optional. The functionality described is also an extension to the
349 ISO C standard.

350 Where applicable, functions are marked with the TPI margin legend in the SYNOPSIS section.
351 Where additional semantics apply to a function, the material is identified by use of the TPI
352 margin legend.

353 TPP **Thread Priority Protection**

354 The functionality described is optional. The functionality described is also an extension to the
355 ISO C standard.

356 Where applicable, functions are marked with the TPP margin legend in the SYNOPSIS section.
357 Where additional semantics apply to a function, the material is identified by use of the TPP
358 margin legend.

359 TPS **Thread Execution Scheduling**

360 The functionality described is optional. The functionality described is also an extension to the
361 ISO C standard.

362 Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.
363 Where additional semantics apply to a function, the material is identified by use of the TPS
364 margin legend.

365 TRC **Trace**

366 The functionality described is optional. The functionality described is also an extension to the
367 ISO C standard.

368 Where applicable, functions are marked with the TRC margin legend in the SYNOPSIS section.
369 Where additional semantics apply to a function, the material is identified by use of the TRC
370 margin legend.

371	TRI	Trace Inherit
372		The functionality described is optional. The functionality described is also an extension to the
373		ISO C standard.
374		Where applicable, functions are marked with the TRI margin legend in the SYNOPSIS section.
375		Where additional semantics apply to a function, the material is identified by use of the TRI
376		margin legend.
377	TRL	Trace Log
378		The functionality described is optional. The functionality described is also an extension to the
379		ISO C standard.
380		Where applicable, functions are marked with the TRL margin legend in the SYNOPSIS section.
381		Where additional semantics apply to a function, the material is identified by use of the TRL
382		margin legend.
383	TSA	Thread Stack Address Attribute
384		The functionality described is optional. The functionality described is also an extension to the
385		ISO C standard.
386		Where applicable, functions are marked with the TSA margin legend for the SYNOPSIS section.
387		Where additional semantics apply to a function, the material is identified by use of the TSA
388		margin legend.
389	TSF	Thread-Safe Functions
390		The functionality described is optional. The functionality described is also an extension to the
391		ISO C standard.
392		Where applicable, functions are marked with the TSF margin legend in the SYNOPSIS section.
393		Where additional semantics apply to a function, the material is identified by use of the TSF
394		margin legend.
395	TSH	Thread Process-Shared Synchronization
396		The functionality described is optional. The functionality described is also an extension to the
397		ISO C standard.
398		Where applicable, functions are marked with the TSH margin legend in the SYNOPSIS section.
399		Where additional semantics apply to a function, the material is identified by use of the TSH
400		margin legend.
401	TSP	Thread Sporadic Server
402		The functionality described is optional. The functionality described is also an extension to the
403		ISO C standard.
404		Where applicable, functions are marked with the TSP margin legend in the SYNOPSIS section.
405		Where additional semantics apply to a function, the material is identified by use of the TSP
406		margin legend.
407	TSS	Thread Stack Size Attribute
408		The functionality described is optional. The functionality described is also an extension to the
409		ISO C standard.
410		Where applicable, functions are marked with the TSS margin legend in the SYNOPSIS section.
411		Where additional semantics apply to a function, the material is identified by use of the TSS
412		margin legend.
413	TYM	Typed Memory Objects
414		The functionality described is optional. The functionality described is also an extension to the
415		ISO C standard.

416 Where applicable, functions are marked with the TYM margin legend in the SYNOPSIS section.
 417 Where additional semantics apply to a function, the material is identified by use of the TYM
 418 margin legend.

419 UP **User Portability Utilities**
 420 The functionality described is optional.

421 Where applicable, utilities are marked with the UP margin legend in the SYNOPSIS section.
 422 Where additional semantics apply to a utility, the material is identified by use of the UP margin
 423 legend.

424 XSI **Extension**
 425 The functionality described is an XSI extension. Functionality marked XSI is also an extension to
 426 the ISO C standard. Application writers may confidently make use of an extension on all
 427 systems supporting the X/Open System Interfaces Extension.

428 If an entire SYNOPSIS section is shaded and marked XSI, all the functionality described in that
 429 reference page is an extension. See the Base Definitions volume of IEEE Std 1003.1-2001, Section
 430 3.439, XSI.

431 XSR **XSI STREAMS**
 432 The functionality described is optional. The functionality described is also an extension to the
 433 ISO C standard.

434 Where applicable, functions are marked with the XSR margin legend in the SYNOPSIS section.
 435 Where additional semantics apply to a function, the material is identified by use of the XSR
 436 margin legend.

437 1.9 Format of Entries

438 The entries in Chapter 3 are based on a common format as follows. The only sections relating to
 439 conformance are the SYNOPSIS, DESCRIPTION, RETURN VALUE, and ERRORS sections.

440 NAME

441 This section gives the name or names of the entry and briefly states its purpose.

442 SYNOPSIS

443 This section summarizes the use of the entry being described. If it is necessary to
 444 include a header to use this function, the names of such headers are shown, for
 445 example:

```
446 #include <stdio.h>
```

447 DESCRIPTION

448 This section describes the functionality of the function or header.

449 RETURN VALUE

450 This section indicates the possible return values, if any.

451 If the implementation can detect errors, “successful completion” means that no error
 452 has been detected during execution of the function. If the implementation does detect
 453 an error, the error is indicated.

454 For functions where no errors are defined, “successful completion” means that if the
 455 implementation checks for errors, no error has been detected. If the implementation can
 456 detect errors, and an error is detected, the indicated return value is returned and *errno*
 457 may be set.

458 ERRORS

459 This section gives the symbolic names of the error values returned by a function or
460 stored into a variable accessed through the symbol *errno* if an error occurs.

461 “No errors are defined” means that error values returned by a function or stored into a
462 variable accessed through the symbol *errno*, if any, depend on the implementation.

463 EXAMPLES

464 This section is informative.

465 This section gives examples of usage, where appropriate. In the event of conflict
466 between an example and a normative part of this volume of IEEE Std 1003.1-2001, the
467 normative material is to be taken as correct.

468 APPLICATION USAGE

469 This section is informative.

470 This section gives warnings and advice to application writers about the entry. In the
471 event of conflict between warnings and advice and a normative part of this volume of
472 IEEE Std 1003.1-2001, the normative material is to be taken as correct.

473 RATIONALE

474 This section is informative.

475 This section contains historical information concerning the contents of this volume of
476 IEEE Std 1003.1-2001 and why features were included or discarded by the standard
477 developers.

478 FUTURE DIRECTIONS

479 This section is informative.

480 This section provides comments which should be used as a guide to current thinking;
481 there is not necessarily a commitment to adopt these future directions.

482 SEE ALSO

483 This section is informative.

484 This section gives references to related information.

485 CHANGE HISTORY

486 This section is informative.

487 This section shows the derivation of the entry and any significant changes that have
488 been made to it.

General Information

489

490 This chapter covers information that is relevant to all the functions specified in Chapter 3 and
491 the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers.

492 2.1 Use and Implementation of Functions

493 Each of the following statements shall apply unless explicitly stated otherwise in the detailed
494 descriptions that follow:

- 495 1. If an argument to a function has an invalid value (such as a value outside the domain of
496 the function, or a pointer outside the address space of the program, or a null pointer), the
497 behavior is undefined.
- 498 2. Any function declared in a header may also be implemented as a macro defined in the
499 header, so a function should not be declared explicitly if its header is included. Any macro
500 definition of a function can be suppressed locally by enclosing the name of the function in
501 parentheses, because the name is then not followed by the left parenthesis that indicates
502 expansion of a macro function name. For the same syntactic reason, it is permitted to take
503 the address of a function even if it is also defined as a macro. The use of the C-language
504 **#undef** construct to remove any such macro definition shall also ensure that an actual
505 function is referred to.
- 506 3. Any invocation of a function that is implemented as a macro shall expand to code that
507 evaluates each of its arguments exactly once, fully protected by parentheses where
508 necessary, so it is generally safe to use arbitrary expressions as arguments. Likewise, those
509 function-like macros described in the following sections may be invoked in an expression
510 anywhere a function with a compatible return type could be called.
- 511 4. Provided that a function can be declared without reference to any type defined in a header,
512 it is also permissible to declare the function explicitly and use it without including its
513 associated header.
- 514 5. If a function that accepts a variable number of arguments is not declared (explicitly or by
515 including its associated header), the behavior is undefined.

516 2.2 The Compilation Environment

517 2.2.1 POSIX.1 Symbols

518 Certain symbols in this volume of IEEE Std 1003.1-2001 are defined in headers (see the Base
519 Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers). Some of those headers could
520 also define symbols other than those defined by IEEE Std 1003.1-2001, potentially conflicting
521 with symbols used by the application. Also, IEEE Std 1003.1-2001 defines symbols that are not
522 permitted by other standards to appear in those headers without some control on the visibility
523 of those symbols.

524 Symbols called “feature test macros” are used to control the visibility of symbols that might be
525 included in a header. Implementations, future versions of IEEE Std 1003.1-2001, and other
526 standards may define additional feature test macros.

527 In the compilation of an application that **#defines** a feature test macro specified by
 528 IEEE Std 1003.1-2001, no header defined by IEEE Std 1003.1-2001 shall be included prior to the
 529 definition of the feature test macro. This restriction also applies to any implementation-
 530 provided header in which these feature test macros are used. If the definition of the macro does
 531 not precede the **#include**, the result is undefined.

532 Feature test macros shall begin with the underscore character ('_').

533 2.2.1.1 *The `_POSIX_C_SOURCE` Feature Test Macro*

534 A POSIX-conforming application should ensure that the feature test macro `_POSIX_C_SOURCE`
 535 is defined before inclusion of any header.

536 When an application includes a header described by IEEE Std 1003.1-2001, and when this feature
 537 test macro is defined to have the value 200112L:

- 538 1. All symbols required by IEEE Std 1003.1-2001 to appear when the header is included shall
 539 be made visible.
- 540 2. Symbols that are explicitly permitted, but not required, by IEEE Std 1003.1-2001 to appear
 541 in that header (including those in reserved name spaces) may be made visible.
- 542 3. Additional symbols not required or explicitly permitted by IEEE Std 1003.1-2001 to be in
 543 that header shall not be made visible, except when enabled by another feature test macro.

544 Identifiers in IEEE Std 1003.1-2001 may only be undefined using the **#undef** directive as
 545 described in Section 2.1 (on page 13) or Section 2.2.2. These **#undef** directives shall follow all
 546 **#include** directives of any header in IEEE Std 1003.1-2001.

547 **Note:** The POSIX.1-1990 standard specified a macro called `_POSIX_SOURCE`. This has been
 548 superseded by `_POSIX_C_SOURCE`.

549 2.2.1.2 *The `_XOPEN_SOURCE` Feature Test Macro*

550 XSI An XSI-conforming application should ensure that the feature test macro `_XOPEN_SOURCE` is
 551 defined with the value 600 before inclusion of any header. This is needed to enable the
 552 functionality described in Section 2.2.1.1 and in addition to enable the XSI extension.

553 Since this volume of IEEE Std 1003.1-2001 is aligned with the ISO C standard, and since all
 554 functionality enabled by `_POSIX_C_SOURCE` set equal to 200112L is enabled by
 555 `_XOPEN_SOURCE` set equal to 600, there should be no need to define `_POSIX_C_SOURCE` if
 556 `_XOPEN_SOURCE` is so defined. Therefore, if `_XOPEN_SOURCE` is set equal to 600 and
 557 `_POSIX_C_SOURCE` is set equal to 200112L, the behavior is the same as if only
 558 `_XOPEN_SOURCE` is defined and set equal to 600. However, should `_POSIX_C_SOURCE` be set
 559 to a value greater than 200112L, the behavior is unspecified.

560 2.2.2 The Name Space

561 All identifiers in this volume of IEEE Std 1003.1-2001, except *environ*, are defined in at least one
 562 of the headers, as shown in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13,
 563 XSI Headers. When `_XOPEN_SOURCE` or `_POSIX_C_SOURCE` is defined, each header defines or
 564 declares some identifiers, potentially conflicting with identifiers used by the application. The set
 565 of identifiers visible to the application consists of precisely those identifiers from the header
 566 pages of the included headers, as well as additional identifiers reserved for the implementation.
 567 In addition, some headers may make visible identifiers from other headers as indicated on the
 568 relevant header pages.

569 Implementations may also add members to a structure or union without controlling the
570 visibility of those members with a feature test macro, as long as a user-defined macro with the
571 same name cannot interfere with the correct interpretation of the program. The identifiers
572 reserved for use by the implementation are described below:

- 573 1. Each identifier with external linkage described in the header section is reserved for use as
574 an identifier with external linkage if the header is included.
- 575 2. Each macro described in the header section is reserved for any use if the header is
576 included.
- 577 3. Each identifier with file scope described in the header section is reserved for use as an
578 identifier with file scope in the same name space if the header is included.

579 The prefixes `posix_`, `POSIX_`, and `_POSIX_` are reserved for use by IEEE Std 1003.1-2001 and
580 other POSIX standards. Implementations may add symbols to the headers shown in the
581 following table, provided the identifiers for those symbols begin with the corresponding
582 reserved prefixes in the following table, and do not use the reserved prefixes `posix_`, `POSIX_`, or
583 `_POSIX_`.

632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651

XSI

XSI

TMR
TMR
XSI
XSI

XSI

Header	Prefix	Suffix	Complete Name
<sys/uio.h>	iov_		UIO_MAXIOV
<sys/un.h>	sun_		
<sys/utsname.h>	uts_		
<sys/wait.h>	si_, W[A-Z], P_		
<termios.h>	c_		
<time.h>	tm_		
	clock_, timer_, it_, tv_, CLOCK_, TIMER_		
<ucontext.h>	uc_, ss_		
<ulimit.h>	UL_		
<utime.h>	utim_		
<utmpx.h>	ut_	_LVL, _TIME, _PROCESS	
<wchar.h>	wcs[a-z]		
<wctype.h>	is[a-z], to[a-z]		
<wordexp.h>	we_		
ANY header		_t	

652
653
654
655

656
657
658
659

Note: The notation [A–Z] indicates any uppercase letter in the portable character set. The notation [a–z] indicates any lowercase letter in the portable character set. Commas and spaces in the lists of prefixes and complete names in the above table are not part of any prefix or complete name.

If any header in the following table is included, macros with the prefixes shown may be defined. After the last inclusion of a given header, an application may use identifiers with the corresponding prefixes for its own purpose, provided their use is preceded by a **#undef** of the corresponding macro.

	Header	Prefix
660		
661		
662	XSI <dlfcn.h>	RTLD_
663	<fcntl.h>	F_, O_, S_
664	XSI <fmtmsg.h>	MM_
665	<fnmatch.h>	FNM_
666	XSI <ftw.h>	FTW_
667	<glob.h>	GLOB_
668	<inttypes.h>	PRI[Xa-z], SCN[Xa-z]
669	XSI <ndbm.h>	DBM_
670	<net/if.h>	IF_
671	<netinet/in.h>	IMPLINK_, IN_, INADDR_, IP_, IPPORT_, IPPROTO_, SOCK_
672	IP6	IPV6_, IN6_
673	<netinet/tcp.h>	TCP_
674	XSI <nl_types.h>	NL_
675	XSI <poll.h>	POLL_
676	<regex.h>	REG_
677	<signal.h>	SA_, SIG_[0-9a-z_],
678	XSI	BUS_, CLD_, FPE_, ILL_, POLL_, SEGV_, SI_, SS_, SV_, TRAP_
679	XSI <stropts.h>	FLUSH[A-Z], I_, M_, MUXID_R[A-Z], S_, SND[A-Z], STR_
680	XSI <syslog.h>	LOG_
681	XSI <sys/ipc.h>	IPC_
682	XSI <sys/mman.h>	PROT_, MAP_, MS_
683	XSI <sys/msg.h>	MSG[A-Z]
684	XSI <sys/resource.h>	PRIO_, RLIM_, RLIMIT_, RUSAGE_
685	XSI <sys/sem.h>	SEM_
686	XSI <sys/shm.h>	SHM[A-Z], SHM_[A-Z]
687	XSI <sys/socket.h>	AF_, CMSG_, MSG_, PF_, SCM_, SHUT_, SO_
688	<sys/stat.h>	S_
689	XSI <sys/statvfs.h>	ST_
690	XSI <sys/time.h>	FD_, ITIMER_
691	XSI <sys/uio.h>	IOV_
692	XSI <sys/wait.h>	BUS_, CLD_, FPE_, ILL_, POLL_, SEGV_, SI_, TRAP_
693	<termios.h>	V, I, O, TC, B[0-9] (See below.)
694	<wordexp.h>	WRDE_

695 The following are used to reserve complete names for the <stdint.h> header:

```

696     INT[0-9A-Za-z-]*_MIN
697     INT[0-9A-Za-z-]*_MAX
698     INT[0-9A-Za-z-]*_C
699     UINT[0-9A-Za-z-]*_MIN
700     UINT[0-9A-Za-z-]*_MAX
701     UINT[0-9A-Za-z-]*_C

```

702 **Note:** The notation [0-9] indicates any digit. The notation [A-Z] indicates any uppercase letter in the
703 portable character set. The notation [0-9a-z_] indicates any digit, any lowercase letter in the
704 portable character set, or underscore.

705 The following reserved names are used as exact matches for <termios.h>:

706	XSI	CBAUD	EXTB	VDSUSP
707		DEFECHO	FLUSHO	VLNEXT
708		ECHOCTL	LOBLK	VREPRINT
709		ECHOKE	PENDIN	VSTATUS
710		ECHOPRT	SWTCH	VWERASE
711		EXTA	VDISCARD	

712 The following identifiers are reserved regardless of the inclusion of headers:

- 713 1. All identifiers that begin with an underscore and either an uppercase letter or another
714 underscore are always reserved for any use by the implementation.
- 715 2. All identifiers that begin with an underscore are always reserved for use as identifiers with
716 file scope in both the ordinary identifier and tag name spaces.
- 717 3. All identifiers in the table below are reserved for use as identifiers with external linkage.
718 Some of these identifiers do not appear in this volume of IEEE Std 1003.1-2001, but are
719 reserved for future use by the ISO C standard.

720	_Exit	ccoshf	csqrt	fputc	lrintl	sinh
721	abort	ccoshl	csqrtf	fputs	lround	sinhf
722	abs	ccosl	csqrtl	fputwc	lroundf	sinhl
723	acos	ceil	ctan	fputws	lroundl	sinl
724	acosf	ceilf	ctanf	fread	malloc	sprintf
725	acosh	ceilf	ctanl	free	mblen	sqrt
726	acoshf	ceilf	ctgamma	freopen	mbrlen	sqrtf
727	acoshl	ceilf	ctgammaf	frexp	mbrtowc	sqrtl
728	acosl	cerf	ctgammaf	frexpf	mbsinit	srand
729	acosl	cerfc	ltime	frexpl	mbsrtowcs	sscanf
730	asctime	cerfcf	difftime	fscanf	mbstowcs	str[a-z]*
731	asin	cerfcl	div	fseek	mbtowc	strtof
732	asinf	cerfff	erfcf	fsetpos	mem[a-z]*	strtoimax
733	asinh	cerfl	erfcl	ftell	mktime	strtold
734	asinhf	cexmp1	erff	fwide	modf	strtoll
735	asinhf	cexmp1f	erfl	fwprintf	modff	strtoull
736	asinl	cexmp1l	errno	fwrite	modfl	strtoumax
737	asinl	cexp	exit	fwscanf	nan	swprintf
738	atan	cexp2	exp	getc	nanf	swscanf
739	atan2	cexp2f	exp2	getchar	nanl	system
740	atan2f	cexp2l	exp2f	getenv	nearbyint	tan
741	atan2l	cexpf	exp2l	gets	nearbyintf	tanf
742	atanf	cexpl	expf	getwc	nearbyintl	tanh
743	atanf	cimag	expl	getwchar	nextafterf	tanhf
744	atanh	cimagf	expm1	gmtime	nextafterl	tanhf
745	atanh	cimagl	expm1f	hypotf	nexttoward	tanl
746	atanhf	clearerr	expm1l	hypotl	nexttowardf	tgamma
747	atanhl	clgamma	fabs	ilogb	nexttowardl	tgammaf
748	atanl	clgammaf	fabsf	ilogbf	perror	tgammaf
749	atanl	clgammaf	fabsf	ilogbf	pow	time
749	atanl	clgammaf	fabsl	ilogbl	pow	time
750	atexit	clock	fclose	imaxabs	powf	tmpfile

751	atof	clog	fdim	imaxdiv	powl	tmpnam
752	atoi	clog10	fdimf	is[a-z]*	printf	to[a-z]*
753	atol	clog10f	fdiml	isblank	putc	trunc
754	atoll	clog10l	feclearexcept	iswblank	putchar	truncf
755	bsearch	clog1p	fegetenv	labs	puts	truncl
756	cabs	clog1pf	fegetexceptflag	ldexp	putwc	ungetc
757	cabsf	clog1pl	fegetround	ldexpf	putwchar	ungetwc
758	cabsl	clog2	feholdexcept	ldexpl	qsort	va_end
759	cacos	clog2f	feof	ldiv	raise	vfprintf
760	cacosf	clog2l	feraiseexcept	ldiv	rand	vfscanf
761	cacosh	clogf	ferror	lgammaf	realloc	vwprintf
762	cacoshf	clogl	fesetenv	lgammal	remainderf	vwscanf
763	cacoshl	conj	fesetexceptflag	llabs	remainderl	vprintf
764	cacosl	conjf	fesetround	llrint	remove	vscanf
765	calloc	conjl	fetestexcept	llrintf	remquo	vsprintf
766	carg	copysign	feupdateenv	llrintl	remquof	vsscanf
767	cargf	copysignf	fflush	llround	remquol	vswprintf
768	cargl	copysignl	fgetc	llroundf	rename	vswscanf
769	casin	cos	fgetpos	llroundl	rewind	vwprintf
770	casinf	cosf	fgets	localeconv	rint	vwscanf
771	casinh	cosh	fgetwc	localtime	rintf	wcrtomb
772	casinhf	coshf	fgetws	log	rintl	wcs[a-z]*
773	casinhl	coshl	floor	log10	round	wcstof
774	casinl	cosl	floorf	log10f	roundf	wcstoimax
775	catan	cpow	floorl	log10l	roundl	wcstold
776	catanf	cpowf	fma	log1p	scalbln	wcstoll
777	catanh	cpowl	fmaf	log1pf	scalblnf	wcstoull
778	catanh	cproj	fmal	log1pl	scalblnl	wcstoumax
779	catanhf	cprojf	fmax	log2	scalbn	wctob
780	catanhf	cprojl	fmaxf	log2f	scalbnf	wctomb
781	catanhl	creal	fmaxl	log2l	scalbnl	wctrans
782	catanhl	crealf	fmin	logb	scanf	wctype
783	catanl	creall	fminf	logbf	setbuf	wcwidth
784	cbirt	csin	fminl	logbl	setjmp	wmem[a-z]*
785	cbirtf	csinf	fmod	logf	setlocale	wprintf
786	cbirtl	csinh	fmodf	logl	setvbuf	wscanf
787	ccos	csinhf	fmodl	longjmp	signal	
788	ccosf	csinhl	fopen	lrint	sin	
789	ccosh	csinl	fprintf	lrintf	sinf	

790 **Note:** The notation [a-z] indicates any lowercase letter in the portable character set. The
 791 notation ' * ' indicates any combination of digits, letters in the portable character set, or
 792 underscore.

793 4. All functions and external identifiers defined in the Base Definitions volume of
 794 IEEE Std 1003.1-2001, Chapter 13, Headers are reserved for use as identifiers with external
 795 linkage.

796 5. All the identifiers defined in this volume of IEEE Std 1003.1-2001 that have external linkage
 797 are always reserved for use as identifiers with external linkage.

798 No other identifiers are reserved.

799 Applications shall not declare or define identifiers with the same name as an identifier reserved
 800 in the same context. Since macro names are replaced whenever found, independent of scope and

801 name space, macro names matching any of the reserved identifier names shall not be defined by
802 an application if any associated header is included.

803 Except that the effect of each inclusion of `<assert.h>` depends on the definition of `NDEBUG`,
804 headers may be included in any order, and each may be included more than once in a given
805 scope, with no difference in effect from that of being included only once.

806 If used, the application shall ensure that a header is included outside of any external declaration
807 or definition, and it shall be first included before the first reference to any type or macro it
808 defines, or to any function or object it declares. However, if an identifier is declared or defined in
809 more than one header, the second and subsequent associated headers may be included after the
810 initial reference to the identifier. Prior to the inclusion of a header, the application shall not
811 define any macros with names lexically identical to symbols defined by that header.

812 **2.3 Error Numbers**

813 Most functions can provide an error number. The means by which each function provides its
814 error numbers is specified in its description.

815 Some functions provide the error number in a variable accessed through the symbol `errno`. The
816 symbol `errno`, defined by including the `<errno.h>` header, expands to a modifiable lvalue of type
817 `int`. It is unspecified whether `errno` is a macro or an identifier declared with external linkage. If a
818 macro definition is suppressed in order to access an actual object, or a program defines an
819 identifier with the name `errno`, the behavior is undefined.

820 The value of `errno` should only be examined when it is indicated to be valid by a function's return
821 value. No function in this volume of IEEE Std 1003.1-2001 shall set `errno` to zero. For each thread
822 of a process, the value of `errno` shall not be affected by function calls or assignments to `errno` by
823 other threads.

824 Some functions return an error number directly as the function value. These functions return a
825 value of zero to indicate success.

826 If more than one error occurs in processing a function call, any one of the possible errors may be
827 returned, as the order of detection is undefined.

828 Implementations may support additional errors not included in this list, may generate errors
829 included in this list under circumstances other than those described here, or may contain
830 extensions or limitations that prevent some errors from occurring. The ERRORS section on each
831 reference page specifies whether an error shall be returned, or whether it may be returned.
832 Implementations shall not generate a different error number from the ones described here for
833 error conditions described in this volume of IEEE Std 1003.1-2001, but may generate additional
834 errors unless explicitly disallowed for a particular function.

835 Each implementation shall document, in the conformance document, situations in which each of
836 the optional conditions defined in IEEE Std 1003.1-2001 is detected. The conformance document
837 may also contain statements that one or more of the optional error conditions are not detected.

838 For functions under the Threads option for which `[EINTR]` is not listed as a possible error
839 condition in this volume of IEEE Std 1003.1-2001, an implementation shall not return an error
840 code of `[EINTR]`.

841 The following symbolic names identify the possible error numbers, in the context of the
842 functions specifically defined in this volume of IEEE Std 1003.1-2001; these general descriptions
843 are more precisely defined in the ERRORS sections of the functions that return them. Only these
844 symbolic names should be used in programs, since the actual value of the error number is

845 unspecified. All values listed in this section shall be unique integer constant expressions with
 846 type **int** suitable for use in **#if** preprocessing directives, except as noted below. The values for all
 847 these names shall be found in the **<errno.h>** header defined in the Base Definitions volume of
 848 IEEE Std 1003.1-2001. The actual values are unspecified by this volume of IEEE Std 1003.1-2001.

849 [E2BIG]

850 Argument list too long. The sum of the number of bytes used by the new process image's
 851 argument list and environment list is greater than the system-imposed limit of {ARG_MAX}
 852 bytes.

853 or:

854 Lack of space in an output buffer.

855 or:

856 Argument is greater than the system-imposed maximum.

857 [EACCES]

858 Permission denied. An attempt was made to access a file in a way forbidden by its file
 859 access permissions.

860 [EADDRINUSE]

861 Address in use. The specified address is in use.

862 [EADDRNOTAVAIL]

863 Address not available. The specified address is not available from the local system.

864 [EAFNOSUPPORT]

865 Address family not supported. The implementation does not support the specified address
 866 family, or the specified address is not a valid address for the address family of the specified
 867 socket.

868 [EAGAIN]

869 Resource temporarily unavailable. This is a temporary condition and later calls to the same
 870 routine may complete normally.

871 [EALREADY]

872 Connection already in progress. A connection request is already in progress for the specified
 873 socket.

874 [EBADF]

875 Bad file descriptor. A file descriptor argument is out of range, refers to no open file, or a
 876 read (write) request is made to a file that is only open for writing (reading).

877 [EBADMSG]

878 XSR Bad message. During a *read()*, *getmsg()*, *getpmsg()*, or *ioctl()* I_RECVFD request to a
 879 STREAMS device, a message arrived at the head of the STREAM that is inappropriate for
 880 the function receiving the message.

881 *read()* Message waiting to be read on a STREAM is not a data message.

882 *getmsg()* or *getpmsg()*

883 A file descriptor was received instead of a control message.

884 *ioctl()* Control or data information was received instead of a file descriptor when
 885 I_RECVFD was specified.

886 or:

887	Bad Message. The implementation has detected a corrupted message.
888	[EBUSY]
889	Resource busy. An attempt was made to make use of a system resource that is not currently
890	available, as it is being used by another process in a manner that would have conflicted with
891	the request being made by this process.
892	[ECANCELED]
893	Operation canceled. The associated asynchronous operation was canceled before
894	completion.
895	[ECHILD]
896	No child process. A <i>wait()</i> or <i>waitpid()</i> function was executed by a process that had no
897	existing or unwaited-for child process.
898	[ECONNABORTED]
899	Connection aborted. The connection has been aborted.
900	[ECONNREFUSED]
901	Connection refused. An attempt to connect to a socket was refused because there was no
902	process listening or because the queue of connection requests was full and the underlying
903	protocol does not support retransmissions.
904	[ECONNRESET]
905	Connection reset. The connection was forcibly closed by the peer.
906	[EDEADLK]
907	Resource deadlock would occur. An attempt was made to lock a system resource that
908	would have resulted in a deadlock situation.
909	[EDESTADDRREQ]
910	Destination address required. No bind address was established.
911	[EDOM]
912	Domain error. An input argument is outside the defined domain of the mathematical
913	function (defined in the ISO C standard).
914	[EDQUOT]
915	Reserved.
916	[EEXIST]
917	File exists. An existing file was mentioned in an inappropriate context; for example, as a
918	new link name in the <i>link()</i> function.
919	[EFAULT]
920	Bad address. The system detected an invalid address in attempting to use an argument of a
921	call. The reliable detection of this error cannot be guaranteed, and when not detected may
922	result in the generation of a signal, indicating an address violation, which is sent to the
923	process.
924	[EFBIG]
925	File too large. The size of a file would exceed the maximum file size of an implementation or
926	offset maximum established in the corresponding file description.
927	[EHOSTUNREACH]
928	Host is unreachable. The destination host cannot be reached (probably because the host is
929	down or a remote router cannot reach it).
930	[EIDRM]
931	Identifier removed. Returned during XSI interprocess communication if an identifier has

932	been removed from the system.
933	[EILSEQ]
934	Illegal byte sequence. A wide-character code has been detected that does not correspond to
935	a valid character, or a byte sequence does not form a valid wide-character code (defined in
936	the ISO C standard).
937	[EINPROGRESS]
938	Operation in progress. This code is used to indicate that an asynchronous operation has not
939	yet completed.
940	or:
941	O_NONBLOCK is set for the socket file descriptor and the connection cannot be
942	immediately established.
943	[EINTR]
944	Interrupted function call. An asynchronous signal was caught by the process during the
945	execution of an interruptible function. If the signal handler performs a normal return, the
946	interrupted function call may return this condition (see the Base Definitions volume of
947	IEEE Std 1003.1-2001, <signal.h>).
948	[EINVAL]
949	Invalid argument. Some invalid argument was supplied; for example, specifying an
950	undefined signal in a <i>signal()</i> function or a <i>kill()</i> function.
951	[EIO]
952	Input/output error. Some physical input or output error has occurred. This error may be
953	reported on a subsequent operation on the same file descriptor. Any other error-causing
954	operation on the same file descriptor may cause the [EIO] error indication to be lost.
955	[EISCONN]
956	Socket is connected. The specified socket is already connected.
957	[EISDIR]
958	Is a directory. An attempt was made to open a directory with write mode specified.
959	[ELOOP]
960	Symbolic link loop. A loop exists in symbolic links encountered during pathname
961	resolution. This error may also be returned if more than {SYMLOOP_MAX} symbolic links
962	are encountered during pathname resolution.
963	[EMFILE]
964	Too many open files. An attempt was made to open more than the maximum number of file
965	descriptors allowed in this process.
966	[EMLINK]
967	Too many links. An attempt was made to have the link count of a single file exceed
968	{LINK_MAX}.
969	[EMSGSIZE]
970	Message too large. A message sent on a transport provider was larger than an internal
971	message buffer or some other network limit.
972	or:
973	Inappropriate message buffer length.
974	[EMULTIHOP]
975	Reserved.

976	[ENAMETOOLONG]
977	Filename too long. The length of a pathname exceeds {PATH_MAX}, or a pathname
978	component is longer than {NAME_MAX}. This error may also occur when pathname
979	substitution, as a result of encountering a symbolic link during pathname resolution, results
980	in a pathname string the size of which exceeds {PATH_MAX}.
981	[ENETDOWN]
982	Network is down. The local network interface used to reach the destination is down.
983	[ENETRESET]
984	The connection was aborted by the network.
985	[ENETUNREACH]
986	Network unreachable. No route to the network is present.
987	[ENFILE]
988	Too many files open in system. Too many files are currently open in the system. The system
989	has reached its predefined limit for simultaneously open files and temporarily cannot accept
990	requests to open another one.
991	[ENOBUFS]
992	No buffer space available. Insufficient buffer resources were available in the system to
993	perform the socket operation.
994	XSR [ENODATA]
995	No message available. No message is available on the STREAM head read queue.
996	[ENODEV]
997	No such device. An attempt was made to apply an inappropriate function to a device; for
998	example, trying to read a write-only device such as a printer.
999	[ENOENT]
1000	No such file or directory. A component of a specified pathname does not exist, or the
1001	pathname is an empty string.
1002	[ENOEXEC]
1003	Executable file format error. A request is made to execute a file that, although it has the
1004	appropriate permissions, is not in the format required by the implementation for executable
1005	files.
1006	[ENOLCK]
1007	No locks available. A system-imposed limit on the number of simultaneous file and record
1008	locks has been reached and no more are currently available.
1009	[ENOLINK]
1010	Reserved.
1011	[ENOMEM]
1012	Not enough space. The new process image requires more memory than is allowed by the
1013	hardware or system-imposed memory management constraints.
1014	[ENOMSG]
1015	No message of the desired type. The message queue does not contain a message of the
1016	required type during XSI interprocess communication.
1017	[ENOPROTOPT]
1018	Protocol not available. The protocol option specified to <i>setsockopt()</i> is not supported by the
1019	implementation.

1020	[ENOSPC]
1021	No space left on a device. During the <i>write()</i> function on a regular file or when extending a
1022	directory, there is no free space left on the device.
1023	XSR [ENOSR]
1024	No STREAM resources. Insufficient STREAMS memory resources are available to perform a
1025	STREAMS-related function. This is a temporary condition; it may be recovered from if other
1026	processes release resources.
1027	XSR [ENOSTR]
1028	Not a STREAM. A STREAM function was attempted on a file descriptor that was not
1029	associated with a STREAMS device.
1030	[ENOSYS]
1031	Function not implemented. An attempt was made to use a function that is not available in
1032	this implementation.
1033	[ENOTCONN]
1034	Socket not connected. The socket is not connected.
1035	[ENOTDIR]
1036	Not a directory. A component of the specified pathname exists, but it is not a directory,
1037	when a directory was expected.
1038	[ENOTEMPTY]
1039	Directory not empty. A directory other than an empty directory was supplied when an
1040	empty directory was expected.
1041	[ENOTSOCK]
1042	Not a socket. The file descriptor does not refer to a socket.
1043	[ENOTSUP]
1044	Not supported. The implementation does not support this feature of the Realtime Option
1045	Group.
1046	[ENOTTY]
1047	Inappropriate I/O control operation. A control function has been attempted for a file or
1048	special file for which the operation is inappropriate.
1049	[ENXIO]
1050	No such device or address. Input or output on a special file refers to a device that does not
1051	exist, or makes a request beyond the capabilities of the device. It may also occur when, for
1052	example, a tape drive is not on-line.
1053	[EOPNOTSUPP]
1054	Operation not supported on socket. The type of socket (address family or protocol) does not
1055	support the requested operation.
1056	[EOVERFLOW]
1057	Value too large to be stored in data type. An operation was attempted which would
1058	generate a value that is outside the range of values that can be represented in the relevant
1059	data type or that are allowed for a given data item.
1060	[EPERM]
1061	Operation not permitted. An attempt was made to perform an operation limited to
1062	processes with appropriate privileges or to the owner of a file or other resource.
1063	[EPIPE]
1064	Broken pipe. A write was attempted on a socket, pipe, or FIFO for which there is no process

1065	to read the data.
1066	[EPROTO]
1067	Protocol error. Some protocol error occurred. This error is device-specific, but is generally
1068	not related to a hardware failure.
1069	[EPROTONOSUPPORT]
1070	Protocol not supported. The protocol is not supported by the address family, or the protocol
1071	is not supported by the implementation.
1072	[EPROTOTYPE]
1073	Protocol wrong type for socket. The socket type is not supported by the protocol.
1074	[ERANGE]
1075	Result too large or too small. The result of the function is too large (overflow) or too small
1076	(underflow) to be represented in the available space (defined in the ISO C standard).
1077	[EROFS]
1078	Read-only file system. An attempt was made to modify a file or directory on a file system
1079	that is read-only.
1080	[ESPIPE]
1081	Invalid seek. An attempt was made to access the file offset associated with a pipe or FIFO.
1082	[ESRCH]
1083	No such process. No process can be found corresponding to that specified by the given
1084	process ID.
1085	[ESTALE]
1086	Reserved.
1087	XSR [ETIME]
1088	STREAM <i>ioctl()</i> timeout. The timer set for a STREAMS <i>ioctl()</i> call has expired. The cause of
1089	this error is device-specific and could indicate either a hardware or software failure, or a
1090	timeout value that is too short for the specific operation. The status of the <i>ioctl()</i> operation
1091	is unspecified.
1092	[ETIMEDOUT]
1093	Connection timed out. The connection to a remote machine has timed out. If the connection
1094	timed out during execution of the function that reported this error (as opposed to timing
1095	out prior to the function being called), it is unspecified whether the function has completed
1096	some or all of the documented behavior associated with a successful completion of the
1097	function.
1098	or:
1099	Operation timed out. The time limit associated with the operation was exceeded before the
1100	operation completed.
1101	[ETXTBSY]
1102	Text file busy. An attempt was made to execute a pure-procedure program that is currently
1103	open for writing, or an attempt has been made to open for writing a pure-procedure
1104	program that is being executed.
1105	[EWOULDBLOCK]
1106	Operation would block. An operation on a socket marked as non-blocking has encountered
1107	a situation such as no data available that otherwise would have caused the function to
1108	suspend execution.

1109 A conforming implementation may assign the same values for [EWOULDBLOCK] and
 1110 [EAGAIN].
 1111 [EXDEV]
 1112 Improper link. A link to a file on another file system was attempted.

1113 **2.3.1 Additional Error Numbers**

1114 Additional implementation-defined error numbers may be defined in `<errno.h>`.

1115 **2.4 Signal Concepts**

1116 **2.4.1 Signal Generation and Delivery**

1117 A signal is said to be “generated” for (or sent to) a process or thread when the event that causes
 1118 the signal first occurs. Examples of such events include detection of hardware faults, timer
 1119 expiration, signals generated via the `sigevent` structure and terminal activity, as well as
 1120 invocations of the `kill()` and `sigqueue()` functions. In some circumstances, the same event
 1121 generates signals for multiple processes.

1122 At the time of generation, a determination shall be made whether the signal has been generated
 1123 for the process or for a specific thread within the process. Signals which are generated by some
 1124 action attributable to a particular thread, such as a hardware fault, shall be generated for the
 1125 thread that caused the signal to be generated. Signals that are generated in association with a
 1126 process ID or process group ID or an asynchronous event, such as terminal activity, shall be
 1127 generated for the process.

1128 Each process has an action to be taken in response to each signal defined by the system (see
 1129 Section 2.4.3 (on page 30)). A signal is said to be “delivered” to a process when the appropriate
 1130 action for the process and signal is taken. A signal is said to be “accepted” by a process when the
 1131 signal is selected and returned by one of the `sigwait()` functions.

1132 During the time between the generation of a signal and its delivery or acceptance, the signal is
 1133 said to be “pending”. Ordinarily, this interval cannot be detected by an application. However, a
 1134 signal can be “blocked” from delivery to a thread. If the action associated with a blocked signal
 1135 is anything other than to ignore the signal, and if that signal is generated for the thread, the
 1136 signal shall remain pending until it is unblocked, it is accepted when it is selected and returned
 1137 by a call to the `sigwait()` function, or the action associated with it is set to ignore the signal.
 1138 Signals generated for the process shall be delivered to exactly one of those threads within the
 1139 process which is in a call to a `sigwait()` function selecting that signal or has not blocked delivery
 1140 of the signal. If there are no threads in a call to a `sigwait()` function selecting that signal, and if all
 1141 threads within the process block delivery of the signal, the signal shall remain pending on the
 1142 process until a thread calls a `sigwait()` function selecting that signal, a thread unblocks delivery
 1143 of the signal, or the action associated with the signal is set to ignore the signal. If the action
 1144 associated with a blocked signal is to ignore the signal and if that signal is generated for the
 1145 process, it is unspecified whether the signal is discarded immediately upon generation or
 1146 remains pending.

1147 Each thread has a “signal mask” that defines the set of signals currently blocked from delivery
 1148 to it. The signal mask for a thread shall be initialized from that of its parent or creating thread,
 1149 or from the corresponding thread in the parent process if the thread was created as the result of a
 1150 call to `fork()`. The `pthread_sigmask()`, `sigaction()`, `sigprocmask()`, and `sigsuspend()` functions control
 1151 the manipulation of the signal mask.

1152 The determination of which action is taken in response to a signal is made at the time the signal
 1153 is delivered, allowing for any changes since the time of generation. This determination is
 1154 independent of the means by which the signal was originally generated. If a subsequent
 1155 occurrence of a pending signal is generated, it is implementation-defined as to whether the
 1156 RTS signal is delivered or accepted more than once in circumstances other than those in which
 1157 queuing is required under the Realtime Signals Extension option. The order in which multiple,
 1158 simultaneously pending signals outside the range SIGRTMIN to SIGRTMAX are delivered to or
 1159 accepted by a process is unspecified.

1160 When any stop signal (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU) is generated for a process, any
 1161 pending SIGCONT signals for that process shall be discarded. Conversely, when SIGCONT is
 1162 generated for a process, all pending stop signals for that process shall be discarded. When
 1163 SIGCONT is generated for a process that is stopped, the process shall be continued, even if the
 1164 SIGCONT signal is blocked or ignored. If SIGCONT is blocked and not ignored, it shall remain
 1165 pending until it is either unblocked or a stop signal is generated for the process.

1166 An implementation shall document any condition not specified by this volume of
 1167 IEEE Std 1003.1-2001 under which the implementation generates signals.

1168 2.4.2 Realtime Signal Generation and Delivery

1169 RTS This section describes extensions to support realtime signal generation and delivery. This
 1170 functionality is dependent on support of the Realtime Signals Extension option (and the rest of
 1171 this section is not further shaded for this option).

1172 Some signal-generating functions, such as high-resolution timer expiration, asynchronous I/O
 1173 completion, interprocess message arrival, and the *sigqueue()* function, support the specification
 1174 of an application-defined value, either explicitly as a parameter to the function or in a **sigevent**
 1175 structure parameter. The **sigevent** structure is defined in <signal.h> and contains at least the
 1176 following members:

1177

1178

1179

1180

1181

1182

1183

Member Type	Member Name	Description
int	<i>sigev_notify</i>	Notification type.
int	<i>sigev_signo</i>	Signal number.
union signal	<i>sigev_value</i>	Signal value.
void*(unsigned signal)	<i>sigev_notify_function</i>	Notification function.
(pthread_attr_t*)	<i>sigev_notify_attributes</i>	Notification attributes.

1184 The *sigev_notify* member specifies the notification mechanism to use when an asynchronous
 1185 event occurs. This volume of IEEE Std 1003.1-2001 defines the following values for the
 1186 *sigev_notify* member:

1187 SIGEV_NONE No asynchronous notification shall be delivered when the event of
 1188 interest occurs.

1189 SIGEV_SIGNAL The signal specified in *sigev_signo* shall be generated for the process when
 1190 the event of interest occurs. If the implementation supports the Realtime
 1191 Signals Extension option and if the SA_SIGINFO flag is set for that signal
 1192 number, then the signal shall be queued to the process and the value
 1193 specified in *sigev_value* shall be the *si_value* component of the generated
 1194 signal. If SA_SIGINFO is not set for that signal number, it is unspecified
 1195 whether the signal is queued and what value, if any, is sent.

1196 SIGEV_THREAD A notification function shall be called to perform notification.

1197 An implementation may define additional notification mechanisms.

1198 The *sigev_signo* member specifies the signal to be generated. The *sigev_value* member is the
1199 application-defined value to be passed to the signal-catching function at the time of the signal
1200 delivery or to be returned at signal acceptance as the *si_value* member of the **siginfo_t** structure.

1201 The **signal** union is defined in `<signal.h>` and contains at least the following members:

1202

1203

1204

1205

Member Type	Member Name	Description
int	<i>sival_int</i>	Integer signal value.
void*	<i>sival_ptr</i>	Pointer signal value.

1206 The *sival_int* member shall be used when the application-defined value is of type **int**; the
1207 *sival_ptr* member shall be used when the application-defined value is a pointer.

1208 When a signal is generated by the *sigqueue()* function or any signal-generating function that
1209 supports the specification of an application-defined value, the signal shall be marked pending
1210 and, if the SA_SIGINFO flag is set for that signal, the signal shall be queued to the process along
1211 with the application-specified signal value. Multiple occurrences of signals so generated are
1212 queued in FIFO order. It is unspecified whether signals so generated are queued when the
1213 SA_SIGINFO flag is not set for that signal.

1214 Signals generated by the *kill()* function or other events that cause signals to occur, such as
1215 detection of hardware faults, *alarm()* timer expiration, or terminal activity, and for which the
1216 implementation does not support queuing, shall have no effect on signals already queued for the
1217 same signal number.

1218 When multiple unblocked signals, all in the range SIGRTMIN to SIGRTMAX, are pending, the
1219 behavior shall be as if the implementation delivers the pending unblocked signal with the lowest
1220 signal number within that range. No other ordering of signal delivery is specified.

1221 If, when a pending signal is delivered, there are additional signals queued to that signal number,
1222 the signal shall remain pending. Otherwise, the pending indication shall be reset.

1223 Multi-threaded programs can use an alternate event notification mechanism. When a
1224 notification is processed, and the *sigev_notify* member of the **sigevent** structure has the value
1225 SIGEV_THREAD, the function *sigev_notify_function* is called with parameter *sigev_value*.

1226 The function shall be executed in an environment as if it were the *start_routine* for a newly
1227 created thread with thread attributes specified by *sigev_notify_attributes*. If *sigev_notify_attributes*
1228 is NULL, the behavior shall be as if the thread were created with the *detachstate* attribute set to
1229 PTHREAD_CREATE_DETACHED. Supplying an attributes structure with a *detachstate* attribute
1230 of PTHREAD_CREATE_JOINABLE results in undefined behavior. The signal mask of this
1231 thread is implementation-defined.

1232 2.4.3 Signal Actions

1233 There are three types of action that can be associated with a signal: SIG_DFL, SIG_IGN, or a
1234 pointer to a function. Initially, all signals shall be set to SIG_DFL or SIG_IGN prior to entry of
1235 the *main()* routine (see the *exec* functions). The actions prescribed by these values are as follows:

1236 SIG_DFL Signal-specific default action.

1237 The default actions for the signals defined in this volume of IEEE Std 1003.1-2001
1238 RTS are specified under `<signal.h>`. If the Realtime Signals Extension option is
1239 supported, the default actions for the realtime signals in the range SIGRTMIN to
1240 SIGRTMAX shall be to terminate the process abnormally.

1241		If the default action is to stop the process, the execution of that process is temporarily suspended. When a process stops, a SIGCHLD signal shall be
1242		generated for its parent process, unless the parent process has set the
1243		SA_NOCLDSTOP flag. While a process is stopped, any additional signals that are
1244		sent to the process shall not be delivered until the process is continued, except
1245		SIGKILL which always terminates the receiving process. A process that is a
1246		member of an orphaned process group shall not be allowed to stop in response to
1247		the SIGTSTP, SIGTTIN, or SIGTTOU signals. In cases where delivery of one of
1248		these signals would stop such a process, the signal shall be discarded.
1249		
1250		Setting a signal action to SIG_DFL for a signal that is pending, and whose default
1251		action is to ignore the signal (for example, SIGCHLD), shall cause the pending
1252	RTS	signal to be discarded, whether or not it is blocked. If the Realtime Signals
1253		Extension option is supported, any queued values pending shall be discarded and
1254		the resources used to queue them shall be released and returned to the system for
1255		other use.
1256		The default action for SIGCONT is to resume execution at the point where the
1257		process was stopped, after first handling any pending unblocked signals.
1258	XSI	When a stopped process is continued, a SIGCHLD signal may be generated for its
1259		parent process, unless the parent process has set the SA_NOCLDSTOP flag.
1260	SIG_IGN	Ignore signal.
1261		Delivery of the signal shall have no effect on the process. The behavior of a process
1262	RTS	is undefined after it ignores a SIGFPE, SIGILL, SIGSEGV, or SIGBUS signal that
1263	RTS	was not generated by <i>kill()</i> , <i>sigqueue()</i> , or <i>raise()</i> .
1264		The system shall not allow the action for the signals SIGKILL or SIGSTOP to be set
1265		to SIG_IGN.
1266		Setting a signal action to SIG_IGN for a signal that is pending shall cause the
1267		pending signal to be discarded, whether or not it is blocked.
1268		If a process sets the action for the SIGCHLD signal to SIG_IGN, the behavior is
1269	XSI	unspecified, except as specified below.
1270		If the action for the SIGCHLD signal is set to SIG_IGN, child processes of the
1271		calling processes shall not be transformed into zombie processes when they
1272		terminate. If the calling process subsequently waits for its children, and the process
1273		has no unwaited-for children that were transformed into zombie processes, it shall
1274		block until all of its children terminate, and <i>wait()</i> , <i>waitid()</i> , and <i>waitpid()</i> shall fail
1275		and set <i>errno</i> to [ECHILD].
1276	RTS	If the Realtime Signals Extension option is supported, any queued values pending
1277		shall be discarded and the resources used to queue them shall be released and
1278		made available to queue other signals.
1279		<i>pointer to a function</i>
1280		Catch signal.
1281		On delivery of the signal, the receiving process is to execute the signal-catching
1282		function at the specified address. After returning from the signal-catching function,
1283		the receiving process shall resume execution at the point at which it was
1284		interrupted.
1285		If the SA_SIGINFO flag for the signal is cleared, the signal-catching function shall
1286		be entered as a C-language function call as follows:

1287 `void func(int signo);`

1288 XSI|RTS If the SA_SIGINFO flag for the signal is set, the signal-catching function shall be
1289 entered as a C-language function call as follows:

1290 `void func(int signo, siginfo_t *info, void *context);`

1291 where *func* is the specified signal-catching function, *signo* is the signal number of
1292 the signal being delivered, and *info* is a pointer to a **siginfo_t** structure defined in
1293 **<signal.h>** containing at least the following members:

Member Type	Member Name	Description
int	<i>si_signo</i>	Signal number.
int	<i>si_code</i>	Cause of the signal.
union signal	<i>si_value</i>	Signal value.

1299 The *si_signo* member shall contain the signal number. This shall be the same as the
1300 *signo* parameter. The *si_code* member shall contain a code identifying the cause of
1301 the signal. The following values are defined for *si_code*:

1302 SI_USER The signal was sent by the *kill()* function. The implementation
1303 may set *si_code* to SI_USER if the signal was sent by the *raise()* or
1304 *abort()* functions or any similar functions provided as
1305 implementation extensions.

1306 RTS SI_QUEUE The signal was sent by the *sigqueue()* function.

1307 RTS SI_TIMER The signal was generated by the expiration of a timer set by
1308 *timer_settime()*.

1309 RTS SI_ASYNCIO The signal was generated by the completion of an asynchronous
1310 I/O request.

1311 RTS SI_MESGQ The signal was generated by the arrival of a message on an
1312 empty message queue.

1313 If the signal was not generated by one of the functions or events listed above, the
1314 *si_code* shall be set to an implementation-defined value that is not equal to any of
1315 the values defined above.

1316 RTS If the Realtime Signals Extension is supported, and *si_code* is one of SI_QUEUE,
1317 SI_TIMER, SI_ASYNCIO, or SI_MESGQ, then *si_value* shall contain the
1318 application-specified signal value. Otherwise, the contents of *si_value* are
1319 undefined.

1320 The behavior of a process is undefined after it returns normally from a signal-
1321 XSI catching function for a SIGBUS, SIGFPE, SIGILL, or SIGSEGV signal that was not
1322 RTS generated by *kill()*, *sigqueue()*, or *raise()*.

1323 The system shall not allow a process to catch the signals SIGKILL and SIGSTOP.

1324 If a process establishes a signal-catching function for the SIGCHLD signal while it
1325 has a terminated child process for which it has not waited, it is unspecified
1326 whether a SIGCHLD signal is generated to indicate that child process.

1327 When signal-catching functions are invoked asynchronously with process
1328 execution, the behavior of some of the functions defined by this volume of
1329 IEEE Std 1003.1-2001 is unspecified if they are called from a signal-catching
1330 function.

1331 The following table defines a set of functions that shall be either reentrant or non-
 1332 interruptible by signals and shall be async-signal-safe. Therefore applications may
 1333 invoke them, without restriction, from signal-catching functions:

1334	<i>_Exit()</i>	<i>fpathconf()</i>	<i>read()</i>	<i>sigset()</i>
1335	<i>_exit()</i>	<i>fstat()</i>	<i>readlink()</i>	<i>sigsuspend()</i>
1336	<i>abort()</i>	<i>fsync()</i>	<i>recv()</i>	<i>socket()</i>
1337	<i>accept()</i>	<i>ftruncate()</i>	<i>recvfrom()</i>	<i>socketpair()</i>
1338	<i>access()</i>	<i>getegid()</i>	<i>recvmsg()</i>	<i>stat()</i>
1339	<i>aio_error()</i>	<i>geteuid()</i>	<i>rename()</i>	<i>symlink()</i>
1340	<i>aio_return()</i>	<i>getgid()</i>	<i>rmdir()</i>	<i>sysconf()</i>
1341	<i>aio_suspend()</i>	<i>getgroups()</i>	<i>select()</i>	<i>tcdrain()</i>
1342	<i>alarm()</i>	<i>getpeername()</i>	<i>sem_post()</i>	<i>tcflow()</i>
1343	<i>bind()</i>	<i>getpgrp()</i>	<i>send()</i>	<i>tcf flush()</i>
1344	<i>cfgetispeed()</i>	<i>getpid()</i>	<i>sendmsg()</i>	<i>tcgetattr()</i>
1345	<i>cfgetospeed()</i>	<i>getppid()</i>	<i>sendto()</i>	<i>tcgetpgrp()</i>
1346	<i>cfsetispeed()</i>	<i>getsockname()</i>	<i>setgid()</i>	<i>tcsendbreak()</i>
1347	<i>cfsetospeed()</i>	<i>getsockopt()</i>	<i>setpgid()</i>	<i>tcsetattr()</i>
1348	<i>chdir()</i>	<i>getuid()</i>	<i>setsid()</i>	<i>tcsetpgrp()</i>
1349	<i>chmod()</i>	<i>kill()</i>	<i>setsockopt()</i>	<i>time()</i>
1350	<i>chown()</i>	<i>link()</i>	<i>setuid()</i>	<i>timer_getoverrun()</i>
1351	<i>clock_gettime()</i>	<i>listen()</i>	<i>shutdown()</i>	<i>timer_gettime()</i>
1352	<i>close()</i>	<i>lseek()</i>	<i>sigaction()</i>	<i>timer_settime()</i>
1353	<i>connect()</i>	<i>lstat()</i>	<i>sigaddset()</i>	<i>times()</i>
1354	<i>creat()</i>	<i>mkdir()</i>	<i>sigdelset()</i>	<i>umask()</i>
1355	<i>dup()</i>	<i>mkfifo()</i>	<i>sigemptyset()</i>	<i>uname()</i>
1356	<i>dup2()</i>	<i>open()</i>	<i>sigfillset()</i>	<i>unlink()</i>
1357	<i>execle()</i>	<i>pathconf()</i>	<i>sigismember()</i>	<i>utime()</i>
1358	<i>execve()</i>	<i>pause()</i>	<i>sleep()</i>	<i>wait()</i>
1359	<i>fchmod()</i>	<i>pipe()</i>	<i>signal()</i>	<i>waitpid()</i>
1360	<i>fchown()</i>	<i>poll()</i>	<i>sigpause()</i>	<i>write()</i>
1361	<i>fcntl()</i>	<i>posix_trace_event()</i>	<i>sigpending()</i>	
1362	<i>fdatasync()</i>	<i>pselect()</i>	<i>sigprocmask()</i>	
1363	<i>fork()</i>	<i>raise()</i>	<i>sigqueue()</i>	

1364 All functions not in the above table are considered to be unsafe with respect to
 1365 signals. In the presence of signals, all functions defined by this volume of
 1366 IEEE Std 1003.1-2001 shall behave as defined when called from or interrupted by a
 1367 signal-catching function, with a single exception: when a signal interrupts an
 1368 unsafe function and the signal-catching function calls an unsafe function, the
 1369 behavior is undefined.

1370 When a signal is delivered to a thread, if the action of that signal specifies termination, stop, or
 1371 continue, the entire process shall be terminated, stopped, or continued, respectively.

1372 2.4.4 Signal Effects on Other Functions

1373 Signals affect the behavior of certain functions defined by this volume of IEEE Std 1003.1-2001 if
1374 delivered to a process while it is executing such a function. If the action of the signal is to
1375 terminate the process, the process shall be terminated and the function shall not return. If the
1376 action of the signal is to stop the process, the process shall stop until continued or terminated.
1377 Generation of a SIGCONT signal for the process shall cause the process to be continued, and the
1378 original function shall continue at the point the process was stopped. If the action of the signal is
1379 to invoke a signal-catching function, the signal-catching function shall be invoked; in this case
1380 the original function is said to be “interrupted” by the signal. If the signal-catching function
1381 executes a **return** statement, the behavior of the interrupted function shall be as described
1382 individually for that function, except as noted for unsafe functions. Signals that are ignored shall
1383 not affect the behavior of any function; signals that are blocked shall not affect the behavior of
1384 any function until they are unblocked and then delivered, except as specified for the *sigpending()*
1385 and *sigwait()* functions.

1386 2.5 Standard I/O Streams

1387 A stream is associated with an external file (which may be a physical device) by “opening” a file,
1388 which may involve “creating” a new file. Creating an existing file causes its former contents to
1389 be discarded if necessary. If a file can support positioning requests (such as a disk file, as
1390 opposed to a terminal), then a “file position indicator” associated with the stream is positioned
1391 at the start (byte number 0) of the file, unless the file is opened with append mode, in which case
1392 it is implementation-defined whether the file position indicator is initially positioned at the
1393 beginning or end of the file. The file position indicator is maintained by subsequent reads,
1394 writes, and positioning requests, to facilitate an orderly progression through the file. All input
1395 takes place as if bytes were read by successive calls to *fgetc()*; all output takes place as if bytes
1396 were written by successive calls to *fputc()*.

1397 When a stream is “unbuffered”, bytes are intended to appear from the source or at the
1398 destination as soon as possible; otherwise, bytes may be accumulated and transmitted as a block.
1399 When a stream is “fully buffered”, bytes are intended to be transmitted as a block when a buffer
1400 is filled. When a stream is “line buffered”, bytes are intended to be transmitted as a block when a
1401 newline byte is encountered. Furthermore, bytes are intended to be transmitted as a block when
1402 a buffer is filled, when input is requested on an unbuffered stream, or when input is requested
1403 on a line-buffered stream that requires the transmission of bytes. Support for these
1404 characteristics is implementation-defined, and may be affected via *setbuf()* and *setvbuf()*.

1405 A file may be disassociated from a controlling stream by “closing” the file. Output streams are
1406 flushed (any unwritten buffer contents are transmitted) before the stream is disassociated from
1407 the file. The value of a pointer to a **FILE** object is unspecified after the associated file is closed
1408 (including the standard streams).

1409 A file may be subsequently reopened, by the same or another program execution, and its
1410 contents reclaimed or modified (if it can be repositioned at its start). If the *main()* function
1411 returns to its original caller, or if the *exit()* function is called, all open files are closed (hence all
1412 output streams are flushed) before program termination. Other paths to program termination,
1413 such as calling *abort()*, need not close all files properly.

1414 The address of the **FILE** object used to control a stream may be significant; a copy of a **FILE**
1415 object need not necessarily serve in place of the original.

1416 At program start-up, three streams are predefined and need not be opened explicitly: *standard*
1417 *input* (for reading conventional input), *standard output* (for writing conventional output), and

1418 *standard error* (for writing diagnostic output). When opened, the standard error stream is not
 1419 fully buffered; the standard input and standard output streams are fully buffered if and only if
 1420 the stream can be determined not to refer to an interactive device.

1421 **2.5.1 Interaction of File Descriptors and Standard I/O Streams**

1422 cx This section describes the interaction of file descriptors and standard I/O streams. This
 1423 functionality is an extension to the ISO C standard (and the rest of this section is not further CX
 1424 shaded).

1425 An open file description may be accessed through a file descriptor, which is created using
 1426 functions such as *open()* or *pipe()*, or through a stream, which is created using functions such as
 1427 *fopen()* or *popen()*. Either a file descriptor or a stream is called a “handle” on the open file
 1428 description to which it refers; an open file description may have several handles.

1429 Handles can be created or destroyed by explicit user action, without affecting the underlying
 1430 open file description. Some of the ways to create them include *fcntl()*, *dup()*, *fdopen()*, *fileno()*,
 1431 and *fork()*. They can be destroyed by at least *fclose()*, *close()*, and the *exec* functions.

1432 A file descriptor that is never used in an operation that could affect the file offset (for example,
 1433 *read()*, *write()*, or *lseek()*) is not considered a handle for this discussion, but could give rise to one
 1434 (for example, as a consequence of *fdopen()*, *dup()*, or *fork()*). This exception does not include the
 1435 file descriptor underlying a stream, whether created with *fopen()* or *fdopen()*, so long as it is not
 1436 used directly by the application to affect the file offset. The *read()* and *write()* functions
 1437 implicitly affect the file offset; *lseek()* explicitly affects it.

1438 The result of function calls involving any one handle (the “active handle”) is defined elsewhere
 1439 in this volume of IEEE Std 1003.1-2001, but if two or more handles are used, and any one of them
 1440 is a stream, the application shall ensure that their actions are coordinated as described below. If
 1441 this is not done, the result is undefined.

1442 A handle which is a stream is considered to be closed when either an *fclose()* or *freopen()* is
 1443 executed on it (the result of *freopen()* is a new stream, which cannot be a handle on the same
 1444 open file description as its previous value), or when the process owning that stream terminates
 1445 with *exit()*, *abort()*, or due to a signal. A file descriptor is closed by *close()*, *_exit()*, or the *exec*
 1446 functions when FD_CLOEXEC is set on that file descriptor.

1447 For a handle to become the active handle, the application shall ensure that the actions below are
 1448 performed between the last use of the handle (the current active handle) and the first use of the
 1449 second handle (the future active handle). The second handle then becomes the active handle. All
 1450 activity by the application affecting the file offset on the first handle shall be suspended until it
 1451 again becomes the active file handle. (If a stream function has as an underlying function one that
 1452 affects the file offset, the stream function shall be considered to affect the file offset.)

1453 The handles need not be in the same process for these rules to apply.

1454 Note that after a *fork()*, two handles exist where one existed before. The application shall ensure
 1455 that, if both handles can ever be accessed, they are both in a state where the other could become
 1456 the active handle first. The application shall prepare for a *fork()* exactly as if it were a change of
 1457 active handle. (If the only action performed by one of the processes is one of the *exec* functions or
 1458 *_exit()* (not *exit()*), the handle is never accessed in that process.)

1459 For the first handle, the first applicable condition below applies. After the actions required
 1460 below are taken, if the handle is still open, the application can close it.

- 1461 • If it is a file descriptor, no action is required.

- 1462 • If the only further action to be performed on any handle to this open file descriptor is to close
1463 it, no action need be taken.
- 1464 • If it is a stream which is unbuffered, no action need be taken.
- 1465 • If it is a stream which is line buffered, and the last byte written to the stream was a
1466 <newline> (that is, as if a:
- 1467 putc('\n')
- 1468 was the most recent operation on that stream), no action need be taken.
- 1469 • If it is a stream which is open for writing or appending (but not also open for reading), the
1470 application shall either perform an *flush()*, or the stream shall be closed.
- 1471 • If the stream is open for reading and it is at the end of the file (*feof()* is true), no action need
1472 be taken.
- 1473 • If the stream is open with a mode that allows reading and the underlying open file
1474 description refers to a device that is capable of seeking, the application shall either perform
1475 an *flush()*, or the stream shall be closed.

1476 Otherwise, the result is undefined.

1477 For the second handle:

- 1478 • If any previous active handle has been used by a function that explicitly changed the file
1479 offset, except as required above for the first handle, the application shall perform an *lseek()* or
1480 *fseek()* (as appropriate to the type of handle) to an appropriate location.

1481 If the active handle ceases to be accessible before the requirements on the first handle, above,
1482 have been met, the state of the open file description becomes undefined. This might occur during
1483 functions such as a *fork()* or *_exit()*.

1484 The *exec* functions make inaccessible all streams that are open at the time they are called,
1485 independent of which streams or file descriptors may be available to the new process image.

1486 When these rules are followed, regardless of the sequence of handles used, implementations
1487 shall ensure that an application, even one consisting of several processes, shall yield correct
1488 results: no data shall be lost or duplicated when writing, and all data shall be written in order,
1489 except as requested by seeks. It is implementation-defined whether, and under what conditions,
1490 all input is seen exactly once.

1491 If the rules above are not followed, the result is unspecified.

1492 Each function that operates on a stream is said to have zero or more “underlying functions”.
1493 This means that the stream function shares certain traits with the underlying functions, but does
1494 not require that there be any relation between the implementations of the stream function and its
1495 underlying functions.

1496 2.5.2 Stream Orientation and Encoding Rules

1497 For conformance to the ISO/IEC 9899:1999 standard, the definition of a stream includes an
1498 “orientation”. After a stream is associated with an external file, but before any operations are
1499 performed on it, the stream is without orientation. Once a wide-character input/output function
1500 has been applied to a stream without orientation, the stream shall become “wide-oriented”.
1501 Similarly, once a byte input/output function has been applied to a stream without orientation,
1502 the stream shall become “byte-oriented”. Only a call to the *freopen()* function or the *fwide()*
1503 function can otherwise alter the orientation of a stream.

1504 A successful call to *freopen()* shall remove any orientation. The three predefined streams *standard*
 1505 *input*, *standard output*, and *standard error* shall be unoriented at program start-up.

1506 Byte input/output functions cannot be applied to a wide-oriented stream, and wide-character
 1507 input/output functions cannot be applied to a byte-oriented stream. The remaining stream
 1508 operations shall not affect and shall not be affected by a stream's orientation, except for the
 1509 following additional restriction:

- 1510 • For wide-oriented streams, after a successful call to a file-positioning function that leaves the
 1511 file position indicator prior to the end-of-file, a wide-character output function can overwrite
 1512 a partial character; any file contents beyond the byte(s) written are henceforth undefined.

1513 Each wide-oriented stream has an associated **mbstate_t** object that stores the current parse state
 1514 of the stream. A successful call to *fgetpos()* shall store a representation of the value of this
 1515 **mbstate_t** object as part of the value of the **fpos_t** object. A later successful call to *fsetpos()* using
 1516 the same stored **fpos_t** value shall restore the value of the associated **mbstate_t** object as well as
 1517 the position within the controlled stream.

1518 Implementations that support multiple encoding rules associate an encoding rule with the
 1519 stream. The encoding rule shall be determined by the setting of the *LC_CTYPE* category in the
 1520 current locale at the time when the stream becomes wide-oriented. As with the stream's
 1521 orientation, the encoding rule associated with a stream cannot be changed once it has been set,
 1522 except by a successful call to *freopen()* which clears the encoding rule and resets the orientation
 1523 to unoriented.

1524 Although wide-oriented streams are conceptually sequences of wide characters, the external file
 1525 associated with a wide-oriented stream is a sequence of (possibly multi-byte) characters
 1526 generalized as follows:

- 1527 • Multi-byte encodings within files may contain embedded null bytes (unlike multi-byte
 1528 encodings valid for use internal to the program).
- 1529 • A file need not begin nor end in the initial shift state.

1530 Moreover, the encodings used for characters may differ among files. Both the nature and choice
 1531 of such encodings are implementation-defined.

1532 The wide-character input functions read characters from the stream and convert them to wide
 1533 characters as if they were read by successive calls to the *fgetwc()* function. Each conversion shall
 1534 occur as if by a call to the *mbrtowc()* function, with the conversion state described by the stream's
 1535 own **mbstate_t** object, except the encoding rule associated with the stream is used instead of the
 1536 encoding rule implied by the *LC_CTYPE* category of the current locale.

1537 The wide-character output functions convert wide characters to (possibly multi-byte) characters
 1538 and write them to the stream as if they were written by successive calls to the *fputwc()* function.
 1539 Each conversion shall occur as if by a call to the *wcrtomb()* function, with the conversion state
 1540 described by the stream's own **mbstate_t** object, except the encoding rule associated with the
 1541 stream is used instead of the encoding rule implied by the *LC_CTYPE* category of the current
 1542 locale.

1543 An “encoding error” shall occur if the character sequence presented to the underlying *mbrtowc()*
 1544 function does not form a valid (generalized) character, or if the code value passed to the
 1545 underlying *wcrtomb()* function does not correspond to a valid (generalized) character. The
 1546 wide-character input/output functions and the byte input/output functions store the value of
 1547 the macro [EILSEQ] in *errno* if and only if an encoding error occurs.

1548 **2.6 STREAMS**

1549 XSR STREAMS functionality is provided on implementations supporting the XSI STREAMS Option
 1550 Group. This functionality is dependent on support of the XSI STREAMS option (and the rest of
 1551 this section is not further shaded for this option).

1552 STREAMS provides a uniform mechanism for implementing networking services and other
 1553 character-based I/O. The STREAMS function provides direct access to protocol modules.
 1554 STREAMS modules are unspecified objects. Access to STREAMS modules is provided by
 1555 interfaces in IEEE Std 1003.1-2001. Creation of STREAMS modules is outside the scope of
 1556 IEEE Std 1003.1-2001.

1557 A STREAM is typically a full-duplex connection between a process and an open device or
 1558 pseudo-device. However, since pipes may be STREAMS-based, a STREAM can be a full-duplex
 1559 connection between two processes. The STREAM itself exists entirely within the implementation
 1560 and provides a general character I/O function for processes. It optionally includes one or more
 1561 intermediate processing modules that are interposed between the process end of the STREAM
 1562 (STREAM head) and a device driver at the end of the STREAM (STREAM end).

1563 STREAMS I/O is based on messages. There are three types of message:

- 1564 • *Data messages* containing actual data for input or output
- 1565 • *Control data* containing instructions for the STREAMS modules and underlying
 1566 implementation
- 1567 • Other messages, which include file descriptors

1568 The interface between the STREAM and the rest of the implementation is provided by a set of
 1569 functions at the STREAM head. When a process calls *write()*, *writv()*, *putmsg()*, *putpmsg()*, or
 1570 *ioctl()*, messages are sent down the STREAM, and *read()*, *readv()*, *getmsg()*, or *getpmsg()* accepts
 1571 data from the STREAM and passes it to a process. Data intended for the device at the
 1572 downstream end of the STREAM is packaged into messages and sent downstream, while data
 1573 and signals from the device are composed into messages by the device driver and sent upstream
 1574 to the STREAM head.

1575 When a STREAMS-based device is opened, a STREAM shall be created that contains the
 1576 STREAM head and the STREAM end (driver). If pipes are STREAMS-based in an
 1577 implementation, when a pipe is created, two STREAMS shall be created, each containing a
 1578 STREAM head. Other modules are added to the STREAM using *ioctl()*. New modules are
 1579 “pushed” onto the STREAM one at a time in last-in, first-out (LIFO) style, as though the
 1580 STREAM was a push-down stack.

1581 **Priority**

1582 Message types are classified according to their queuing priority and may be *normal* (non-
 1583 priority), *priority*, or *high-priority* messages. A message belongs to a particular priority band that
 1584 determines its ordering when placed on a queue. Normal messages have a priority band of 0 and
 1585 shall always be placed at the end of the queue following all other messages in the queue. High-
 1586 priority messages are always placed at the head of a queue, but shall be discarded if there is
 1587 already a high-priority message in the queue. Their priority band shall be ignored; they are
 1588 high-priority by virtue of their type. Priority messages have a priority band greater than 0.
 1589 Priority messages are always placed after any messages of the same or higher priority. High-
 1590 priority and priority messages are used to send control and data information outside the normal
 1591 flow of control. By convention, high-priority messages shall not be affected by flow control.
 1592 Normal and priority messages have separate flow controls.

1593 **Message Parts**

1594 A process may access STREAMS messages that contain a data part, control part, or both. The
 1595 data part is that information which is transmitted over the communication medium and the
 1596 control information is used by the local STREAMS modules. The other types of messages are
 1597 used between modules and are not accessible to processes. Messages containing only a data part
 1598 are accessible via *putmsg()*, *putpmsg()*, *getmsg()*, *getpmsg()*, *read()*, *readv()*, *write()*, or *writev()*.
 1599 Messages containing a control part with or without a data part are accessible via calls to
 1600 *putmsg()*, *putpmsg()*, *getmsg()*, or *getpmsg()*.

1601 **2.6.1 Accessing STREAMS**

1602 A process accesses STREAMS-based files using the standard functions *close()*, *ioctl()*, *getmsg()*,
 1603 *getpmsg()*, *open()*, *pipe()*, *poll()*, *putmsg()*, *putpmsg()*, *read()*, or *write()*. Refer to the applicable
 1604 function definitions for general properties and errors.

1605 Calls to *ioctl()* shall perform control functions on the STREAM associated with the file descriptor
 1606 *fildev*. The control functions may be performed by the STREAM head, a STREAMS module, or
 1607 the STREAMS driver for the STREAM.

1608 STREAMS modules and drivers can detect errors, sending an error message to the STREAM
 1609 head, thus causing subsequent functions to fail and set *errno* to the value specified in the
 1610 message. In addition, STREAMS modules and drivers can elect to fail a particular *ioctl()* request
 1611 alone by sending a negative acknowledgement message to the STREAM head. This shall cause
 1612 just the pending *ioctl()* request to fail and set *errno* to the value specified in the message.

1613 **2.7 XSI Interprocess Communication**

1614 XSI This section describes extensions to support interprocess communication. This functionality is
 1615 dependent on support of the XSI extension (and the rest of this section is not further shaded for
 1616 this option).

1617 The following message passing, semaphore, and shared memory services form an XSI
 1618 interprocess communication facility. Certain aspects of their operation are common, and are
 1619 defined as follows.

1620

1621

1622

1623

1624

1625

IPC Functions		
<i>msgctl()</i>	<i>semctl()</i>	<i>shmctl()</i>
<i>msgget()</i>	<i>semget()</i>	<i>shmdt()</i>
<i>msgrcv()</i>	<i>semop()</i>	<i>shmget()</i>
<i>msgsnd()</i>	<i>shmat()</i>	

1626 Another interprocess communication facility is provided by functions in the Realtime Option
 1627 Group; see Section 2.8 (on page 41).

1628 **2.7.1 IPC General Description**

1629 Each individual shared memory segment, message queue, and semaphore set shall be identified
 1630 by a unique positive integer, called, respectively, a shared memory identifier, *shmid*, a
 1631 semaphore identifier, *semid*, and a message queue identifier, *msgid*. The identifiers shall be
 1632 returned by calls to *shmget()*, *semget()*, and *msgget()*, respectively.

1633 Associated with each identifier is a data structure which contains data related to the operations
 1634 which may be or may have been performed; see the Base Definitions volume of
 1635 IEEE Std 1003.1-2001, <*sys/shm.h*>, <*sys/sem.h*>, and <*sys/msg.h*> for their descriptions.

1636 Each of the data structures contains both ownership information and an **ipc_perm** structure (see
 1637 the Base Definitions volume of IEEE Std 1003.1-2001, <*sys/ipc.h*>) which are used in conjunction
 1638 to determine whether or not read/write (read/alter for semaphores) permissions should be
 1639 granted to processes using the IPC facilities. The *mode* member of the **ipc_perm** structure acts as
 1640 a bit field which determines the permissions.

1641 The values of the bits are given below in octal notation.

1642
 1643

Bit	Meaning
0400	Read by user.
0200	Write by user.
0040	Read by group.
0020	Write by group.
0004	Read by others.
0002	Write by others.

1644
 1645
 1646
 1647
 1648
 1649

1650 The name of the **ipc_perm** structure is *shm_perm*, *sem_perm*, or *msg_perm*, depending on which
 1651 service is being used. In each case, read and write/alter permissions shall be granted to a process
 1652 if one or more of the following are true ("xxx" is replaced by *shm*, *sem*, or *msg*, as appropriate):

- 1653 • The process has appropriate privileges.
- 1654 • The effective user ID of the process matches *xxx_perm.cuid* or *xxx_perm.uid* in the data
 1655 structure associated with the IPC identifier, and the appropriate bit of the *user* field in
 1656 *xxx_perm.mode* is set.
- 1657 • The effective user ID of the process does not match *xxx_perm.cuid* or *xxx_perm.uid* but the
 1658 effective group ID of the process matches *xxx_perm.cgid* or *xxx_perm.gid* in the data structure
 1659 associated with the IPC identifier, and the appropriate bit of the *group* field in *xxx_perm.mode*
 1660 is set.
- 1661 • The effective user ID of the process does not match *xxx_perm.cuid* or *xxx_perm.uid* and the
 1662 effective group ID of the process does not match *xxx_perm.cgid* or *xxx_perm.gid* in the data
 1663 structure associated with the IPC identifier, but the appropriate bit of the *other* field in
 1664 *xxx_perm.mode* is set.

1665 Otherwise, the permission shall be denied.

1666 2.8 Realtime

1667 This section defines functions to support the source portability of applications with realtime
1668 requirements. The presence of many of these functions is dependent on support for
1669 implementation options described in the text.

1670 The specific functional areas included in this section and their scope include the following. Full
1671 definitions of these terms can be found in the Base Definitions volume of IEEE Std 1003.1-2001,
1672 Chapter 3, Definitions.

- 1673 • Semaphores
- 1674 • Process Memory Locking
- 1675 • Memory Mapped Files and Shared Memory Objects
- 1676 • Priority Scheduling
- 1677 • Realtime Signal Extension
- 1678 • Timers
- 1679 • Interprocess Communication
- 1680 • Synchronized Input and Output
- 1681 • Asynchronous Input and Output

1682 All the realtime functions defined in this volume of IEEE Std 1003.1-2001 are portable, although
1683 some of the numeric parameters used by an implementation may have hardware dependencies.

1684 2.8.1 Realtime Signals

1685 RTS Realtime signal generation and delivery is dependent on support for the Realtime Signals
1686 Extension option.

1687 See Section 2.4.2 (on page 29).

1688 2.8.2 Asynchronous I/O

1689 AIO The functionality described in this section is dependent on support of the Asynchronous Input
1690 and Output option (and the rest of this section is not further shaded for this option).

1691 An asynchronous I/O control block structure **aiocb** is used in many asynchronous I/O
1692 functions. It is defined in the Base Definitions volume of IEEE Std 1003.1-2001, `<aio.h>` and has
1693 at least the following members:

1694	Member Type	Member Name	Description
1695	int	<i>aio_fildes</i>	File descriptor.
1696	off_t	<i>aio_offset</i>	File offset.
1697	volatile void*	<i>aio_buf</i>	Location of buffer.
1698	size_t	<i>aio_nbytes</i>	Length of transfer.
1699	int	<i>aio_reqprio</i>	Request priority offset.
1700	struct sigevent	<i>aio_sigevent</i>	Signal number and value.
1701	int	<i>aio_lio_opcode</i>	Operation to be performed.

1702 The *aio_fildes* element is the file descriptor on which the asynchronous operation is performed.

1703 If `O_APPEND` is not set for the file descriptor *aio_fildes* and if *aio_fildes* is associated with a
1704 device that is capable of seeking, then the requested operation takes place at the absolute
1705 position in the file as given by *aio_offset*, as if *lseek()* were called immediately prior to the

1706 operation with an *offset* argument equal to *aio_offset* and a *whence* argument equal to `SEEK_SET`.
 1707 If `O_APPEND` is set for the file descriptor, or if *aio_fildes* is associated with a device that is
 1708 incapable of seeking, write operations append to the file in the same order as the calls were
 1709 made, with the following exception: under implementation-defined circumstances, such as
 1710 operation on a multi-processor or when requests of differing priorities are submitted at the same
 1711 time, the ordering restriction may be relaxed. Since there is no way for a strictly conforming
 1712 application to determine whether this relaxation applies, all strictly conforming applications
 1713 which rely on ordering of output shall be written in such a way that they will operate correctly if
 1714 the relaxation applies. After a successful call to enqueue an asynchronous I/O operation, the
 1715 value of the file offset for the file is unspecified. The *aio_nbytes* and *aio_buf* elements are the same
 1716 as the *nbyte* and *buf* arguments defined by `read()` and `write()`, respectively.

1717 If `_POSIX_PRIORITIZED_IO` and `_POSIX_PRIORITY_SCHEDULING` are defined, then
 1718 asynchronous I/O is queued in priority order, with the priority of each asynchronous operation
 1719 based on the current scheduling priority of the calling process. The *aio_reqprio* member can be
 1720 used to lower (but not raise) the asynchronous I/O operation priority and is within the range
 1721 zero through `{AIO_PRIO_DELTA_MAX}`, inclusive. Unless both `_POSIX_PRIORITIZED_IO` and
 1722 `_POSIX_PRIORITY_SCHEDULING` are defined, the order of processing asynchronous I/O
 1723 requests is unspecified. When both `_POSIX_PRIORITIZED_IO` and
 1724 `_POSIX_PRIORITY_SCHEDULING` are defined, the order of processing of requests submitted
 1725 by processes whose schedulers are not `SCHED_FIFO`, `SCHED_RR`, or `SCHED_SPORADIC` is
 1726 unspecified. The priority of an asynchronous request is computed as (process scheduling
 1727 priority) minus *aio_reqprio*. The priority assigned to each asynchronous I/O request is an
 1728 indication of the desired order of execution of the request relative to other asynchronous I/O
 1729 requests for this file. If `_POSIX_PRIORITIZED_IO` is defined, requests issued with the same
 1730 priority to a character special file are processed by the underlying device in FIFO order; the order
 1731 of processing of requests of the same priority issued to files that are not character special files is
 1732 unspecified. Numerically higher priority values indicate requests of higher priority. The value of
 1733 *aio_reqprio* has no effect on process scheduling priority. When prioritized asynchronous I/O
 1734 requests to the same file are blocked waiting for a resource required for that I/O operation, the
 1735 higher-priority I/O requests shall be granted the resource before lower-priority I/O requests are
 1736 granted the resource. The relative priority of asynchronous I/O and synchronous I/O is
 1737 implementation-defined. If `_POSIX_PRIORITIZED_IO` is defined, the implementation shall
 1738 define for which files I/O prioritization is supported.

1739 The *aio_sigevent* determines how the calling process shall be notified upon I/O completion, as
 1740 specified in Section 2.4.1 (on page 28). If *aio_sigevent.sigev_notify* is `SIGEV_NONE`, then no signal
 1741 shall be posted upon I/O completion, but the error status for the operation and the return status
 1742 for the operation shall be set appropriately.

1743 The *aio_lio_opcode* field is used only by the `lio_listio()` call. The `lio_listio()` call allows multiple
 1744 asynchronous I/O operations to be submitted at a single time. The function takes as an
 1745 argument an array of pointers to `aiocb` structures. Each `aiocb` structure indicates the operation to
 1746 be performed (read or write) via the *aio_lio_opcode* field.

1747 The address of the `aiocb` structure is used as a handle for retrieving the error status and return
 1748 status of the asynchronous operation while it is in progress.

1749 The `aiocb` structure and the data buffers associated with the asynchronous I/O operation are
 1750 being used by the system for asynchronous I/O while, and only while, the error status of the
 1751 asynchronous operation is equal to `[EINPROGRESS]`. Applications shall not modify the `aiocb`
 1752 structure while the structure is being used by the system for asynchronous I/O.

1753 The return status of the asynchronous operation is the number of bytes transferred by the I/O
 1754 operation. If the error status is set to indicate an error completion, then the return status is set to

1755 the return value that the corresponding *read()*, *write()*, or *fsync()* call would have returned.
 1756 When the error status is not equal to [EINPROGRESS], the return status shall reflect the return
 1757 status of the corresponding synchronous operation.

1758 **2.8.3 Memory Management**

1759 *2.8.3.1 Memory Locking*

1760 MLR Range memory locking operations are defined in terms of pages. Implementations may restrict |
 1761 the size and alignment of range lockings to be on page-size boundaries. The page size, in bytes, |
 1762 is the value of the configurable system variable {PAGESIZE}. If an implementation has no |
 1763 restrictions on size or alignment, it may specify a 1-byte page size.

1764 ML|MLR Memory locking guarantees the residence of portions of the address space. It is |
 1765 implementation-defined whether locking memory guarantees fixed translation between virtual |
 1766 addresses (as seen by the process) and physical addresses. Per-process memory locks are not |
 1767 inherited across a *fork()*, and all memory locks owned by a process are unlocked upon *exec* or
 1768 process termination. Unmapping of an address range removes any memory locks established on
 1769 that address range by this process.

1770 *2.8.3.2 Memory Mapped Files*

1771 MF The functionality described in this section is dependent on support of the Memory Mapped Files
 1772 option (and the rest of this section is not further shaded for this option).

1773 Range memory mapping operations are defined in terms of pages. Implementations may
 1774 restrict the size and alignment of range mappings to be on page-size boundaries. The page size,
 1775 in bytes, is the value of the configurable system variable {PAGESIZE}. If an implementation has
 1776 no restrictions on size or alignment, it may specify a 1-byte page size.

1777 Memory mapped files provide a mechanism that allows a process to access files by directly
 1778 incorporating file data into its address space. Once a file is mapped into a process address space,
 1779 the data can be manipulated as memory. If more than one process maps a file, its contents are
 1780 shared among them. If the mappings allow shared write access, then data written into the
 1781 memory object through the address space of one process appears in the address spaces of all
 1782 processes that similarly map the same portion of the memory object.

1783 SHM Shared memory objects are named regions of storage that may be independent of the file system
 1784 and can be mapped into the address space of one or more processes to allow them to share the
 1785 associated memory.

1786 SHM An *unlink()* of a file or *shm_unlink()* of a shared memory object, while causing the removal of the
 1787 name, does not unmap any mappings established for the object. Once the name has been
 1788 removed, the contents of the memory object are preserved as long as it is referenced. The
 1789 memory object remains referenced as long as a process has the memory object open or has some
 1790 area of the memory object mapped.

1791 *2.8.3.3 Memory Protection*

1792 MPR MF The functionality described in this section is dependent on support of the Memory Protection
 1793 and Memory Mapped Files option (and the rest of this section is not further shaded for these
 1794 options).

1795 When an object is mapped, various application accesses to the mapped region may result in
 1796 signals. In this context, SIGBUS is used to indicate an error using the mapped object, and
 1797 SIGSEGV is used to indicate a protection violation or misuse of an address:

- 1798 • A mapping may be restricted to disallow some types of access.
- 1799 • Write attempts to memory that was mapped without write access, or any access to memory
1800 mapped PROT_NONE, shall result in a SIGSEGV signal.
- 1801 • References to unmapped addresses shall result in a SIGSEGV signal.
- 1802 • Reference to whole pages within the mapping, but beyond the current length of the object,
1803 shall result in a SIGBUS signal.
- 1804 • The size of the object is unaffected by access beyond the end of the object (even if a SIGBUS is
1805 not generated).

1806 2.8.3.4 *Typed Memory Objects*

1807 TYM The functionality described in this section is dependent on support of the Typed Memory
1808 Objects option (and the rest of this section is not further shaded for this option).

1809 Implementations may support the Typed Memory Objects option without supporting the
1810 Memory Mapped Files option or the Shared Memory Objects option. Typed memory objects are
1811 implementation-configurable named storage pools accessible from one or more processors in a
1812 system, each via one or more ports, such as backplane buses, LANs, I/O channels, and so on.
1813 Each valid combination of a storage pool and a port is identified through a name that is defined
1814 at system configuration time, in an implementation-defined manner; the name may be
1815 independent of the file system. Using this name, a typed memory object can be opened and
1816 mapped into process address space. For a given storage pool and port, it is necessary to support
1817 both dynamic allocation from the pool as well as mapping at an application-supplied offset
1818 within the pool; when dynamic allocation has been performed, subsequent deallocation must be
1819 supported. Lastly, accessing typed memory objects from different ports requires a method for
1820 obtaining the offset and length of contiguous storage of a region of typed memory (dynamically
1821 allocated or not); this allows typed memory to be shared among processes and/or processors
1822 while being accessed from the desired port.

1823 2.8.4 **Process Scheduling**

1824 PS The functionality described in this section is dependent on support of the Process Scheduling
1825 option (and the rest of this section is not further shaded for this option).

1826 **Scheduling Policies**

1827 The scheduling semantics described in this volume of IEEE Std 1003.1-2001 are defined in terms
1828 of a conceptual model that contains a set of thread lists. No implementation structures are
1829 necessarily implied by the use of this conceptual model. It is assumed that no time elapses
1830 during operations described using this model, and therefore no simultaneous operations are
1831 possible. This model discusses only processor scheduling for runnable threads, but it should be
1832 noted that greatly enhanced predictability of realtime applications results if the sequencing of
1833 other resources takes processor scheduling policy into account.

1834 There is, conceptually, one thread list for each priority. A runnable thread will be on the thread
1835 list for that thread's priority. Multiple scheduling policies shall be provided. Each non-empty
1836 thread list is ordered, contains a head as one end of its order, and a tail as the other. The purpose
1837 of a scheduling policy is to define the allowable operations on this set of lists (for example,
1838 moving threads between and within lists).

1839 Each process shall be controlled by an associated scheduling policy and priority. These
1840 parameters may be specified by explicit application execution of the *sched_setscheduler()* or
1841 *sched_setparam()* functions.

1842 Each thread shall be controlled by an associated scheduling policy and priority. These
 1843 parameters may be specified by explicit application execution of the *pthread_setschedparam()*
 1844 function.

1845 Associated with each policy is a priority range. Each policy definition shall specify the minimum
 1846 priority range for that policy. The priority ranges for each policy may but need not overlap the
 1847 priority ranges of other policies.

1848 A conforming implementation shall select the thread that is defined as being at the head of the
 1849 highest priority non-empty thread list to become a running thread, regardless of its associated
 1850 policy. This thread is then removed from its thread list.

1851 Four scheduling policies are specifically required. Other implementation-defined scheduling
 1852 policies may be defined. The following symbols are defined in the Base Definitions volume of
 1853 IEEE Std 1003.1-2001, <sched.h>:

1854 SCHED_FIFO First in, first out (FIFO) scheduling policy.

1855 SCHED_RR Round robin scheduling policy.

1856 ss SCHED_SPORADIC Sporadic server scheduling policy.

1857 SCHED_OTHER Another scheduling policy.

1858 The values of these symbols shall be distinct.

1859 SCHED_FIFO

1860 Conforming implementations shall include a scheduling policy called the FIFO scheduling
 1861 policy.

1862 Threads scheduled under this policy are chosen from a thread list that is ordered by the time its
 1863 threads have been on the list without being executed; generally, the head of the list is the thread
 1864 that has been on the list the longest time, and the tail is the thread that has been on the list the
 1865 shortest time.

1866 Under the SCHED_FIFO policy, the modification of the definitional thread lists is as follows:

- 1867 1. When a running thread becomes a preempted thread, it becomes the head of the thread list
 1868 for its priority.
- 1869 2. When a blocked thread becomes a runnable thread, it becomes the tail of the thread list for
 1870 its priority.
- 1871 3. When a running thread calls the *sched_setscheduler()* function, the process specified in the
 1872 function call is modified to the specified policy and the priority specified by the *param*
 1873 argument.
- 1874 4. When a running thread calls the *sched_setparam()* function, the priority of the process
 1875 specified in the function call is modified to the priority specified by the *param* argument.
- 1876 5. When a running thread calls the *pthread_setschedparam()* function, the thread specified in
 1877 the function call is modified to the specified policy and the priority specified by the *param*
 1878 argument.
- 1879 6. When a running thread calls the *pthread_setschedprio()* function, the thread specified in the
 1880 function call is modified to the priority specified by the *prio* argument.
- 1881 7. If a thread whose policy or priority has been modified other than by *pthread_setschedprio()*
 1882 is a running thread or is runnable, it then becomes the tail of the thread list for its new
 1883 priority.

- 1884 8. If a thread whose policy or priority has been modified by *pthread_setschedprio()* is a
 1885 running thread or is runnable, the effect on its position in the thread list depends on the
 1886 direction of the modification, as follows:
- 1887 a. If the priority is raised, the thread becomes the tail of the thread list.
 - 1888 b. If the priority is unchanged, the thread does not change position in the thread list.
 - 1889 c. If the priority is lowered, the thread becomes the head of the thread list.
- 1890 9. When a running thread issues the *sched_yield()* function, the thread becomes the tail of the
 1891 thread list for its priority.
- 1892 10. At no other time is the position of a thread with this scheduling policy within the thread
 1893 lists affected.

1894 For this policy, valid priorities shall be within the range returned by the *sched_get_priority_max()*
 1895 and *sched_get_priority_min()* functions when SCHED_FIFO is provided as the parameter.
 1896 Conforming implementations shall provide a priority range of at least 32 priorities for this
 1897 policy.

1898 SCHED_RR

1899 Conforming implementations shall include a scheduling policy called the “round robin”
 1900 scheduling policy. This policy shall be identical to the SCHED_FIFO policy with the additional
 1901 condition that when the implementation detects that a running thread has been executing as a
 1902 running thread for a time period of the length returned by the *sched_rr_get_interval()* function or
 1903 longer, the thread shall become the tail of its thread list and the head of that thread list shall be
 1904 removed and made a running thread.

1905 The effect of this policy is to ensure that if there are multiple SCHED_RR threads at the same
 1906 priority, one of them does not monopolize the processor. An application should not rely only on
 1907 the use of SCHED_RR to ensure application progress among multiple threads if the application
 1908 includes threads using the SCHED_FIFO policy at the same or higher priority levels or
 1909 SCHED_RR threads at a higher priority level.

1910 A thread under this policy that is preempted and subsequently resumes execution as a running
 1911 thread completes the unexpired portion of its round robin interval time period.

1912 For this policy, valid priorities shall be within the range returned by the *sched_get_priority_max()*
 1913 and *sched_get_priority_min()* functions when SCHED_RR is provided as the parameter.
 1914 Conforming implementations shall provide a priority range of at least 32 priorities for this
 1915 policy.

1916 SCHED_SPORADIC

1917 SS|TSP The functionality described in this section is dependent on support of the Process Sporadic
 1918 Server or Thread Sporadic Server options (and the rest of this section is not further shaded for
 1919 these options).

1920 If `_POSIX_SPORADIC_SERVER` or `_POSIX_THREAD_SPORADIC_SERVER` is defined, the
 1921 implementation shall include a scheduling policy identified by the value SCHED_SPORADIC.

1922 The sporadic server policy is based primarily on two parameters: the replenishment period and
 1923 the available execution capacity. The replenishment period is given by the *sched_ss_repl_period*
 1924 member of the **sched_param** structure. The available execution capacity is initialized to the
 1925 value given by the *sched_ss_init_budget* member of the same parameter. The sporadic server
 1926 policy is identical to the SCHED_FIFO policy with some additional conditions that cause the
 1927 thread’s assigned priority to be switched between the values specified by the *sched_priority* and

1928 *sched_ss_low_priority* members of the **sched_param** structure.

1929 The priority assigned to a thread using the sporadic server scheduling policy is determined in
 1930 the following manner: if the available execution capacity is greater than zero and the number of
 1931 pending replenishment operations is strictly less than *sched_ss_max_repl*, the thread is assigned
 1932 the priority specified by *sched_priority*; otherwise, the assigned priority shall be
 1933 *sched_ss_low_priority*. If the value of *sched_priority* is less than or equal to the value of
 1934 *sched_ss_low_priority*, the results are undefined. When active, the thread shall belong to the
 1935 thread list corresponding to its assigned priority level, according to the mentioned priority
 1936 assignment. The modification of the available execution capacity and, consequently of the
 1937 assigned priority, is done as follows:

- 1938 1. When the thread at the head of the *sched_priority* list becomes a running thread, its
 1939 execution time shall be limited to at most its available execution capacity, plus the
 1940 resolution of the execution time clock used for this scheduling policy. This resolution shall
 1941 be implementation-defined.
- 1942 2. Each time the thread is inserted at the tail of the list associated with *sched_priority*—
 1943 because as a blocked thread it became runnable with priority *sched_priority* or because a
 1944 replenishment operation was performed—the time at which this operation is done is
 1945 posted as the *activation_time*.
- 1946 3. When the running thread with assigned priority equal to *sched_priority* becomes a
 1947 preempted thread, it becomes the head of the thread list for its priority, and the execution
 1948 time consumed is subtracted from the available execution capacity. If the available
 1949 execution capacity would become negative by this operation, it shall be set to zero.
- 1950 4. When the running thread with assigned priority equal to *sched_priority* becomes a blocked
 1951 thread, the execution time consumed is subtracted from the available execution capacity,
 1952 and a replenishment operation is scheduled, as described in 6 and 7. If the available
 1953 execution capacity would become negative by this operation, it shall be set to zero.
- 1954 5. When the running thread with assigned priority equal to *sched_priority* reaches the limit
 1955 imposed on its execution time, it becomes the tail of the thread list for
 1956 *sched_ss_low_priority*, the execution time consumed is subtracted from the available
 1957 execution capacity (which becomes zero), and a replenishment operation is scheduled, as
 1958 described in 6 and 7.
- 1959 6. Each time a replenishment operation is scheduled, the amount of execution capacity to be
 1960 replenished, *replenish_amount*, is set equal to the execution time consumed by the thread
 1961 since the *activation_time*. The replenishment is scheduled to occur at *activation_time* plus
 1962 *sched_ss_repl_period*. If the scheduled time obtained is before the current time, the
 1963 replenishment operation is carried out immediately. Several replenishment operations may
 1964 be pending at the same time, each of which will be serviced at its respective scheduled
 1965 time. With the above rules, the number of replenishment operations simultaneously
 1966 pending for a given thread that is scheduled under the sporadic server policy shall not be
 1967 greater than *sched_ss_max_repl*.
- 1968 7. A replenishment operation consists of adding the corresponding *replenish_amount* to the
 1969 available execution capacity at the scheduled time. If, as a consequence of this operation,
 1970 the execution capacity would become larger than *sched_ss_initial_budget*, it shall be
 1971 rounded down to a value equal to *sched_ss_initial_budget*. Additionally, if the thread was
 1972 runnable or running, and had assigned priority equal to *sched_ss_low_priority*, then it
 1973 becomes the tail of the thread list for *sched_priority*.

1974 Execution time is defined in Section 2.2.2 (on page 14).

1975 For this policy, changing the value of a CPU-time clock via *clock_settime()* shall have no effect on
 1976 its behavior.

1977 For this policy, valid priorities shall be within the range returned by the *sched_get_priority_min()*
 1978 and *sched_get_priority_max()* functions when SCHED_SPORADIC is provided as the parameter.
 1979 Conforming implementations shall provide a priority range of at least 32 distinct priorities for
 1980 this policy.

1981 **SCHED_OTHER**

1982 Conforming implementations shall include one scheduling policy identified as SCHED_OTHER
 1983 (which may execute identically with either the FIFO or round robin scheduling policy). The
 1984 effect of scheduling threads with the SCHED_OTHER policy in a system in which other threads
 1985 ss are executing under SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC is implementation-
 1986 defined.

1987 This policy is defined to allow strictly conforming applications to be able to indicate in a
 1988 portable manner that they no longer need a realtime scheduling policy.

1989 For threads executing under this policy, the implementation shall use only priorities within the
 1990 range returned by the *sched_get_priority_max()* and *sched_get_priority_min()* functions when
 1991 SCHED_OTHER is provided as the parameter.

1992 **2.8.5 Clocks and Timers**

1993 TMR The functionality described in this section is dependent on support of the Timers option (and the
 1994 rest of this section is not further shaded for this option).

1995 The <time.h> header defines the types and manifest constants used by the timing facility.

1996 **Time Value Specification Structures**

1997 Many of the timing facility functions accept or return time value specifications. A time value
 1998 structure **timespec** specifies a single time value and includes at least the following members:

1999
 2000
 2001
 2002

Member Type	Member Name	Description
time_t	<i>tv_sec</i>	Seconds.
long	<i>tv_nsec</i>	Nanoseconds.

2003 The *tv_nsec* member is only valid if greater than or equal to zero, and less than the number of
 2004 nanoseconds in a second (1 000 million). The time interval described by this structure is (*tv_sec* *
 2005 10⁹ + *tv_nsec*) nanoseconds.

2006 A time value structure **itimerspec** specifies an initial timer value and a repetition interval for use
 2007 by the per-process timer functions. This structure includes at least the following members:

2008
 2009
 2010
 2011

Member Type	Member Name	Description
struct timespec	<i>it_interval</i>	Timer period.
struct timespec	<i>it_value</i>	Timer expiration.

2012 If the value described by *it_value* is non-zero, it indicates the time to or time of the next timer
 2013 expiration (for relative and absolute timer values, respectively). If the value described by *it_value*
 2014 is zero, the timer shall be disarmed.

2015 If the value described by *it_interval* is non-zero, it specifies an interval which shall be used in
 2016 reloading the timer when it expires; that is, a periodic timer is specified. If the value described by

2017 *it_interval* is zero, the timer is disarmed after its next expiration; that is, a one-shot timer is
2018 specified.

2019 **Timer Event Notification Control Block**

2020 RTS Per-process timers may be created that notify the process of timer expirations by queuing a
2021 realtime extended signal. The **sigevent** structure, defined in the Base Definitions volume of
2022 IEEE Std 1003.1-2001, <**signal.h**>, is used in creating such a timer. The **sigevent** structure
2023 contains the signal number and an application-specific data value which shall be used when
2024 notifying the calling process of timer expiration events.

2025 **Manifest Constants**

2026 The following constants are defined in the Base Definitions volume of IEEE Std 1003.1-2001,
2027 <**time.h**>:

2028 **CLOCK_REALTIME** The identifier for the system-wide realtime clock.

2029 **TIMER_ABSTIME** Flag indicating time is absolute with respect to the clock associated
2030 with a timer.

2031 MON **CLOCK_MONOTONIC** The identifier for the system-wide monotonic clock, which is defined
2032 as a clock whose value cannot be set via *clock_settime()* and which
2033 cannot have backward clock jumps. The maximum possible clock
2034 jump is implementation-defined.

2035 MON The maximum allowable resolution for **CLOCK_REALTIME** and **CLOCK_MONOTONIC** clocks
2036 and all time services based on these clocks is represented by `{_POSIX_CLOCKRES_MIN}` and
2037 shall be defined as 20 ms (1/50 of a second). Implementations may support smaller values of
2038 resolution for these clocks to provide finer granularity time bases. The actual resolution
2039 supported by an implementation for a specific clock is obtained using the *clock_getres()* function.
2040 If the actual resolution supported for a time service based on one of these clocks differs from the
2041 resolution supported for that clock, the implementation shall document this difference.

2042 MON The minimum allowable maximum value for **CLOCK_REALTIME** and **CLOCK_MONOTONIC**
2043 clocks and all absolute time services based on them is the same as that defined by the ISO C
2044 standard for the **time_t** type. If the maximum value supported by a time service based on one of
2045 these clocks differs from the maximum value supported by that clock, the implementation shall
2046 document this difference.

2047 **Execution Time Monitoring**

2048 CPT If **_POSIX_CPUTIME** is defined, process CPU-time clocks shall be supported in addition to the
2049 clocks described in **Manifest Constants**.

2050 TCT If **_POSIX_THREAD_CPUTIME** is defined, thread CPU-time clocks shall be supported.

2051 CPT|TCT CPU-time clocks measure execution or CPU time, which is defined in the Base Definitions
2052 volume of IEEE Std 1003.1-2001, Section 3.117, CPU Time (Execution Time). The mechanism
2053 used to measure execution time is described in the Base Definitions volume of
2054 IEEE Std 1003.1-2001, Section 4.9, Measurement of Execution Time.

2055 CPT If **_POSIX_CPUTIME** is defined, the following constant of the type **clockid_t** is defined in
2056 <**time.h**>:

2057 **CLOCK_PROCESS_CPUTIME_ID**

2058 When this value of the type **clockid_t** is used in a *clock()* or *timer*()* function call, it is
2059 interpreted as the identifier of the CPU-time clock associated with the process making the

2060 function call.

2061

2062 TCT If `_POSIX_THREAD_CPUTIME` is defined, the following constant of the type `clockid_t` is
2063 defined in `<time.h>`:

2064 `CLOCK_THREAD_CPUTIME_ID`

2065 When this value of the type `clockid_t` is used in a `clock()` or `timer*()` function call, it is
2066 interpreted as the identifier of the CPU-time clock associated with the thread making the
2067 function call.

2068 2.9 Threads

2069 THR The functionality described in this section is dependent on support of the Threads option (and
2070 the rest of this section is not further shaded for this option).

2071 This section defines functionality to support multiple flows of control, called “threads”, within a
2072 process. For the definition of threads, see the Base Definitions volume of IEEE Std 1003.1-2001,
2073 Section 3.393, Thread.

2074 The specific functional areas covered by threads and their scope include:

- 2075 • Thread management: the creation, control, and termination of multiple flows of control in the
2076 same process under the assumption of a common shared address space
- 2077 • Synchronization primitives optimized for tightly coupled operation of multiple control flows
2078 in a common, shared address space

2079 2.9.1 Thread-Safety

2080 All functions defined by this volume of IEEE Std 1003.1-2001 shall be thread-safe, except that the
2081 following functions¹ need not be thread-safe.

2082	<code>asctime()</code>	<code>ecvt()</code>	<code>gethostent()</code>	<code>getutxline()</code>	<code>putc_unlocked()</code>
2083	<code>basename()</code>	<code>encrypt()</code>	<code>getlogin()</code>	<code>gmtime()</code>	<code>putchar_unlocked()</code>
2084	<code>catgets()</code>	<code>endgrent()</code>	<code>getnetbyaddr()</code>	<code>hcreate()</code>	<code>putenv()</code>
2085	<code>crypt()</code>	<code>endpwent()</code>	<code>getnetbyname()</code>	<code>hdestroy()</code>	<code>pututxline()</code>
2086	<code>ctime()</code>	<code>endutxent()</code>	<code>getnetent()</code>	<code>hsearch()</code>	<code>rand()</code>
2087	<code>dbm_clearerr()</code>	<code>fcvt()</code>	<code>getopt()</code>	<code>inet_ntoa()</code>	<code>readdir()</code>
2088	<code>dbm_close()</code>	<code>ftw()</code>	<code>getprotobyname()</code>	<code>l64a()</code>	<code>setenv()</code>
2089	<code>dbm_delete()</code>	<code>gcvt()</code>	<code>getprotobyname()</code>	<code>lgamma()</code>	<code>setgrent()</code>
2090	<code>dbm_error()</code>	<code>getc_unlocked()</code>	<code>getprotoent()</code>	<code>lgammaf()</code>	<code>setkey()</code>
2091	<code>dbm_fetch()</code>	<code>getchar_unlocked()</code>	<code>getpwent()</code>	<code>lgammal()</code>	<code>setpwent()</code>
2092	<code>dbm_firstkey()</code>	<code>getdate()</code>	<code>getpwnam()</code>	<code>localeconv()</code>	<code>setutxent()</code>
2093	<code>dbm_nextkey()</code>	<code>getenv()</code>	<code>getpwuid()</code>	<code>localtime()</code>	<code>strerror()</code>
2094	<code>dbm_open()</code>	<code>getgrent()</code>	<code>getservbyname()</code>	<code>lrand48()</code>	<code>strtok()</code>
2095	<code>dbm_store()</code>	<code>getgrgid()</code>	<code>getservbyport()</code>	<code>mrnd48()</code>	<code>ttyname()</code>
2096	<code>dirname()</code>	<code>getgrnam()</code>	<code>getservent()</code>	<code>nftw()</code>	<code>unsetenv()</code>

2097 _____

2098 1. The functions in the table are not shaded to denote applicable options. Individual reference pages should be consulted.

2099 *derror()* *gethostbyaddr()* *getutxent()* *nl_langinfo()* *wcstombs()*
 2100 *drand48()* *gethostbyname()* *getutxid()* *ptsname()* *wctomb()*

2101 The *ctermid()* and *tmpnam()* functions need not be thread-safe if passed a NULL argument. The
 2102 *wctomb()* and *wcsrombs()* functions need not be thread-safe if passed a NULL *ps* argument.

2103 Implementations shall provide internal synchronization as necessary in order to satisfy this
 2104 requirement.

2105 2.9.2 Thread IDs

2106 Although implementations may have thread IDs that are unique in a system, applications
 2107 should only assume that thread IDs are usable and unique within a single process. The effect of
 2108 calling any of the functions defined in this volume of IEEE Std 1003.1-2001 and passing as an
 2109 argument the thread ID of a thread from another process is unspecified. A conforming
 2110 implementation is free to reuse a thread ID after the thread terminates if it was created with the
 2111 *detachstate* attribute set to *PTHREAD_CREATE_DETACHED* or if *pthread_detach()* or
 2112 *pthread_join()* has been called for that thread. If a thread is detached, its thread ID is invalid for
 2113 use as an argument in a call to *pthread_detach()* or *pthread_join()*.

2114 2.9.3 Thread Mutexes

2115 A thread that has blocked shall not prevent any unblocked thread that is eligible to use the same
 2116 processing resources from eventually making forward progress in its execution. Eligibility for
 2117 processing resources is determined by the scheduling policy.

2118 A thread shall become the owner of a mutex, *m*, when one of the following occurs:

- 2119 • It returns successfully from *pthread_mutex_lock()* with *m* as the *mutex* argument.
- 2120 • It returns successfully from *pthread_mutex_trylock()* with *m* as the *mutex* argument.
- 2121 TMO • It returns successfully from *pthread_mutex_timedlock()* with *m* as the *mutex* argument.
- 2122 • It returns (successfully or not) from *pthread_cond_wait()* with *m* as the *mutex* argument
 2123 (except as explicitly indicated otherwise for certain errors).
- 2124 • It returns (successfully or not) from *pthread_cond_timedwait()* with *m* as the *mutex* argument
 2125 (except as explicitly indicated otherwise for certain errors).

2126 The thread shall remain the owner of *m* until one of the following occurs:

- 2127 • It executes *pthread_mutex_unlock()* with *m* as the *mutex* argument
- 2128 • It blocks in a call to *pthread_cond_wait()* with *m* as the *mutex* argument.
- 2129 • It blocks in a call to *pthread_cond_timedwait()* with *m* as the *mutex* argument.

2130 The implementation shall behave as if at all times there is at most one owner of any mutex.

2131 A thread that becomes the owner of a mutex is said to have “acquired” the mutex and the mutex
 2132 is said to have become “locked”; when a thread gives up ownership of a mutex it is said to have
 2133 “released” the mutex and the mutex is said to have become “unlocked”.

2134 2.9.4 Thread Scheduling

2135 TPS The functionality described in this section is dependent on support of the Thread Execution
2136 Scheduling option (and the rest of this section is not further shaded for this option).

2137 Thread Scheduling Attributes

2138 In support of the scheduling function, threads have attributes which are accessed through the
2139 **pthread_attr_t** thread creation attributes object.

2140 The *contentionscope* attribute defines the scheduling contention scope of the thread to be either
2141 PTHREAD_SCOPE_PROCESS or PTHREAD_SCOPE_SYSTEM.

2142 The *inheritsched* attribute specifies whether a newly created thread is to inherit the scheduling
2143 attributes of the creating thread or to have its scheduling values set according to the other
2144 scheduling attributes in the **pthread_attr_t** object.

2145 The *schedpolicy* attribute defines the scheduling policy for the thread. The *schedparam* attribute
2146 defines the scheduling parameters for the thread. The interaction of threads having different
2147 policies within a process is described as part of the definition of those policies.

2148 If the Thread Execution Scheduling option is defined, and the *schedpolicy* attribute specifies one
2149 of the priority-based policies defined under this option, the *schedparam* attribute contains the
2150 scheduling priority of the thread. A conforming implementation ensures that the priority value
2151 in *schedparam* is in the range associated with the scheduling policy when the thread attributes
2152 object is used to create a thread, or when the scheduling attributes of a thread are dynamically
2153 modified. The meaning of the priority value in *schedparam* is the same as that of *priority*.

2154 TSP If `_POSIX_THREAD_SPORADIC_SERVER` is defined, the *schedparam* attribute supports four
2155 new members that are used for the sporadic server scheduling policy. These members are
2156 *sched_ss_low_priority*, *sched_ss_repl_period*, *sched_ss_init_budget*, and *sched_ss_max_repl*. The
2157 meaning of these attributes is the same as in the definitions that appear under Section 2.8.4 (on
2158 page 44).

2159 When a process is created, its single thread has a scheduling policy and associated attributes
2160 equal to the process' policy and attributes. The default scheduling contention scope value is
2161 implementation-defined. The default values of other scheduling attributes are implementation-
2162 defined.

2163 Thread Scheduling Contention Scope

2164 The scheduling contention scope of a thread defines the set of threads with which the thread
2165 competes for use of the processing resources. The scheduling operation selects at most one
2166 thread to execute on each processor at any point in time and the thread's scheduling attributes
2167 (for example, *priority*), whether under process scheduling contention scope or system scheduling
2168 contention scope, are the parameters used to determine the scheduling decision.

2169 The scheduling contention scope, in the context of scheduling a mixed scope environment,
2170 affects threads as follows:

- 2171 • A thread created with PTHREAD_SCOPE_SYSTEM scheduling contention scope contends
2172 for resources with all other threads in the same scheduling allocation domain relative to their
2173 system scheduling attributes. The system scheduling attributes of a thread created with
2174 PTHREAD_SCOPE_SYSTEM scheduling contention scope are the scheduling attributes with
2175 which the thread was created. The system scheduling attributes of a thread created with
2176 PTHREAD_SCOPE_PROCESS scheduling contention scope are the implementation-defined
2177 mapping into system attribute space of the scheduling attributes with which the thread was
2178 created.

- 2179 • Threads created with PTHREAD_SCOPE_PROCESS scheduling contention scope contend
 2180 directly with other threads within their process that were created with
 2181 PTHREAD_SCOPE_PROCESS scheduling contention scope. The contention is resolved
 2182 based on the threads' scheduling attributes and policies. It is unspecified how such threads
 2183 are scheduled relative to threads in other processes or threads with
 2184 PTHREAD_SCOPE_SYSTEM scheduling contention scope.
- 2185 • Conforming implementations shall support the PTHREAD_SCOPE_PROCESS scheduling
 2186 contention scope, the PTHREAD_SCOPE_SYSTEM scheduling contention scope, or both.

2187 **Scheduling Allocation Domain**

2188 Implementations shall support scheduling allocation domains containing one or more
 2189 processors. It should be noted that the presence of multiple processors does not automatically
 2190 indicate a scheduling allocation domain size greater than one. Conforming implementations on
 2191 multi-processors may map all or any subset of the CPUs to one or multiple scheduling allocation
 2192 domains, and could define these scheduling allocation domains on a per-thread, per-process, or
 2193 per-system basis, depending on the types of applications intended to be supported by the
 2194 implementation. The scheduling allocation domain is independent of scheduling contention
 2195 scope, as the scheduling contention scope merely defines the set of threads with which a thread
 2196 contends for processor resources, while scheduling allocation domain defines the set of
 2197 processors for which it contends. The semantics of how this contention is resolved among
 2198 threads for processors is determined by the scheduling policies of the threads.

2199 The choice of scheduling allocation domain size and the level of application control over
 2200 scheduling allocation domains is implementation-defined. Conforming implementations may
 2201 change the size of scheduling allocation domains and the binding of threads to scheduling
 2202 allocation domains at any time.

2203 For application threads with scheduling allocation domains of size equal to one, the scheduling
 2204 rules defined for SCHED_FIFO and SCHED_RR shall be used; see **Scheduling Policies** (on page
 2205 44). All threads with system scheduling contention scope, regardless of the processes in which
 2206 they reside, compete for the processor according to their priorities. Threads with process
 2207 scheduling contention scope compete only with other threads with process scheduling
 2208 contention scope within their process.

2209 For application threads with scheduling allocation domains of size greater than one, the rules
 2210 TSP defined for SCHED_FIFO, SCHED_RR, and SCHED_SPORADIC shall be used in an
 2211 implementation-defined manner. Each thread with system scheduling contention scope
 2212 competes for the processors in its scheduling allocation domain in an implementation-defined
 2213 manner according to its priority. Threads with process scheduling contention scope are
 2214 scheduled relative to other threads within the same scheduling contention scope in the process.

2215 TSP If `_POSIX_THREAD_SPORADIC_SERVER` is defined, the rules defined for SCHED_SPORADIC
 2216 in **Scheduling Policies** (on page 44) shall be used in an implementation-defined manner for
 2217 application threads whose scheduling allocation domain size is greater than one.

2218 **Scheduling Documentation**

2219 If `_POSIX_PRIORITY_SCHEDULING` is defined, then any scheduling policies beyond
 2220 TSP `SCHED_OTHER`, `SCHED_FIFO`, `SCHED_RR`, and `SCHED_SPORADIC`, as well as the effects of
 2221 the scheduling policies indicated by these other values, and the attributes required in order to
 2222 support such a policy, are implementation-defined. Furthermore, the implementation shall
 2223 document the effect of all processor scheduling allocation domain values supported for these
 2224 policies.

2225 **2.9.5 Thread Cancellation**

2226 The thread cancellation mechanism allows a thread to terminate the execution of any other
 2227 thread in the process in a controlled manner. The target thread (that is, the one that is being
 2228 canceled) is allowed to hold cancellation requests pending in a number of ways and to perform
 2229 application-specific cleanup processing when the notice of cancellation is acted upon.

2230 Cancellation is controlled by the cancellation control functions. Each thread maintains its own
 2231 cancelability state. Cancellation may only occur at cancellation points or when the thread is
 2232 asynchronously cancelable.

2233 The thread cancellation mechanism described in this section depends upon programs having set
 2234 *deferred* cancelability state, which is specified as the default. Applications shall also carefully
 2235 follow static lexical scoping rules in their execution behavior. For example, use of *setjmp()*,
 2236 *return*, *goto*, and so on, to leave user-defined cancellation scopes without doing the necessary
 2237 scope pop operation results in undefined behavior.

2238 Use of asynchronous cancelability while holding resources which potentially need to be released
 2239 may result in resource loss. Similarly, cancellation scopes may only be safely manipulated
 2240 (pushed and popped) when the thread is in the *deferred* or *disabled* cancelability states.

2241 **2.9.5.1 Cancelability States**

2242 The cancelability state of a thread determines the action taken upon receipt of a cancellation
 2243 request. The thread may control cancellation in a number of ways.

2244 Each thread maintains its own cancelability state, which may be encoded in two bits:

- 2245 1. Cancelability-Enable: When cancelability is `PTHREAD_CANCEL_DISABLE` (as defined in
 2246 the Base Definitions volume of IEEE Std 1003.1-2001, `<pthread.h>`), cancellation requests
 2247 against the target thread are held pending. By default, cancelability is set to
 2248 `PTHREAD_CANCEL_ENABLE` (as defined in `<pthread.h>`).
- 2249 2. Cancelability Type: When cancelability is enabled and the cancelability type is
 2250 `PTHREAD_CANCEL_ASYNCHRONOUS` (as defined in `<pthread.h>`), new or pending
 2251 cancellation requests may be acted upon at any time. When cancelability is enabled and
 2252 the cancelability type is `PTHREAD_CANCEL_DEFERRED` (as defined in `<pthread.h>`),
 2253 cancellation requests are held pending until a cancellation point (see below) is reached. If
 2254 cancelability is disabled, the setting of the cancelability type has no immediate effect as all
 2255 cancellation requests are held pending; however, once cancelability is enabled again the
 2256 new type is in effect. The cancelability type is `PTHREAD_CANCEL_DEFERRED` in all
 2257 newly created threads including the thread in which *main()* was first invoked.

2258 2.9.5.2 Cancellation Points |

2259 Cancellation points shall occur when a thread is executing the following functions: |

2260	<i>accept()</i>	<i>mq_timedsend()</i>	<i>putmsg()</i>	<i>sigpause()</i>
2261	<i>aio_suspend()</i>	<i>msgrcv()</i>	<i>putpmsg()</i>	<i>sigsuspend()</i>
2262	<i>clock_nanosleep()</i>	<i>msgsnd()</i>	<i>pwrite()</i>	<i>sigtimedwait()</i>
2263	<i>close()</i>	<i>msync()</i>	<i>read()</i>	<i>sigwait()</i>
2264	<i>connect()</i>	<i>nanosleep()</i>	<i>readv()</i>	<i>sigwaitinfo()</i>
2265	<i>creat()</i>	<i>open()</i>	<i>recv()</i>	<i>sleep()</i>
2266	<i>fcntl()</i> ²	<i>pause()</i>	<i>recvfrom()</i>	<i>system()</i>
2267	<i>fsync()</i>	<i>poll()</i>	<i>recvmsg()</i>	<i>tcdrain()</i>
2268	<i>getmsg()</i>	<i>pread()</i>	<i>select()</i>	<i>usleep()</i>
2269	<i>getpmsg()</i>	<i>pselect()</i>	<i>sem_timedwait()</i>	<i>wait()</i>
2270	<i>lockf()</i>	<i>pthread_cond_timedwait()</i>	<i>sem_wait()</i>	<i>waitid()</i>
2271	<i>mq_receive()</i>	<i>pthread_cond_wait()</i>	<i>send()</i>	<i>waitpid()</i>
2272	<i>mq_send()</i>	<i>pthread_join()</i>	<i>sendmsg()</i>	<i>write()</i>
2273	<i>mq_timedreceive()</i>	<i>pthread_testcancel()</i>	<i>sendto()</i>	<i>writew()</i>

2274 _____

2275 2. When the *cmd* argument is F_SETLKW.

2276 A cancellation point may also occur when a thread is executing the following functions: |

2277	<i>catclose()</i>	<i>ftell()</i>	<i>getwc()</i>	<i>pthread_rwlock_wrlock()</i>
2278	<i>catgets()</i>	<i>ftello()</i>	<i>getwchar()</i>	<i>putc()</i>
2279	<i>catopen()</i>	<i>ftw()</i>	<i>getwd()</i>	<i>putc_unlocked()</i>
2280	<i>closedir()</i>	<i>fwprintf()</i>	<i>glob()</i>	<i>putchar()</i>
2281	<i>closelog()</i>	<i>fwrite()</i>	<i>iconv_close()</i>	<i>putchar_unlocked()</i>
2282	<i>ctermid()</i>	<i>fwscanf()</i>	<i>iconv_open()</i>	<i>puts()</i>
2283	<i>dbm_close()</i>	<i>getc()</i>	<i>ioctl()</i>	<i>pututxline()</i>
2284	<i>dbm_delete()</i>	<i>getc_unlocked()</i>	<i>lseek()</i>	<i>putwc()</i>
2285	<i>dbm_fetch()</i>	<i>getchar()</i>	<i>mkstemp()</i>	<i>putwchar()</i>
2286	<i>dbm_nextkey()</i>	<i>getchar_unlocked()</i>	<i>nftw()</i>	<i>readdir()</i>
2287	<i>dbm_open()</i>	<i>getcwd()</i>	<i>opendir()</i>	<i>readdir_r()</i>
2288	<i>dbm_store()</i>	<i>getdate()</i>	<i>openlog()</i>	<i>remove()</i>
2289	<i>dlclose()</i>	<i>getgrent()</i>	<i>pclose()</i>	<i>rename()</i>
2290	<i>dlopen()</i>	<i>getgrgid()</i>	<i>perror()</i>	<i>rewind()</i>
2291	<i>endgrent()</i>	<i>getgrgid_r()</i>	<i>popen()</i>	<i>rewinddir()</i>
2292	<i>endhostent()</i>	<i>getgrnam()</i>	<i>posix_fadvise()</i>	<i>scanf()</i>
2293	<i>endnetent()</i>	<i>getgrnam_r()</i>	<i>posix_fallocate()</i>	<i>seekdir()</i>
2294	<i>endprotoent()</i>	<i>gethostbyaddr()</i>	<i>posix_madvise()</i>	<i>semop()</i>
2295	<i>endpwent()</i>	<i>gethostbyname()</i>	<i>posix_spawn()</i>	<i>setgrent()</i>
2296	<i>endservent()</i>	<i>gethostent()</i>	<i>posix_spawnnp()</i>	<i>sethostent()</i>
2297	<i>endutxent()</i>	<i>gethostname()</i>	<i>posix_trace_clear()</i>	<i>setnetent()</i>
2298	<i>fclose()</i>	<i>getlogin()</i>	<i>posix_trace_close()</i>	<i>setprotoent()</i>
2299	<i>fcntl()</i> ³	<i>getlogin_r()</i>	<i>posix_trace_create()</i>	<i>setpwent()</i>
2300	<i>fflush()</i>	<i>getnetbyaddr()</i>	<i>posix_trace_create_withlog()</i>	<i>setservent()</i>
2301	<i>fgetc()</i>	<i>getnetbyname()</i>	<i>posix_trace_eventtypelist_getnext_id()</i>	<i>setutxent()</i>
2302	<i>fgetpos()</i>	<i>getnetent()</i>	<i>posix_trace_eventtypelist_rewind()</i>	<i>strerror()</i>
2303	<i>fgets()</i>	<i>getprotobyname()</i>	<i>posix_trace_flush()</i>	<i>syslog()</i>
2304	<i>fgetwc()</i>	<i>getprotobynumber()</i>	<i>posix_trace_get_attr()</i>	<i>tmpfile()</i>
2305	<i>fgetws()</i>	<i>getprotoent()</i>	<i>posix_trace_get_filter()</i>	<i>tmpnam()</i>
2306	<i>fopen()</i>	<i>getpwent()</i>	<i>posix_trace_get_status()</i>	<i>ttyname()</i>
2307	<i>fprintf()</i>	<i>getpwnam()</i>	<i>posix_trace_getnext_event()</i>	<i>ttyname_r()</i>
2308	<i>fputc()</i>	<i>getpwnam_r()</i>	<i>posix_trace_open()</i>	<i>ungetc()</i>
2309	<i>fputs()</i>	<i>getpwuid()</i>	<i>posix_trace_rewind()</i>	<i>ungetwc()</i>
2310	<i>fputwc()</i>	<i>getpwuid_r()</i>	<i>posix_trace_set_filter()</i>	<i>unlink()</i>
2311	<i>fputws()</i>	<i>gets()</i>	<i>posix_trace_shutdown()</i>	<i>vfprintf()</i>
2312	<i>fread()</i>	<i>getservbyname()</i>	<i>posix_trace_timedgetnext_event()</i>	<i>vwprintf()</i>
2313	<i>freopen()</i>	<i>getservbyport()</i>	<i>posix_typed_mem_open()</i>	<i>vprintf()</i>
2314	<i>fscanf()</i>	<i>getservent()</i>	<i>printf()</i>	<i>vwprintf()</i>
2315	<i>fseek()</i>	<i>getutxent()</i>	<i>pthread_rwlock_rdlock()</i>	<i>wprintf()</i>
2316	<i>fseeko()</i>	<i>getutxid()</i>	<i>pthread_rwlock_timedrdlock()</i>	<i>wscanf()</i>
2317	<i>fsetpos()</i>	<i>getutxline()</i>	<i>pthread_rwlock_timedwrlock()</i>	

2318 An implementation shall not introduce cancellation points into any other functions specified in |
 2319 this volume of IEEE Std 1003.1-2001.

2320 _____

2321 3. For any value of the *cmd* argument.

2322 The side effects of acting upon a cancellation request while suspended during a call of a function
2323 are the same as the side effects that may be seen in a single-threaded program when a call to a
2324 function is interrupted by a signal and the given function returns [EINTR]. Any such side effects
2325 occur before any cancellation cleanup handlers are called.

2326 Whenever a thread has cancelability enabled and a cancellation request has been made with that
2327 thread as the target, and the thread then calls any function that is a cancellation point (such as
2328 *pthread_testcancel()* or *read()*), the cancellation request shall be acted upon before the function
2329 returns. If a thread has cancelability enabled and a cancellation request is made with the thread
2330 as a target while the thread is suspended at a cancellation point, the thread shall be awakened
2331 and the cancellation request shall be acted upon. However, if the thread is suspended at a
2332 cancellation point and the event for which it is waiting occurs before the cancellation request is
2333 acted upon, it is unspecified whether the cancellation request is acted upon or whether the
2334 cancellation request remains pending and the thread resumes normal execution.

2335 2.9.5.3 Thread Cancellation Cleanup Handlers

2336 Each thread maintains a list of cancellation cleanup handlers. The programmer uses the
2337 *pthread_cleanup_push()* and *pthread_cleanup_pop()* functions to place routines on and remove
2338 routines from this list.

2339 When a cancellation request is acted upon, the routines in the list are invoked one by one in
2340 LIFO sequence; that is, the last routine pushed onto the list (Last In) is the first to be invoked
2341 (First Out). The thread invokes the cancellation cleanup handler with cancellation disabled until
2342 the last cancellation cleanup handler returns. When the cancellation cleanup handler for a scope
2343 is invoked, the storage for that scope remains valid. If the last cancellation cleanup handler
2344 returns, thread execution is terminated and a status of PTHREAD_CANCELED is made
2345 available to any threads joining with the target. The symbolic constant PTHREAD_CANCELED
2346 expands to a constant expression of type (**void ***) whose value matches no pointer to an object in
2347 memory nor the value NULL.

2348 The cancellation cleanup handlers are also invoked when the thread calls *pthread_exit()*.

2349 A side effect of acting upon a cancellation request while in a condition variable wait is that the
2350 mutex is re-acquired before calling the first cancellation cleanup handler. In addition, the thread
2351 is no longer considered to be waiting for the condition and the thread shall not have consumed
2352 any pending condition signals on the condition.

2353 A cancellation cleanup handler cannot exit via *longjmp()* or *siglongjmp()*.

2354 2.9.5.4 Async-Cancel Safety

2355 The *pthread_cancel()*, *pthread_setcancelstate()*, and *pthread_setcanceltype()* functions are defined to
2356 be async-cancel safe.

2357 No other functions in this volume of IEEE Std 1003.1-2001 are required to be async-cancel-safe.

2358 **2.9.6 Thread Read-Write Locks**

2359 Multiple readers, single writer (read-write) locks allow many threads to have simultaneous
2360 read-only access to data while allowing only one thread to have exclusive write access at any
2361 given time. They are typically used to protect data that is read more frequently than it is
2362 changed.

2363 One or more readers acquire read access to the resource by performing a read lock operation on
2364 the associated read-write lock. A writer acquires exclusive write access by performing a write
2365 lock operation. Basically, all readers exclude any writers and a writer excludes all readers and
2366 any other writers.

2367 A thread that has blocked on a read-write lock (for example, has not yet returned from a
2368 *pthread_rwlock_rdlock()* or *pthread_rwlock_wrlock()* call) shall not prevent any unblocked thread
2369 that is eligible to use the same processing resources from eventually making forward progress in
2370 its execution. Eligibility for processing resources shall be determined by the scheduling policy.

2371 Read-write locks can be used to synchronize threads in the current process and other processes if
2372 they are allocated in memory that is writable and shared among the cooperating processes and
2373 have been initialized for this behavior.

2374 **2.9.7 Thread Interactions with Regular File Operations**

2375 All of the functions *chmod()*, *close()*, *fchmod()*, *fcntl()*, *fstat()*, *ftruncate()*, *lseek()*, *open()*, *read()*,
2376 *readlink()*, *stat()*, *symlink()*, and *write()* shall be atomic with respect to each other in the effects
2377 specified in IEEE Std 1003.1-2001 when they operate on regular files. If two threads each call one
2378 of these functions, each call shall either see all of the specified effects of the other call, or none of
2379 them.

2380 **2.10 Sockets**

2381 A socket is an endpoint for communication using the facilities described in this section. A socket
2382 is created with a specific socket type, described in Section 2.10.6 (on page 59), and is associated
2383 with a specific protocol, detailed in Section 2.10.3 (on page 59). A socket is accessed via a file
2384 descriptor obtained when the socket is created.

2385 **2.10.1 Address Families**

2386 All network protocols are associated with a specific address family. An address family provides
2387 basic services to the protocol implementation to allow it to function within a specific network
2388 environment. These services may include packet fragmentation and reassembly, routing,
2389 addressing, and basic transport. An address family is normally comprised of a number of
2390 protocols, one per socket type. Each protocol is characterized by an abstract socket type. It is not
2391 required that an address family support all socket types. An address family may contain
2392 multiple protocols supporting the same socket abstraction.

2393 Section 2.10.17 (on page 66), Section 2.10.19 (on page 67), and Section 2.10.20 (on page 67),
2394 respectively, describe the use of sockets for local UNIX connections, for Internet protocols based
2395 on IPv4, and for Internet protocols based on IPv6.

2396 **2.10.2 Addressing**

2397 An address family defines the format of a socket address. All network addresses are described
 2398 using a general structure, called a **sockaddr**, as defined in the Base Definitions volume of
 2399 IEEE Std 1003.1-2001, <**sys/socket.h**>. However, each address family imposes finer and more
 2400 specific structure, generally defining a structure with fields specific to the address family. The
 2401 field *sa_family* in the **sockaddr** structure contains the address family identifier, specifying the
 2402 format of the *sa_data* area. The size of the *sa_data* area is unspecified.

2403 **2.10.3 Protocols**

2404 A protocol supports one of the socket abstractions detailed in Section 2.10.6. Selecting a protocol
 2405 involves specifying the address family, socket type, and protocol number to the *socket()*
 2406 function. Certain semantics of the basic socket abstractions are protocol-specific. All protocols
 2407 are expected to support the basic model for their particular socket type, but may, in addition,
 2408 provide non-standard facilities or extensions to a mechanism.

2409 **2.10.4 Routing**

2410 Sockets provides packet routing facilities. A routing information database is maintained, which
 2411 is used in selecting the appropriate network interface when transmitting packets.

2412 **2.10.5 Interfaces**

2413 Each network interface in a system corresponds to a path through which messages can be sent
 2414 and received. A network interface usually has a hardware device associated with it, though
 2415 certain interfaces such as the loopback interface, do not.

2416 **2.10.6 Socket Types**

2417 A socket is created with a specific type, which defines the communication semantics and which
 2418 RS allows the selection of an appropriate communication protocol. Four types are defined:
 2419 **SOCK_RAW**, **SOCK_STREAM**, **SOCK_SEQPACKET**, and **SOCK_DGRAM**. Implementations
 2420 may specify additional socket types.

2421 The **SOCK_STREAM** socket type provides reliable, sequenced, full-duplex octet streams
 2422 between the socket and a peer to which the socket is connected. A socket of type
 2423 **SOCK_STREAM** must be in a connected state before any data may be sent or received. Record
 2424 boundaries are not maintained; data sent on a stream socket using output operations of one size
 2425 may be received using input operations of smaller or larger sizes without loss of data. Data may
 2426 be buffered; successful return from an output function does not imply that the data has been
 2427 delivered to the peer or even transmitted from the local system. If data cannot be successfully
 2428 transmitted within a given time then the connection is considered broken, and subsequent
 2429 operations shall fail. A SIGPIPE signal is raised if a thread sends on a broken stream (one that is
 2430 no longer connected). Support for an out-of-band data transmission facility is protocol-specific.

2431 The **SOCK_SEQPACKET** socket type is similar to the **SOCK_STREAM** type, and is also
 2432 connection-oriented. The only difference between these types is that record boundaries are
 2433 maintained using the **SOCK_SEQPACKET** type. A record can be sent using one or more output
 2434 operations and received using one or more input operations, but a single operation never
 2435 transfers parts of more than one record. Record boundaries are visible to the receiver via the
 2436 **MSG_EOR** flag in the received message flags returned by the *recvmsg()* function. It is protocol-
 2437 specific whether a maximum record size is imposed.

2438 The **SOCK_DGRAM** socket type supports connectionless data transfer which is not necessarily
 2439 acknowledged or reliable. Datagrams may be sent to the address specified (possibly multicast or

2440 broadcast) in each output operation, and incoming datagrams may be received from multiple
2441 sources. The source address of each datagram is available when receiving the datagram. An
2442 application may also pre-specify a peer address, in which case calls to output functions shall
2443 send to the pre-specified peer. If a peer has been specified, only datagrams from that peer shall
2444 be received. A datagram must be sent in a single output operation, and must be received in a
2445 single input operation. The maximum size of a datagram is protocol-specific; with some
2446 protocols, the limit is implementation-defined. Output datagrams may be buffered within the
2447 system; thus, a successful return from an output function does not guarantee that a datagram is
2448 actually sent or received. However, implementations should attempt to detect any errors
2449 possible before the return of an output function, reporting any error by an unsuccessful return
2450 value.

2451 RS The SOCK_RAW socket type is similar to the SOCK_DGRAM type. It differs in that it is
2452 normally used with communication providers that underlie those used for the other socket
2453 types. For this reason, the creation of a socket with type SOCK_RAW shall require appropriate
2454 privilege. The format of datagrams sent and received with this socket type generally include
2455 specific protocol headers, and the formats are protocol-specific and implementation-defined.

2456 **2.10.7 Socket I/O Mode**

2457 The I/O mode of a socket is described by the O_NONBLOCK file status flag which pertains to
2458 the open file description for the socket. This flag is initially off when a socket is created, but may
2459 be set and cleared by the use of the F_SETFL command of the *fcntl()* function.

2460 When the O_NONBLOCK flag is set, functions that would normally block until they are
2461 complete shall either return immediately with an error, or shall complete asynchronously to the
2462 execution of the calling process. Data transfer operations (the *read()*, *write()*, *send()*, and *recv()*
2463 functions) shall complete immediately, transfer only as much as is available, and then return
2464 without blocking, or return an error indicating that no transfer could be made without blocking.
2465 The *connect()* function initiates a connection and shall return without blocking when
2466 O_NONBLOCK is set; it shall return the error [EINPROGRESS] to indicate that the connection
2467 was initiated successfully, but that it has not yet completed.

2468 **2.10.8 Socket Owner**

2469 The owner of a socket is unset when a socket is created. The owner may be set to a process ID or
2470 process group ID using the F_SETOWN command of the *fcntl()* function.

2471 **2.10.9 Socket Queue Limits**

2472 The transmit and receive queue sizes for a socket are set when the socket is created. The default
2473 sizes used are both protocol-specific and implementation-defined. The sizes may be changed
2474 using the *setsockopt()* function.

2475 **2.10.10 Pending Error**

2476 Errors may occur asynchronously, and be reported to the socket in response to input from the
2477 network protocol. The socket stores the pending error to be reported to a user of the socket at the
2478 next opportunity. The error is returned in response to a subsequent *send()*, *recv()*, or *getsockopt()*
2479 operation on the socket, and the pending error is then cleared.

2.10.11 Socket Receive Queue

2481 A socket has a receive queue that buffers data when it is received by the system until it is
2482 removed by a receive call. Depending on the type of the socket and the communication provider,
2483 the receive queue may also contain ancillary data such as the addressing and other protocol data
2484 associated with the normal data in the queue, and may contain out-of-band or expedited data.
2485 The limit on the queue size includes any normal, out-of-band data, datagram source addresses,
2486 and ancillary data in the queue. The description in this section applies to all sockets, even though
2487 some elements cannot be present in some instances.

2488 The contents of a receive buffer are logically structured as a series of data segments with
2489 associated ancillary data and other information. A data segment may contain normal data or
2490 out-of-band data, but never both. A data segment may complete a record if the protocol
2491 supports records (always true for types SOCK_SEQPACKET and SOCK_DGRAM). A record
2492 may be stored as more than one segment; the complete record might never be present in the
2493 receive buffer at one time, as a portion might already have been returned to the application, and
2494 another portion might not yet have been received from the communications provider. A data
2495 segment may contain ancillary protocol data, which is logically associated with the segment.
2496 Ancillary data is received as if it were queued along with the first normal data octet in the
2497 segment (if any). A segment may contain ancillary data only, with no normal or out-of-band
2498 data. For the purposes of this section, a datagram is considered to be a data segment that
2499 terminates a record, and that includes a source address as a special type of ancillary data. Data
2500 segments are placed into the queue as data is delivered to the socket by the protocol. Normal
2501 data segments are placed at the end of the queue as they are delivered. If a new segment
2502 contains the same type of data as the preceding segment and includes no ancillary data, and if
2503 the preceding segment does not terminate a record, the segments are logically merged into a
2504 single segment.

2505 The receive queue is logically terminated if an end-of-file indication has been received or a
2506 connection has been terminated. A segment shall be considered to be terminated if another
2507 segment follows it in the queue, if the segment completes a record, or if an end-of-file or other
2508 connection termination has been reported. The last segment in the receive queue shall also be
2509 considered to be terminated while the socket has a pending error to be reported.

2510 A receive operation shall never return data or ancillary data from more than one segment.

2.10.12 Socket Out-of-Band Data State

2512 The handling of received out-of-band data is protocol-specific. Out-of-band data may be placed
2513 in the socket receive queue, either at the end of the queue or before all normal data in the queue.
2514 In this case, out-of-band data is returned to an application program by a normal receive call.
2515 Out-of-band data may also be queued separately rather than being placed in the socket receive
2516 queue, in which case it shall be returned only in response to a receive call that requests out-of-
2517 band data. It is protocol-specific whether an out-of-band data mark is placed in the receive
2518 queue to demarcate data preceding the out-of-band data and following the out-of-band data. An
2519 out-of-band data mark is logically an empty data segment that cannot be merged with other
2520 segments in the queue. An out-of-band data mark is never returned in response to an input
2521 operation. The *socketmark()* function can be used to test whether an out-of-band data mark is the
2522 first element in the queue. If an out-of-band data mark is the first element in the queue when an
2523 input function is called without the MSG_PEEK option, the mark is removed from the queue and
2524 the following data (if any) is processed as if the mark had not been present.

2525 2.10.13 Connection Indication Queue

2526 Sockets that are used to accept incoming connections maintain a queue of outstanding
2527 connection indications. This queue is a list of connections that are awaiting acceptance by the
2528 application; see *listen()*.

2529 2.10.14 Signals

2530 One category of event at the socket interface is the generation of signals. These signals report
2531 protocol events or process errors relating to the state of the socket. The generation or delivery of
2532 a signal does not change the state of the socket, although the generation of the signal may have
2533 been caused by a state change.

2534 The SIGPIPE signal shall be sent to a thread that attempts to send data on a socket that is no
2535 longer able to send. In addition, the send operation fails with the error [EPIPE].

2536 If a socket has an owner, the SIGURG signal is sent to the owner of the socket when it is notified
2537 of expedited or out-of-band data. The socket state at this time is protocol-dependent, and the
2538 status of the socket is specified in Section 2.10.17 (on page 66), Section 2.10.19 (on page 67), and
2539 Section 2.10.20 (on page 67). Depending on the protocol, the expedited data may or may not
2540 have arrived at the time of signal generation.

2541 2.10.15 Asynchronous Errors

2542 If any of the following conditions occur asynchronously for a socket, the corresponding value
2543 listed below shall become the pending error for the socket:

2544 [ECONNABORTED]

2545 The connection was aborted locally.

2546 [ECONNREFUSED]

2547 For a connection-mode socket attempting a non-blocking connection, the attempt to connect
2548 was forcefully rejected. For a connectionless-mode socket, an attempt to deliver a datagram
2549 was forcefully rejected.

2550 [ECONNRESET]

2551 The peer has aborted the connection.

2552 [EHOSTDOWN]

2553 The destination host has been determined to be down or disconnected.

2554 [EHOSTUNREACH]

2555 The destination host is not reachable.

2556 [EMSGSIZE]

2557 For a connectionless-mode socket, the size of a previously sent datagram prevented
2558 delivery.

2559 [ENETDOWN]

2560 The local network connection is not operational.

2561 [ENETRESET]

2562 The connection was aborted by the network.

2563 [ENETUNREACH]

2564 The destination network is not reachable.

2565 **2.10.16 Use of Options**

2566 There are a number of socket options which either specialize the behavior of a socket or provide
 2567 useful information. These options may be set at different protocol levels and are always present
 2568 at the uppermost “socket” level.

2569 Socket options are manipulated by two functions, *getsockopt()* and *setsockopt()*. These functions
 2570 allow an application program to customize the behavior and characteristics of a socket to
 2571 provide the desired effect.

2572 All of the options have default values. The type and meaning of these values is defined by the
 2573 protocol level to which they apply. Instead of using the default values, an application program
 2574 may choose to customize one or more of the options. However, in the bulk of cases, the default
 2575 values are sufficient for the application.

2576 Some of the options are used to enable or disable certain behavior within the protocol modules
 2577 (for example, turn on debugging) while others may be used to set protocol-specific information
 2578 (for example, IP time-to-live on all the application’s outgoing packets). As each of the options is
 2579 introduced, its effect on the underlying protocol modules is described.

2580 Table 2-1 shows the value for the socket level.

2581 **Table 2-1** Value of Level for Socket Options

Name	Description
SOL_SOCKET	Options are intended for the sockets level.

2584 Table 2-2 (on page 64) lists those options present at the socket level; that is, when the *level*
 2585 parameter of the *getsockopt()* or *setsockopt()* function is SOL_SOCKET, the types of the option
 2586 value parameters associated with each option, and a brief synopsis of the meaning of the option
 2587 value parameter. Unless otherwise noted, each may be examined with *getsockopt()* and set with
 2588 *setsockopt()* on all types of socket.

Table 2-2 Socket-Level Options

2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618

Option	Parameter Type	Parameter Meaning
SO_BROADCAST	int	Non-zero requests permission to transmit broadcast datagrams (SOCK_DGRAM sockets only).
SO_DEBUG	int	Non-zero requests debugging in underlying protocol modules.
SO_DONTROUTE	int	Non-zero requests bypass of normal routing; route based on destination address only.
SO_ERROR	int	Requests and clears pending error information on the socket (<i>getsockopt()</i> only).
SO_KEEPALIVE	int	Non-zero requests periodic transmission of keepalive messages (protocol-specific).
SO_LINGER	struct linger	Specify actions to be taken for queued, unsent data on <i>close()</i> : linger on/off and linger time in seconds.
SO_OOBINLINE	int	Non-zero requests that out-of-band data be placed into normal data input queue as received.
SO_RCVBUF	int	Size of receive buffer (in bytes).
SO_RCVLOWAT	int	Minimum amount of data to return to application for input operations (in bytes).
SO_RCVTIMEO	struct timeval	Timeout value for a socket receive operation.
SO_REUSEADDR	int	Non-zero requests reuse of local addresses in <i>bind()</i> (protocol-specific).
SO_SNDBUF	int	Size of send buffer (in bytes).
SO_SNDLOWAT	int	Minimum amount of data to send for output operations (in bytes).
SO_SNDTIMEO	struct timeval	Timeout value for a socket send operation.
SO_TYPE	int	Identify socket type (<i>getsockopt()</i> only).

2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635

The SO_BROADCAST option requests permission to send broadcast datagrams on the socket. Support for SO_BROADCAST is protocol-specific. The default for SO_BROADCAST is that the ability to send broadcast datagrams on a socket is disabled.

The SO_DEBUG option enables debugging in the underlying protocol modules. This can be useful for tracing the behavior of the underlying protocol modules during normal system operation. The semantics of the debug reports are implementation-defined. The default value for SO_DEBUG is for debugging to be turned off.

The SO_DONTROUTE option requests that outgoing messages bypass the standard routing facilities. The destination must be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. It is protocol-specific whether this option has any effect and how the outgoing network interface is chosen. Support for this option with each protocol is implementation-defined.

The SO_ERROR option is used only on *getsockopt()*. When this option is specified, *getsockopt()* shall return any pending error on the socket and clear the error status. It shall return a value of 0 if there is no pending error. SO_ERROR may be used to check for asynchronous errors on connected connectionless-mode sockets or for other types of asynchronous errors. SO_ERROR has no default value.

2636 The SO_KEEPALIVE option enables the periodic transmission of messages on a connected
2637 socket. The behavior of this option is protocol-specific. The default value for SO_KEEPALIVE is
2638 zero, specifying that this capability is turned off.

2639 The SO_LINGER option controls the action of the interface when unsent messages are queued
2640 on a socket and a *close()* is performed. The details of this option are protocol-specific. The
2641 default value for SO_LINGER is zero, or off, for the *L_onoff* element of the option value and zero
2642 seconds for the linger time specified by the *L_linger* element.

2643 The SO_OOBINLINE option is valid only on protocols that support out-of-band data. The
2644 SO_OOBINLINE option requests that out-of-band data be placed in the normal data input queue
2645 as received; it is then accessible using the *read()* or *recv()* functions without the MSG_OOB flag
2646 set. The default for SO_OOBINLINE is off; that is, for out-of-band data not to be placed in the
2647 normal data input queue.

2648 The SO_RCVBUF option requests that the buffer space allocated for receive operations on this
2649 socket be set to the value, in bytes, of the option value. Applications may wish to increase buffer
2650 size for high volume connections, or may decrease buffer size to limit the possible backlog of
2651 incoming data. The default value for the SO_RCVBUF option value is implementation-defined,
2652 and may vary by protocol.

2653 The maximum value for the option for a socket may be obtained by the use of the *fpathconf()*
2654 function, using the value *_PC_SOCKET_MAXBUF*.

2655 The SO_RCVLOWAT option sets the minimum number of bytes to process for socket input
2656 operations. In general, receive calls block until any (non-zero) amount of data is received, then
2657 return the smaller of the amount available or the amount requested. The default value for
2658 SO_RCVLOWAT is 1, and does not affect the general case. If SO_RCVLOWAT is set to a larger
2659 value, blocking receive calls normally wait until they have received the smaller of the low water
2660 mark value or the requested amount. Receive calls may still return less than the low water mark
2661 if an error occurs, a signal is caught, or the type of data next in the receive queue is different
2662 from that returned (for example, out-of-band data). As mentioned previously, the default value
2663 for SO_RCVLOWAT is 1 byte. It is implementation-defined whether the SO_RCVLOWAT option
2664 can be set.

2665 The SO_RCVTIMEO option is an option to set a timeout value for input operations. It accepts a
2666 **timeval** structure with the number of seconds and microseconds specifying the limit on how
2667 long to wait for an input operation to complete. If a receive operation has blocked for this much
2668 time without receiving additional data, it shall return with a partial count or *errno* shall be set to
2669 [EWOULDBLOCK] if no data were received. The default for this option is the value zero, which
2670 indicates that a receive operation will not time out. It is implementation-defined whether the
2671 SO_RCVTIMEO option can be set.

2672 The SO_REUSEADDR option indicates that the rules used in validating addresses supplied in a
2673 *bind()* should allow reuse of local addresses. Operation of this option is protocol-specific. The
2674 default value for SO_REUSEADDR is off; that is, reuse of local addresses is not permitted.

2675 The SO_SNDBUF option requests that the buffer space allocated for send operations on this
2676 socket be set to the value, in bytes, of the option value. The default value for the SO_SNDBUF
2677 option value is implementation-defined, and may vary by protocol. The maximum value for the
2678 option for a socket may be obtained by the use of the *fpathconf()* function, using the value
2679 *_PC_SOCKET_MAXBUF*.

2680 The SO_SNDLOWAT option sets the minimum number of bytes to process for socket output
2681 operations. Most output operations process all of the data supplied by the call, delivering data to
2682 the protocol for transmission and blocking as necessary for flow control. Non-blocking output
2683 operations process as much data as permitted subject to flow control without blocking, but

2684 process no data if flow control does not allow the smaller of the send low water mark value or
 2685 the entire request to be processed. A *select()* operation testing the ability to write to a socket shall
 2686 return true only if the send low water mark could be processed. The default value for
 2687 SO_SNDLOWAT is implementation-defined and protocol-specific. It is implementation-defined
 2688 whether the SO_SNDLOWAT option can be set.

2689 The SO_SNDTIMEO option is an option to set a timeout value for the amount of time that an
 2690 output function shall block because flow control prevents data from being sent. As noted in
 2691 Table 2-2 (on page 64), the option value is a **timeval** structure with the number of seconds and
 2692 microseconds specifying the limit on how long to wait for an output operation to complete. If a
 2693 send operation has blocked for this much time, it shall return with a partial count or *errno* set to
 2694 [EWOULDBLOCK] if no data were sent. The default for this option is the value zero, which
 2695 indicates that a send operation will not time out. It is implementation-defined whether the
 2696 SO_SNDTIMEO option can be set.

2697 The SO_TYPE option is used only on *getsockopt()*. When this option is specified, *getsockopt()*
 2698 shall return the type of the socket (for example, SOCK_STREAM). This option is useful to
 2699 servers that inherit sockets on start-up. SO_TYPE has no default value.

2700 2.10.17 Use of Sockets for Local UNIX Connections

2701 Support for UNIX domain sockets is mandatory.

2702 UNIX domain sockets provide process-to-process communication in a single system.

2703 2.10.17.1 Headers

2704 The symbolic constant AF_UNIX defined in the `<sys/socket.h>` header is used to identify the
 2705 UNIX domain address family. The `<sys/un.h>` header contains other definitions used in
 2706 connection with UNIX domain sockets. See the Base Definitions volume of IEEE Std 1003.1-2001,
 2707 Chapter 13, Headers.

2708 The **sockaddr_storage** structure defined in `<sys/socket.h>` shall be large enough to
 2709 accommodate a **sockaddr_un** structure (see the `<sys/un.h>` header defined in the Base
 2710 Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers) and shall be aligned at an
 2711 appropriate boundary so that pointers to it can be cast as pointers to **sockaddr_un** structures
 2712 and used to access the fields of those structures without alignment problems. When a
 2713 **sockaddr_storage** structure is cast as a **sockaddr_un** structure, the *ss_family* field maps onto the
 2714 *sun_family* field.

2715 2.10.18 Use of Sockets over Internet Protocols

2716 When a socket is created in the Internet family with a protocol value of zero, the implementation
 2717 shall use the protocol listed below for the type of socket created.

2718 SOCK_STREAM IPPROTO_TCP.

2719 SOCK_DGRAM IPPROTO_UDP.

2720 RS SOCK_RAW IPPROTO_RAW.

2721 SOCK_SEQPACKET Unspecified.

2722 RS A raw interface to IP is available by creating an Internet socket of type SOCK_RAW. The default
 2723 protocol for type SOCK_RAW shall be identified in the IP header with the value
 2724 IPPROTO_RAW. Applications should not use the default protocol when creating a socket with
 2725 type SOCK_RAW, but should identify a specific protocol by value. The ICMP control protocol is
 2726 accessible from a raw socket by specifying a value of IPPROTO_ICMP for protocol.

2727 **2.10.19 Use of Sockets over Internet Protocols Based on IPv4**

2728 Support for sockets over Internet protocols based on IPv4 is mandatory.

2729 **2.10.19.1 Headers**

2730 The symbolic constant `AF_INET` defined in the `<sys/socket.h>` header is used to identify the
 2731 IPv4 Internet address family. The `<netinet/in.h>` header contains other definitions used in
 2732 connection with IPv4 Internet sockets. See the Base Definitions volume of IEEE Std 1003.1-2001,
 2733 Chapter 13, Headers.

2734 The `sockaddr_storage` structure defined in `<sys/socket.h>` shall be large enough to
 2735 accommodate a `sockaddr_in` structure (see the `<netinet/in.h>` header defined in the Base
 2736 Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers) and shall be aligned at an
 2737 appropriate boundary so that pointers to it can be cast as pointers to `sockaddr_in` structures and
 2738 used to access the fields of those structures without alignment problems. When a
 2739 `sockaddr_storage` structure is cast as a `sockaddr_in` structure, the `ss_family` field maps onto the
 2740 `sin_family` field.

2741 **2.10.20 Use of Sockets over Internet Protocols Based on IPv6**

2742 IP6 This section describes extensions to support sockets over Internet protocols based on IPv6. This
 2743 functionality is dependent on support of the IPV6 option (and the rest of this section is not
 2744 further shaded for this option).

2745 To enable smooth transition from IPv4 to IPv6, the features defined in this section may, in certain
 2746 circumstances, also be used in connection with IPv4; see Section 2.10.20.2 (on page 68).

2747 **2.10.20.1 Addressing**

2748 IPv6 overcomes the addressing limitations of previous versions by using 128-bit addresses
 2749 instead of 32-bit addresses. The IPv6 address architecture is described in RFC 2373.

2750 There are three kinds of IPv6 address:

2751 Unicast

2752 Identifies a single interface.

2753 A unicast address can be global, link-local (designed for use on a single link), or site-local
 2754 (designed for systems not connected to the Internet). Link-local and site-local addresses
 2755 need not be globally unique.

2756 Anycast

2757 Identifies a set of interfaces such that a packet sent to the address can be delivered to any
 2758 member of the set.

2759 An anycast address is similar to a unicast address; the nodes to which an anycast address is
 2760 assigned must be explicitly configured to know that it is an anycast address.

2761 Multicast

2762 Identifies a set of interfaces such that a packet sent to the address should be delivered to
 2763 every member of the set.

2764 An application can send multicast datagrams by simply specifying an IPv6 multicast
 2765 address in the `address` argument of `sendto()`. To receive multicast datagrams, an application
 2766 must join the multicast group (using `setsockopt()` with `IPV6_JOIN_GROUP`) and must bind
 2767 to the socket the UDP port on which datagrams will be received. Some applications should
 2768 also bind the multicast group address to the socket, to prevent other datagrams destined to
 2769 that port from being delivered to the socket.

2770 A multicast address can be global, node-local, link-local, site-local, or organization-local.
2771 The following special IPv6 addresses are defined:
2772 Unspecified
2773 An address that is not assigned to any interface and is used to indicate the absence of an
2774 address.
2775 Loopback
2776 A unicast address that is not assigned to any interface and can be used by a node to send
2777 packets to itself.
2778 Two sets of IPv6 addresses are defined to correspond to IPv4 addresses:
2779 IPv4-compatible addresses
2780 These are assigned to nodes that support IPv6 and can be used when traffic is “tunneled”
2781 through IPv4.
2782 IPv4-mapped addresses
2783 These are used to represent IPv4 addresses in IPv6 address format; see Section 2.10.20.2.
2784 Note that the unspecified address and the loopback address must not be treated as IPv4-
2785 compatible addresses.

2786 2.10.20.2 Compatibility with IPv4

2787 The API provides the ability for IPv6 applications to interoperate with applications using IPv4,
2788 by using IPv4-mapped IPv6 addresses. These addresses can be generated automatically by the
2789 *getaddrinfo()* function when the specified host has only IPv4 addresses.

2790 Applications can use AF_INET6 sockets to open TCP connections to IPv4 nodes, or send UDP
2791 packets to IPv4 nodes, by simply encoding the destination's IPv4 address as an IPv4-mapped
2792 IPv6 address, and passing that address, within a **sockaddr_in6** structure, in the *connect()*,
2793 *sendto()*, or *sendmsg()* function. When applications use AF_INET6 sockets to accept TCP
2794 connections from IPv4 nodes, or receive UDP packets from IPv4 nodes, the system shall return
2795 the peer's address to the application in the *accept()*, *recvfrom()*, *recvmsg()*, or *getpeername()*
2796 function using a **sockaddr_in6** structure encoded this way. If a node has an IPv4 address, then
2797 the implementation shall allow applications to communicate using that address via an
2798 AF_INET6 socket. In such a case, the address will be represented at the API by the
2799 corresponding IPv4-mapped IPv6 address. Also, the implementation may allow an AF_INET6
2800 socket bound to **in6addr_any** to receive inbound connections and packets destined to one of the
2801 node's IPv4 addresses.

2802 An application can use AF_INET6 sockets to bind to a node's IPv4 address by specifying the
2803 address as an IPv4-mapped IPv6 address in a **sockaddr_in6** structure in the *bind()* function. For
2804 an AF_INET6 socket bound to a node's IPv4 address, the system shall return the address in the
2805 *getsockname()* function as an IPv4-mapped IPv6 address in a **sockaddr_in6** structure.

2806 2.10.20.3 Interface Identification

2807 Each local interface is assigned a unique positive integer as a numeric index. Indexes start at 1;
2808 zero is not used. There may be gaps so that there is no current interface for a particular positive
2809 index. Each interface also has a unique implementation-defined name.

2810 2.10.20.4 Options

2811 The following options apply at the IPPROTO_IPV6 level:

2812 IPV6_JOIN_GROUP

2813 When set via *setsockopt()*, it joins the application to a multicast group on an interface
 2814 (identified by its index) and addressed by a given multicast address, enabling packets sent
 2815 to that address to be read via the socket. If the interface index is specified as zero, the
 2816 system selects the interface (for example, by looking up the address in a routing table and
 2817 using the resulting interface).

2818 An attempt to read this option using *getsockopt()* shall result in an [EOPNOTSUPP] error.

2819 The parameter type of this option is a pointer to an **ipv6_mreq** structure.

2820 IPV6_LEAVE_GROUP

2821 When set via *setsockopt()*, it removes the application from the multicast group on an
 2822 interface (identified by its index) and addressed by a given multicast address.

2823 An attempt to read this option using *getsockopt()* shall result in an [EOPNOTSUPP] error.

2824 The parameter type of this option is a pointer to an **ipv6_mreq** structure.

2825 IPV6_MULTICAST_HOPS

2826 The value of this option is the hop limit for outgoing multicast IPv6 packets sent via the
 2827 socket. Its possible values are the same as those of IPV6_UNICAST_HOPS. If the
 2828 IPV6_MULTICAST_HOPS option is not set, a value of 1 is assumed. This option can be set
 2829 via *setsockopt()* and read via *getsockopt()*.

2830 The parameter type of this option is a pointer to an **int**. (Default value: 1)

2831 IPV6_MULTICAST_IF

2832 The index of the interface to be used for outgoing multicast packets. It can be set via
 2833 *setsockopt()* and read via *getsockopt()*. If the interface index is specified as zero, the system
 2834 selects the interface (for example, by looking up the address in a routing table and using the
 2835 resulting interface).

2836 The parameter type of this option is a pointer to an **unsigned int**. (Default value: 0)

2837 IPV6_MULTICAST_LOOP

2838 This option controls whether outgoing multicast packets should be delivered back to the
 2839 local application when the sending interface is itself a member of the destination multicast
 2840 group. If it is set to 1 they are delivered. If it is set to 0 they are not. Other values result in an
 2841 [EINVAL] error. This option can be set via *setsockopt()* and read via *getsockopt()*.

2842 The parameter type of this option is a pointer to an **unsigned int** which is used as a Boolean
 2843 value. (Default value: 1)

2844 IPV6_UNICAST_HOPS

2845 The value of this option is the hop limit for outgoing unicast IPv6 packets sent via the
 2846 socket. If the option is not set, or is set to -1, the system selects a default value. Attempts to
 2847 set a value less than -1 or greater than 255 shall result in an [EINVAL] error. This option can
 2848 be set via *setsockopt()* and read via *getsockopt()*.

2849 The parameter type of this option is a pointer to an **int**. (Default value: Unspecified)

2850 IPV6_V6ONLY

2851 This socket option restricts AF_INET6 sockets to IPv6 communications only. AF_INET6
 2852 sockets may be used for both IPv4 and IPv6 communications. Some applications may want
 2853 to restrict their use of an AF_INET6 socket to IPv6 communications only. For these

2854 applications, the IPv6_V6ONLY socket option is defined. When this option is turned on, the
 2855 socket can be used to send and receive IPv6 packets only. This is an IPPROTO_IPV6-level
 2856 option.

2857 The parameter type of this option is a pointer to an **int** which is used as a Boolean value.
 2858 (Default value: 0)

2859 An [EOPNOTSUPP] error shall result if IPV6_JOIN_GROUP or IPV6_LEAVE_GROUP is used
 2860 with *getsockopt()*.

2861 2.10.20.5 Headers

2862 The symbolic constant AF_INET6 is defined in the `<sys/socket.h>` header to identify the IPv6
 2863 Internet address family. See the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 13,
 2864 Headers.

2865 The **sockaddr_storage** structure defined in `<sys/socket.h>` shall be large enough to
 2866 accommodate a **sockaddr_in6** structure (see the `<netinet/in.h>` header defined in the Base
 2867 Definitions volume of IEEE Std 1003.1-2001, Chapter 13, Headers) and shall be aligned at an
 2868 appropriate boundary so that pointers to it can be cast as pointers to **sockaddr_in6** structures
 2869 and used to access the fields of those structures without alignment problems. When a
 2870 **sockaddr_storage** structure is cast as a **sockaddr_in6** structure, the *ss_family* field maps onto the
 2871 *sin6_family* field.

2872 The `<netinet/in.h>`, `<arpa/inet.h>`, and `<netdb.h>` headers contain other definitions used in
 2873 connection with IPv6 Internet sockets; see the Base Definitions volume of IEEE Std 1003.1-2001,
 2874 Chapter 13, Headers.

2875 2.11 Tracing

2876 TRC This section describes extensions to support tracing of user applications. This functionality is
 2877 dependent on support of the Trace option (and the rest of this section is not further shaded for
 2878 this option).

2879 The tracing facilities defined in IEEE Std 1003.1-2001 allow a process to select a set of trace event
 2880 types, to activate a trace stream of the selected trace events as they occur in the flow of
 2881 execution, and to retrieve the recorded trace events.

2882 The tracing operation relies on three logically different components: the traced process, the
 2883 controller process, and the analyzer process. During the execution of the traced process, when a
 2884 trace point is reached, a trace event is recorded into the trace streams created for that process in
 2885 which the associated trace event type identifier is not being filtered out. The controller process
 2886 controls the operation of recording the trace events into the trace stream. It shall be able to:

- 2887 • Initialize the attributes of a trace stream
- 2888 • Create the trace stream (for a specified traced process) using those attributes
- 2889 • Start and stop tracing for the trace stream
- 2890 • Filter the type of trace events to be recorded, if the Trace Event Filter option is supported
- 2891 • Shut a trace stream down

2892 These operations can be done for an active trace stream. The analyzer process retrieves the
 2893 traced events either at runtime, when the trace stream has not yet been shut down, but is still
 2894 recording trace events; or after opening a trace log that had been previously recorded and shut
 2895 down. These three logically different operations can be performed by the same process, or can be

2896 distributed into different processes.

2897 A trace stream identifier can be created by a call to `posix_trace_create()`,
2898 `posix_trace_create_withlog()`, or `posix_trace_open()`. The `posix_trace_create()` and
2899 `posix_trace_create_withlog()` functions should be used by a controller process. The
2900 `posix_trace_open()` should be used by an analyzer process.

2901 The tracing functions can serve different purposes. One purpose is debugging the possibly pre-
2902 instrumented code, while another is post-mortem fault analysis. These two potential uses differ
2903 in that the first requires pre-filtering capabilities to avoid overwhelming the trace stream and
2904 permits focusing on expected information; while the second needs comprehensive trace
2905 capabilities in order to be able to record all types of information.

2906 The events to be traced belong to two classes:

- 2907 1. User trace events (generated by the application instrumentation)
- 2908 2. System trace events (generated by the operating system)

2909 The trace interface defines several system trace event types associated with control of and
2910 operation of the trace stream. This small set of system trace events includes the minimum
2911 required to interpret correctly the trace event information present in the stream. Other desirable
2912 system trace events for some particular application profile may be implemented and are
2913 encouraged; for example, process and thread scheduling, signal occurrence, and so on.

2914 Each traced process shall have a mapping of the trace event names to trace event type identifiers
2915 that have been defined for that process. Each active trace stream shall have a mapping that
2916 incorporates all the trace event type identifiers predefined by the trace system plus all the
2917 mappings of trace event names to trace event type identifiers of the processes that are being
2918 traced into that trace stream. These mappings are defined from the instrumented application by
2919 calling the `posix_trace_eventid_open()` function and from the controller process by calling the
2920 `posix_trace_trid_eventid_open()` function. For a pre-recorded trace stream, the list of trace event
2921 types is obtained from the pre-recorded trace log.

2922 The `st_ctime` and `st_mtime` fields of a file associated with an active trace stream shall be marked
2923 for update every time any of the tracing operations modifies that file.

2924 The `st_atime` field of a file associated with a trace stream shall be marked for update every time
2925 any of the tracing operations causes data to be read from that file.

2926 Results are undefined if the application performs any operation on a file descriptor associated
2927 with an active or pre-recorded trace stream until `posix_trace_shutdown()` or `posix_trace_close()` is
2928 called for that trace stream.

2929 The main purpose of this option is to define a complete set of functions and concepts that allow
2930 a conforming application to be traced from creation to termination, whatever its realtime
2931 constraints and properties.

2932 2.11.1 Tracing Data Definitions

2933 2.11.1.1 Structures

2934 The `<trace.h>` header shall define the `posix_trace_status_info` and `posix_trace_event_info` structures
2935 described below. Implementations may add extensions to these structures.

2936 **posix_trace_status_info Structure**

2937 To facilitate control of a trace stream, information about the current state of an active trace
 2938 stream can be obtained dynamically. This structure is returned by a call to the
 2939 *posix_trace_get_status()* function.

2940 The **posix_trace_status_info** structure defined in `<trace.h>` shall contain at least the following
 2941 members:

Member Type	Member Name	Description
2942 int	<i>posix_stream_status</i>	The operating mode of the trace stream.
2943 int	<i>posix_stream_full_status</i>	The full status of the trace stream.
2944 int	<i>posix_stream_overrun_status</i>	Indicates whether trace events were 2945 lost in the trace stream.

2948 If the Trace Log option is supported in addition to the Trace option, the **posix_trace_status_info**
 2949 structure defined in `<trace.h>` shall contain at least the following additional members:

Member Type	Member Name	Description
2950 int	<i>posix_stream_flush_status</i>	Indicates whether a flush is in progress.
2951 int	<i>posix_stream_flush_error</i>	Indicates whether any error occurred 2952 during the last flush operation.
2953 int	<i>posix_log_overrun_status</i>	Indicates whether trace events were 2954 lost in the trace log.
2955 int	<i>posix_log_full_status</i>	The full status of the trace log.

2958 The *posix_stream_status* member indicates the operating mode of the trace stream and shall have
 2959 one of the following values defined by manifest constants in the `<trace.h>` header:

2960 **POSIX_TRACE_RUNNING**

2961 Tracing is in progress; that is, the trace stream is accepting trace events.

2962 **POSIX_TRACE_SUSPENDED**

2963 The trace stream is not accepting trace events. The tracing operation has not yet started or
 2964 has stopped, either following a *posix_trace_stop()* function call or because the trace resources
 2965 are exhausted.

2966 The *posix_stream_full_status* member indicates the full status of the trace stream, and it shall have
 2967 one of the following values defined by manifest constants in the `<trace.h>` header:

2968 **POSIX_TRACE_FULL**

2969 The space in the trace stream for trace events is exhausted.

2970 **POSIX_TRACE_NOT_FULL**

2971 There is still space available in the trace stream.

2972 The combination of the *posix_stream_status* and *posix_stream_full_status* members also indicates
 2973 the actual status of the stream. The status shall be interpreted as follows:

2974 **POSIX_TRACE_RUNNING and POSIX_TRACE_NOT_FULL**

2975 This status combination indicates that tracing is in progress, and there is space available for
 2976 recording more trace events.

2977 **POSIX_TRACE_RUNNING and POSIX_TRACE_FULL**

2978 This status combination indicates that tracing is in progress and that the trace stream is full
 2979 of trace events. This status combination cannot occur unless the *stream-full-policy* is set to

2980 POSIX_TRACE_LOOP. The trace stream contains trace events recorded during a moving
 2981 time window of prior trace events, and some older trace events may have been overwritten
 2982 and thus lost.

2983 POSIX_TRACE_SUSPENDED and POSIX_TRACE_NOT_FULL

2984 This status combination indicates that tracing has not yet been started, has been stopped by
 2985 the *posix_trace_stop()* function, or has been cleared by the *posix_trace_clear()* function.

2986 POSIX_TRACE_SUSPENDED and POSIX_TRACE_FULL

2987 This status combination indicates that tracing has been stopped by the implementation
 2988 because the *stream-full-policy* attribute was POSIX_TRACE_UNTIL_FULL and trace
 2989 resources were exhausted, or that the trace stream was stopped by the function
 2990 *posix_trace_stop()* at a time when trace resources were exhausted.

2991 The *posix_stream_outrun_status* member indicates whether trace events were lost in the trace
 2992 stream, and shall have one of the following values defined by manifest constants in the
 2993 `<trace.h>` header:

2994 POSIX_TRACE_OVERRUN

2995 At least one trace event was lost and thus was not recorded in the trace stream.

2996 POSIX_TRACE_NO_OVERRUN

2997 No trace events were lost.

2998 When the corresponding trace stream is created, the *posix_stream_outrun_status* member shall be
 2999 set to POSIX_TRACE_NO_OVERRUN.

3000 Whenever an overrun occurs, the *posix_stream_outrun_status* member shall be set to
 3001 POSIX_TRACE_OVERRUN.

3002 An overrun occurs when:

- 3003 • The policy is POSIX_TRACE_LOOP and a recorded trace event is overwritten.
- 3004 • The policy is POSIX_TRACE_UNTIL_FULL and the trace stream is full when a trace event is
 3005 generated.
- 3006 • If the Trace Log option is supported, the policy is POSIX_TRACE_FLUSH and at least one
 3007 trace event is lost while flushing the trace stream to the trace log.

3008 The *posix_stream_outrun_status* member is reset to zero after its value is read.

3009 If the Trace Log option is supported in addition to the Trace option, the *posix_stream_flush_status*,
 3010 *posix_stream_flush_error*, *posix_log_outrun_status*, and *posix_log_full_status* members are defined
 3011 as follows; otherwise, they are undefined.

3012 The *posix_stream_flush_status* member indicates whether a flush operation is being performed
 3013 and shall have one of the following values defined by manifest constants in the header
 3014 `<trace.h>`:

3015 POSIX_TRACE_FLUSHING

3016 The trace stream is currently being flushed to the trace log.

3017 POSIX_TRACE_NOT_FLUSHING

3018 No flush operation is in progress.

3019 The *posix_stream_flush_status* member shall be set to POSIX_TRACE_FLUSHING if a flush
 3020 operation is in progress either due to a call to the *posix_trace_flush()* function (explicit or caused
 3021 by a trace stream shutdown operation) or because the trace stream has become full with the
 3022 *stream-full-policy* attribute set to POSIX_TRACE_FLUSH. The *posix_stream_flush_status* member
 3023 shall be set to POSIX_TRACE_NOT_FLUSHING if no flush operation is in progress.

3024 The *posix_stream_flush_error* member shall be set to zero if no error occurred during flushing. If
 3025 an error occurred during a previous flushing operation, the *posix_stream_flush_error* member
 3026 shall be set to the value of the first error that occurred. If more than one error occurs while
 3027 flushing, error values after the first shall be discarded. The *posix_stream_flush_error* member is
 3028 reset to zero after its value is read.

3029 The *posix_log_overrun_status* member indicates whether trace events were lost in the trace log,
 3030 and shall have one of the following values defined by manifest constants in the `<trace.h>`
 3031 header:

3032 POSIX_TRACE_OVERRUN
 3033 At least one trace event was lost.

3034 POSIX_TRACE_NO_OVERRUN
 3035 No trace events were lost.

3036 When the corresponding trace stream is created, the *posix_log_overrun_status* member shall be set
 3037 to POSIX_TRACE_NO_OVERRUN. Whenever an overrun occurs, this status shall be set to
 3038 POSIX_TRACE_OVERRUN. The *posix_log_overrun_status* member is reset to zero after its value
 3039 is read.

3040 The *posix_log_full_status* member indicates the full status of the trace log, and it shall have one of
 3041 the following values defined by manifest constants in the `<trace.h>` header:

3042 POSIX_TRACE_FULL
 3043 The space in the trace log is exhausted.

3044 POSIX_TRACE_NOT_FULL
 3045 There is still space available in the trace log.

3046 The *posix_log_full_status* member is only meaningful if the *log-full-policy* attribute is either
 3047 POSIX_TRACE_UNTIL_FULL or POSIX_TRACE_LOOP.

3048 For an active trace stream without log, that is created by the *posix_trace_create()* function, the
 3049 *posix_log_overrun_status* member shall be set to POSIX_TRACE_NO_OVERRUN and the
 3050 *posix_log_full_status* member shall be set to POSIX_TRACE_NOT_FULL.

3051 **posix_trace_event_info Structure**

3052 The trace event structure **posix_trace_event_info** contains the information for one recorded
 3053 trace event. This structure is returned by the set of functions *posix_trace_getnext_event()*,
 3054 *posix_trace_timedgetnext_event()*, and *posix_trace_trygetnext_event()*.

3055 The **posix_trace_event_info** structure defined in `<trace.h>` shall contain at least the following
 3056 members:

Member Type	Member Name	Description
trace_event_id_t	<i>posix_event_id</i>	Trace event type identification.
pid_t	<i>posix_pid</i>	Process ID of the process that generated the trace event.
void *	<i>posix_prog_address</i>	Address at which the trace point was invoked.
int	<i>posix_truncation_status</i>	Status about the truncation of the data associated with this trace event.
struct timespec	<i>posix_timestamp</i>	Time at which the trace event was generated.

3065 In addition, if the Trace option and the Threads option are both supported, the
 3066 **posix_trace_event_info** structure defined in `<trace.h>` shall contain the following additional
 3067 member:

3068
3069
3070
3071
3072

Member Type	Member Name	Description
pthread_t	<i>posix_thread_id</i>	Thread ID of the thread that generated the trace event.

3073
3074
3075
3076
3077

The *posix_event_id* member represents the identification of the trace event type and its value is not directly defined by the user. This identification is returned by a call to one of the following functions: *posix_trace_trid_eventid_open()*, *posix_trace_eventtypelist_getnext_id()*, or *posix_trace_eventid_open()*. The name of the trace event type can be obtained by calling *posix_trace_eventid_get_name()*.

3078
3079
3080

The *posix_pid* is the process identifier of the traced process which generated the trace event. If the *posix_event_id* member is one of the implementation-defined system trace events and that trace event is not associated with any process, the *posix_pid* member shall be set to zero.

3081
3082
3083
3084
3085

For a user trace event, the *posix_prog_address* member is the process mapped address of the point at which the associated call to the *posix_trace_event()* function was made. For a system trace event, if the trace event is caused by a system service explicitly called by the application, the *posix_prog_address* member shall be the address of the process at the point where the call to that system service was made.

3086
3087
3088
3089
3090

The *posix_truncation_status* member defines whether the data associated with a trace event has been truncated at the time the trace event was generated, or at the time the trace event was read from the trace stream, or (if the Trace Log option is supported) from the trace log (see the *event* argument from the *posix_trace_getnext_event()* function). The *posix_truncation_status* member shall have one of the following values defined by manifest constants in the `<trace.h>` header:

3091
3092

POSIX_TRACE_NOT_TRUNCATED

All the traced data is available.

3093
3094

POSIX_TRACE_TRUNCATED_RECORD

Data was truncated at the time the trace event was generated.

3095
3096
3097
3098

POSIX_TRACE_TRUNCATED_READ

Data was truncated at the time the trace event was read from a trace stream or a trace log because the reader's buffer was too small. This truncation status overrides the POSIX_TRACE_TRUNCATED_RECORD status.

3099
3100
3101

The *posix_timestamp* member shall be the time at which the trace event was generated. The clock used is implementation-defined, but the resolution of this clock can be retrieved by a call to the *posix_trace_attr_getclockres()* function.

3102

If the Threads option is supported in addition to the Trace option:

3103
3104
3105

- The *posix_thread_id* member is the identifier of the thread that generated the trace event. If the *posix_event_id* member is one of the implementation-defined system trace events and that trace event is not associated with any thread, the *posix_thread_id* member shall be set to zero.

3106

Otherwise, this member is undefined.

3107 2.11.1.2 Trace Stream Attributes

3108
3109

Trace streams have attributes that compose the **posix_trace_attr_t** trace stream attributes object. This object shall contain at least the following attributes:

3110

- The *generation-version* attribute identifies the origin and version of the trace system.

- 3111 • The *trace-name* attribute is a character string defined by the trace controller, and that
3112 identifies the trace stream.
- 3113 • The *creation-time* attribute represents the time of the creation of the trace stream.
- 3114 • The *clock-resolution* attribute defines the clock resolution of the clock used to generate
3115 timestamps.
- 3116 • The *stream-min-size* attribute defines the minimum size in bytes of the trace stream strictly
3117 reserved for the trace events.
- 3118 • The *stream-full-policy* attribute defines the policy followed when the trace stream is full; its
3119 value is `POSIX_TRACE_LOOP`, `POSIX_TRACE_UNTIL_FULL`, or `POSIX_TRACE_FLUSH`.
- 3120 • The *max-data-size* attribute defines the maximum record size in bytes of a trace event.

3121 In addition, if the Trace option and the Trace Inherit option are both supported, the
3122 **posix_trace_attr_t** trace stream creation attributes object shall contain at least the following
3123 attributes:

- 3124 • The *inheritance* attribute specifies whether a newly created trace stream will inherit tracing in
3125 its parent's process trace stream. It is either `POSIX_TRACE_INHERITED` or
3126 `POSIX_TRACE_CLOSE_FOR_CHILD`.

3127 In addition, if the Trace option and the Trace Log option are both supported, the
3128 **posix_trace_attr_t** trace stream creation attributes object shall contain at least the following
3129 attribute:

- 3130 • If the file type corresponding to the trace log supports the `POSIX_TRACE_LOOP` or the
3131 `POSIX_TRACE_UNTIL_FULL` policies, the *log-max-size* attribute defines the maximum size
3132 in bytes of the trace log associated with an active trace stream. Other stream data—for
3133 example, trace attribute values—shall not be included in this size.
- 3134 • The *log-full-policy* attribute defines the policy of a trace log associated with an active trace
3135 stream to be `POSIX_TRACE_LOOP`, `POSIX_TRACE_UNTIL_FULL`, or
3136 `POSIX_TRACE_APPEND`.

3137 2.11.2 Trace Event Type Definitions

3138 2.11.2.1 System Trace Event Type Definitions

3139 The following system trace event types, defined in the `<trace.h>` header, track the invocation of
3140 the trace operations:

- 3141 • `POSIX_TRACE_START` shall be associated with a trace start operation.
- 3142 • `POSIX_TRACE_STOP` shall be associated with a trace stop operation.
- 3143 • If the Trace Event Filter option is supported, `POSIX_TRACE_FILTER` shall be associated with
3144 a trace event type filter change operation.

3145 The following system trace event types, defined in the `<trace.h>` header, report operational trace
3146 events:

- 3147 • `POSIX_TRACE_OVERFLOW` shall mark the beginning of a trace overflow condition.
- 3148 • `POSIX_TRACE_RESUME` shall mark the end of a trace overflow condition.
- 3149 • If the Trace Log option is supported, `POSIX_TRACE_FLUSH_START` shall mark the
3150 beginning of a flush operation.

3151 • If the Trace Log option is supported, POSIX_TRACE_FLUSH_STOP shall mark the end of a
 3152 flush operation.

3153 • If an implementation-defined trace error condition is reported, it shall be marked
 3154 POSIX_TRACE_ERROR.

3155 The interpretation of a trace stream or a trace log by a trace analyzer process relies on the
 3156 information recorded for each trace event, and also on system trace events that indicate the
 3157 invocation of trace control operations and trace system operational trace events.

3158 The POSIX_TRACE_START and POSIX_TRACE_STOP trace events specify the time windows
 3159 during which the trace stream is running.

3160 • The POSIX_TRACE_STOP trace event with an associated data that is equal to zero indicates
 3161 a call of the function *posix_trace_stop()*.

3162 • The POSIX_TRACE_STOP trace event with an associated data that is different from zero
 3163 indicates an automatic stop of the trace stream (see *posix_trace_attr_getstreamfullpolicy()*).

3164 The POSIX_TRACE_FILTER trace event indicates that a trace event type filter value changed
 3165 while the trace stream was running.

3166 The POSIX_TRACE_ERROR serves to inform the analyzer process that an implementation-
 3167 defined internal error of the trace system occurred.

3168 The POSIX_TRACE_OVERFLOW trace event shall be reported with a timestamp equal to the
 3169 timestamp of the first trace event overwritten. This is an indication that some generated trace
 3170 events have been lost.

3171 The POSIX_TRACE_RESUME trace event shall be reported with a timestamp equal to the
 3172 timestamp of the first valid trace event reported after the overflow condition ends and shall be
 3173 reported before this first valid trace event. This is an indication that the trace system is reliably
 3174 recording trace events after an overflow condition.

3175 Each of these trace event types shall be defined by a constant trace event name and a
 3176 **trace_event_id_t** constant; trace event data is associated with some of these trace events.

3177 If the Trace option is supported and the Trace Event Filter option and the Trace Log option are
 3178 not supported, the following predefined system trace events in Table 2-3 shall be defined:

3179 **Table 2-3** Trace Option: System Trace Events

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error int
posix_trace_start	POSIX_TRACE_START	None.
posix_trace_stop	POSIX_TRACE_STOP	auto int
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.

3189 If the Trace option and the Trace Event Filter option are both supported, and if the Trace Log
 3190 option is not supported, the following predefined system trace events in Table 2-4 (on page 78)
 3191 shall be defined:

3192

Table 2-4 Trace and Trace Event Filter Options: System Trace Events

3193

3194

3195

3196

3197

3198

3199

3200

3201

3202

3203

3204

3205

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error
		int
posix_trace_start	POSIX_TRACE_START	event_filter
		trace_event_set_t
posix_trace_stop	POSIX_TRACE_STOP	auto
		int
posix_trace_filter	POSIX_TRACE_FILTER	old_event_filter
		new_event_filter
		trace_event_set_t
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.

3206

3207

3208

If the Trace option and the Trace Log option are both supported, and if the Trace Event Filter option is not supported, the following predefined system trace events in Table 2-5 shall be defined:

3209

Table 2-5 Trace and Trace Log Options: System Trace Events

3210

3211

3212

3213

3214

3215

3216

3217

3218

3219

3220

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error
		int
posix_trace_start	POSIX_TRACE_START	None.
posix_trace_stop	POSIX_TRACE_STOP	auto
		int
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.
posix_trace_flush_start	POSIX_TRACE_FLUSH_START	None.
posix_trace_flush_stop	POSIX_TRACE_FLUSH_STOP	None.

3221

3222

If the Trace option, the Trace Event Filter option, and the Trace Log option are all supported, the following predefined system trace events in Table 2-6 (on page 79) shall be defined:

3223 **Table 2-6** Trace, Trace Log, and Trace Event Filter Options: System Trace Events

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error int
posix_trace_start	POSIX_TRACE_START	event_filter trace_event_set_t
posix_trace_stop	POSIX_TRACE_STOP	auto int
posix_trace_filter	POSIX_TRACE_FILTER	old_event_filter new_event_filter trace_event_set_t
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.
posix_trace_flush_start	POSIX_TRACE_FLUSH_START	None.
posix_trace_flush_stop	POSIX_TRACE_FLUSH_STOP	None.

3239 **2.11.2.2** *User Trace Event Type Definitions*

3240 The user trace event `POSIX_TRACE_UNNAMED_USEREVENT` is defined in the `<trace.h>`
 3241 header. If the limit of per-process user trace event names represented by
 3242 `{TRACE_USER_EVENT_MAX}` has already been reached, this predefined user event shall be
 3243 returned when the application tries to register more events than allowed. The data associated
 3244 with this trace event is application-defined.

3245 The following predefined user trace event in Table 2-7 shall be defined:

3246 **Table 2-7** Trace Option: User Trace Event

Event Name	Constant
posix_trace_unnamed_userevent	POSIX_TRACE_UNNAMED_USEREVENT

3249 **2.11.3** **Trace Functions**

3250 The trace interface is built and structured to improve portability through use of trace data of
 3251 opaque type. The object-oriented approach for the manipulation of trace attributes and trace
 3252 event type identifiers requires definition of many constructor and selector functions which
 3253 operate on these opaque types. Also, the trace interface must support several different tracing
 3254 roles. To facilitate reading the trace interface, the trace functions are grouped into small
 3255 functional sets supporting the three different roles:

- 3256 • A trace controller process requires functions to set up and customize all the resources needed
 3257 to run a trace stream, including:
 - 3258 — Attribute initialization and destruction (`posix_trace_attr_init()`)
 - 3259 — Identification information manipulation (`posix_trace_attr_getgenversion()`)
 - 3260 — Trace system behavior modification (`posix_trace_attr_getinherited()`)
 - 3261 — Trace stream and trace log size set (`posix_trace_attr_getmaxusereventsize()`)

- 3262 — Trace stream creation, flush, and shutdown (*posix_trace_create()*)
- 3263 — Trace stream and trace log clear (*posix_trace_clear()*)
- 3264 — Trace event type identifier manipulation (*posix_trace_trid_eventid_open()*)
- 3265 — Trace event type identifier list exploration (*posix_trace_eventtypelist_getnext_id()*)
- 3266 — Trace event type set manipulation (*posix_trace_eventset_empty()*)
- 3267 — Trace event type filter set (*posix_trace_set_filter()*)
- 3268 — Trace stream start and stop (*posix_trace_start()*)
- 3269 — Trace stream information and status read (*posix_trace_get_attr()*)
- 3270 • A traced process requires functions to instrument trace points:
- 3271 — Trace event type identifiers definition and trace points insertion (*posix_trace_event()*)
- 3272 • A trace analyzer process requires functions to retrieve information from a trace stream and
- 3273 trace log:
- 3274 — Identification information read (*posix_trace_attr_getgenversion()*)
- 3275 — Trace system behavior information read (*posix_trace_attr_getinherited()*)
- 3276 — Trace stream and trace log size get (*posix_trace_attr_getmaxusereventsized()*)
- 3277 — Trace event type identifier manipulation (*posix_trace_trid_eventid_open()*)
- 3278 — Trace event type identifier list exploration (*posix_trace_eventtypelist_getnext_id()*)
- 3279 — Trace log open, rewind, and close (*posix_trace_open()*)
- 3280 — Trace stream information and status read (*posix_trace_get_attr()*)
- 3281 — Trace event read (*posix_trace_getnext_event()*)

3282 2.12 Data Types

3283 All of the data types used by various functions are defined by the implementation. The
 3284 following table describes some of these types. Other types referenced in the description of a
 3285 function, not mentioned here, can be found in the appropriate header for that function.

3286	Defined Type	Description
3288	cc_t	Type used for terminal special characters.
3289	clock_t	Integer or real-floating type used for processor times, as defined in the ISO C standard.
3290		
3291	clockid_t	Used for clock ID type in some timer functions.
3292	dev_t	Arithmetic type used for device numbers.
3293	DIR	Type representing a directory stream.
3294	div_t	Structure type returned by the <i>div()</i> function.
3295	FILE	Structure containing information about a file.
3296	glob_t	Structure type used in pathname pattern matching.
3297	fpos_t	Type containing all information needed to specify uniquely every

	Defined Type	Description
3298		position within a file.
3299		
3300		
3301	gid_t	Integer type used for group IDs.
3302	iconv_t	Type used for conversion descriptors.
3303	id_t	Integer type used as a general identifier; can be used to contain at least the largest of a pid_t , uid_t , or gid_t .
3304		
3305	ino_t	Unsigned integer type used for file serial numbers.
3306	key_t	Arithmetic type used for XSI interprocess communication.
3307	ldiv_t	Structure type returned by the <i>ldiv()</i> function.
3308	mode_t	Integer type used for file attributes.
3309	mqd_t	Used for message queue descriptors.
3310	nfds_t	Integer type used for the number of file descriptors.
3311	nlink_t	Integer type used for link counts.
3312	off_t	Signed integer type used for file sizes.
3313	pid_t	Signed integer type used for process and process group IDs.
3314	pthread_attr_t	Used to identify a thread attribute object.
3315	pthread_cond_t	Used for condition variables.
3316	pthread_condattr_t	Used to identify a condition attribute object.
3317	pthread_key_t	Used for thread-specific data keys.
3318	pthread_mutex_t	Used for mutexes.
3319	pthread_mutexattr_t	Used to identify a mutex attribute object.
3320	pthread_once_t	Used for dynamic package initialization.
3321	pthread_rwlock_t	Used for read-write locks.
3322	pthread_rwlockattr_t	Used for read-write lock attributes.
3323	pthread_t	Used to identify a thread.
3324	ptrdiff_t	Signed integer type of the result of subtracting two pointers.
3325	regex_t	Structure type used in regular expression matching.
3326	regmatch_t	Structure type used in regular expression matching.
3327	rlim_t	Unsigned integer type used for limit values, to which objects of type int and off_t can be cast without loss of value.
3328		
3329	sem_t	Type used in performing semaphore operations.
3330	sig_atomic_t	Integer type of an object that can be accessed as an atomic entity, even in the presence of asynchronous interrupts.
3331		
3332	sigset_t	Integer or structure type of an object used to represent sets of signals.
3333		
3334	size_t	Unsigned integer type used for size of objects.
3335	speed_t	Type used for terminal baud rates.
3336	ssize_t	Signed integer type used for a count of bytes or an error indication.
3337		
3338	suseconds_t	Signed integer type used for time in microseconds.
3339	tcflag_t	Type used for terminal modes.
3340	time_t	Integer or real-floating type used for time in seconds, as defined in the ISO C standard.
3341		
3342	timer_t	Used for timer ID returned by the <i>timer_create()</i> function.
3343	uid_t	Integer type used for user IDs.
3344	useconds_t	Unsigned integer type used for time in microseconds.
3345	va_list	Type used for traversing variable argument lists.
3346	wchar_t	Integer type whose range of values can represent distinct codes for

3347
3348
3349
3350
3351
3352
3353
3354
3355

Defined Type	Description
	all members of the largest extended character set specified by the supported locales.
wctype_t	Scalar type which represents a character class descriptor.
wint_t	Integer type capable of storing any valid value of wchar_t or WEOF.
wordexp_t	Structure type used in word expansion.

System Interfaces

3356

3357
3358

This chapter describes the functions, macros, and external variables to support applications portability at the C-language source level.

3359 **NAME**3360 **FD_CLR** — macros for synchronous I/O multiplexing3361 **SYNOPSIS**3362 `#include <sys/time.h>`3363 `FD_CLR(int fd, fd_set *fdset);`3364 `FD_ISSET(int fd, fd_set *fdset);`3365 `FD_SET(int fd, fd_set *fdset);`3366 `FD_ZERO(fd_set *fdset);`3367 **DESCRIPTION**3368 Refer to *pselect()*.

3369 **NAME**

3370 _Exit, _exit — terminate a process

3371 **SYNOPSIS**

3372 #include <stdlib.h>

3373 void _Exit(int *status*);

3374 #include <unistd.h>

3375 void _exit(int *status*);3376 **DESCRIPTION**3377 Refer to *exit()*.

3378 **NAME**

3379 _longjmp, _setjmp — non-local goto

3380 **SYNOPSIS**

```
3381 xSI       #include <setjmp.h>
3382       void _longjmp(jmp_buf env, int val);
3383       int _setjmp(jmp_buf env);
3384
```

3385 **DESCRIPTION**

3386 The *_longjmp()* and *_setjmp()* functions shall be equivalent to *longjmp()* and *setjmp()*,
3387 respectively, with the additional restriction that *_longjmp()* and *_setjmp()* shall not manipulate
3388 the signal mask.

3389 If *_longjmp()* is called even though *env* was never initialized by a call to *_setjmp()*, or when the
3390 last such call was in a function that has since returned, the results are undefined.

3391 **RETURN VALUE**

3392 Refer to *longjmp()* and *setjmp()*.

3393 **ERRORS**

3394 No errors are defined.

3395 **EXAMPLES**

3396 None.

3397 **APPLICATION USAGE**

3398 If *_longjmp()* is executed and the environment in which *_setjmp()* was executed no longer exists,
3399 errors can occur. The conditions under which the environment of the *_setjmp()* no longer exists
3400 include exiting the function that contains the *_setjmp()* call, and exiting an inner block with
3401 temporary storage. This condition might not be detectable, in which case the *_longjmp()* occurs
3402 and, if the environment no longer exists, the contents of the temporary storage of an inner block
3403 are unpredictable. This condition might also cause unexpected process termination. If the
3404 function has returned, the results are undefined.

3405 Passing *longjmp()* a pointer to a buffer not created by *setjmp()*, passing *_longjmp()* a pointer to a
3406 buffer not created by *_setjmp()*, passing *siglongjmp()* a pointer to a buffer not created by
3407 *sigsetjmp()*, or passing any of these three functions a buffer that has been modified by the user
3408 can cause all the problems listed above, and more.

3409 The *_longjmp()* and *_setjmp()* functions are included to support programs written to historical
3410 system interfaces. New applications should use *siglongjmp()* and *sigsetjmp()* respectively.

3411 **RATIONALE**

3412 None.

3413 **FUTURE DIRECTIONS**

3414 The *_longjmp()* and *_setjmp()* functions may be marked LEGACY in a future version.

3415 **SEE ALSO**

3416 *longjmp()*, *setjmp()*, *siglongjmp()*, *sigsetjmp()*, the Base Definitions volume of
3417 IEEE Std 1003.1-2001, <setjmp.h>

3418 **CHANGE HISTORY**

3419 First released in Issue 4, Version 2.

3420 **Issue 5**

3421 Moved from X/OPEN UNIX extension to BASE.

3422 **NAME**

3423 _toupper — transliterate uppercase characters to lowercase

3424 **SYNOPSIS**

3425 XSI #include <ctype.h>

3426 int _tolower(int c);

3427

3428 **DESCRIPTION**

3429 The *_tolower()* macro shall be equivalent to *tolower(c)* except that the application shall ensure
3430 that the argument *c* is an uppercase letter.

3431 **RETURN VALUE**

3432 Upon successful completion, *_tolower()* shall return the lowercase letter corresponding to the
3433 argument passed.

3434 **ERRORS**

3435 No errors are defined.

3436 **EXAMPLES**

3437 None.

3438 **APPLICATION USAGE**

3439 None.

3440 **RATIONALE**

3441 None.

3442 **FUTURE DIRECTIONS**

3443 None.

3444 **SEE ALSO**

3445 *tolower()*, *isupper()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,
3446 <ctype.h>

3447 **CHANGE HISTORY**

3448 First released in Issue 1. Derived from Issue 1 of the SVID.

3449 **Issue 6**

3450 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

3451 **NAME**

3452 _toupper — transliterate lowercase characters to uppercase

3453 **SYNOPSIS**

3454 xSI #include <ctype.h>

3455 int _toupper(int c);

3456

3457 **DESCRIPTION**3458 The *_toupper()* macro shall be equivalent to *toupper()* except that the application shall ensure
3459 that the argument *c* is a lowercase letter.3460 **RETURN VALUE**3461 Upon successful completion, *_toupper()* shall return the uppercase letter corresponding to the
3462 argument passed.3463 **ERRORS**

3464 No errors are defined.

3465 **EXAMPLES**

3466 None.

3467 **APPLICATION USAGE**

3468 None.

3469 **RATIONALE**

3470 None.

3471 **FUTURE DIRECTIONS**

3472 None.

3473 **SEE ALSO**3474 *islower()*, *toupper()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,
3475 <ctype.h>3476 **CHANGE HISTORY**

3477 First released in Issue 1. Derived from Issue 1 of the SVID.

3478 **Issue 6**

3479 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

3480 **NAME**

3481 a64l, l64a — convert between a 32-bit integer and a radix-64 ASCII string

3482 **SYNOPSIS**

3483 xSI #include <stdlib.h>

3484 long a64l(const char *s);

3485 char *l64a(long value);

3486

3487 **DESCRIPTION**

3488 These functions maintain numbers stored in radix-64 ASCII characters. This is a notation by
 3489 which 32-bit integers can be represented by up to six characters; each character represents a digit
 3490 in radix-64 notation. If the type **long** contains more than 32 bits, only the low-order 32 bits shall
 3491 be used for these operations.

3492 The characters used to represent digits are '.' (dot) for 0, '/' for 1, '0' through '9' for [2,11],
 3493 'A' through 'Z' for [12,37], and 'a' through 'z' for [38,63].

3494 The *a64l()* function shall take a pointer to a radix-64 representation, in which the first digit is the
 3495 least significant, and return the corresponding **long** value. If the string pointed to by *s* contains
 3496 more than six characters, *a64l()* shall use the first six. If the first six characters of the string
 3497 contain a null terminator, *a64l()* shall use only characters preceding the null terminator. The
 3498 *a64l()* function shall scan the character string from left to right with the least significant digit on
 3499 the left, decoding each character as a 6-bit radix-64 number. If the type **long** contains more than
 3500 32 bits, the resulting value is sign-extended. The behavior of *a64l()* is unspecified if *s* is a null
 3501 pointer or the string pointed to by *s* was not generated by a previous call to *l64a()*.

3502 The *l64a()* function shall take a **long** argument and return a pointer to the corresponding radix-
 3503 64 representation. The behavior of *l64a()* is unspecified if *value* is negative.

3504 The value returned by *l64a()* may be a pointer into a static buffer. Subsequent calls to *l64a()* may
 3505 overwrite the buffer.

3506 The *l64a()* function need not be reentrant. A function that is not required to be reentrant is not
 3507 required to be thread-safe.

3508 **RETURN VALUE**

3509 Upon successful completion, *a64l()* shall return the **long** value resulting from conversion of the
 3510 input string. If a string pointed to by *s* is an empty string, *a64l()* shall return 0L.

3511 The *l64a()* function shall return a pointer to the radix-64 representation. If *value* is 0L, *l64a()* shall
 3512 return a pointer to an empty string.

3513 **ERRORS**

3514 No errors are defined.

3515 **EXAMPLES**

3516 None.

3517 **APPLICATION USAGE**3518 If the type **long** contains more than 32 bits, the result of *a64l(l64a(x))* is *x* in the low-order 32 bits.3519 **RATIONALE**3520 This is not the same encoding as used by either encoding variant of the *uuencode* utility.

3521 **FUTURE DIRECTIONS**

3522 None.

3523 **SEE ALSO**3524 *strtol()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<stdlib.h>`, the Shell and Utilities
3525 volume of IEEE Std 1003.1-2001, *uuencode*3526 **CHANGE HISTORY**

3527 First released in Issue 4, Version 2.

3528 **Issue 5**

3529 Moved from X/OPEN UNIX extension to BASE.

3530 Normative text previously in the APPLICATION USAGE section is moved to the
3531 DESCRIPTION.

3532 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

3533 **NAME**

3534 abort — generate an abnormal process abort

3535 **SYNOPSIS**

3536 #include <stdlib.h>

3537 void abort(void);

3538 **DESCRIPTION**

3539 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 3540 conflict between the requirements described here and the ISO C standard is unintentional. This
 3541 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

3542 The *abort()* function shall cause abnormal process termination to occur, unless the signal
 3543 SIGABRT is being caught and the signal handler does not return.

3544 cx The abnormal termination processing shall include the default actions defined for SIGABRT and
 3545 may include an attempt to effect *fclose()* on all open streams.

3546 The SIGABRT signal shall be sent to the calling process as if by means of *raise()* with the
 3547 argument SIGABRT.

3548 cx The status made available to *wait()* or *waitpid()* by *abort()* shall be that of a process terminated
 3549 by the SIGABRT signal. The *abort()* function shall override blocking or ignoring the SIGABRT
 3550 signal.

3551 **RETURN VALUE**3552 The *abort()* function shall not return.3553 **ERRORS**

3554 No errors are defined.

3555 **EXAMPLES**

3556 None.

3557 **APPLICATION USAGE**

3558 Catching the signal is intended to provide the application writer with a portable means to abort
 3559 processing, free from possible interference from any implementation-defined functions.

3560 **RATIONALE**

3561 The ISO/IEC 9899:1999 standard requires the *abort()* function to be async-signal-safe. Since
 3562 IEEE Std 1003.1-2001 defers to the ISO C standard, this required a change to the DESCRIPTION
 3563 from “shall include the effect of *fclose()*” to “may include an attempt to effect *fclose()*.”

3564 The revised wording permits some backwards-compatibility and avoids a potential deadlock
 3565 situation.

3566 The Open Group Base Resolution bwg2002-003 is applied, removing the following XSI shaded
 3567 paragraph from the DESCRIPTION:

3568 “On XSI-conformant systems, in addition the abnormal termination processing shall include the
 3569 effect of *fclose()* on message catalog descriptors.”

3570 There were several reasons to remove this paragraph:

- 3571 • No special processing of open message catalogs needs to be performed prior to abnormal
 3572 process termination.
- 3573 • The main reason to specifically mention that *abort()* includes the effect of *fclose()* on open
 3574 streams is to flush output queued on the stream. Message catalogs in this context are read-
 3575 only and, therefore, do not need to be flushed.

3576 • The effect of *fclose()* on a message catalog descriptor is unspecified. Message catalog
3577 descriptors are allowed, but not required to be implemented using a file descriptor, but there
3578 is no mention in IEEE Std 1003.1-2001 of a message catalog descriptor using a standard I/O
3579 stream FILE object as would be expected by *fclose()*.

3580 **FUTURE DIRECTIONS**

3581 None.

3582 **SEE ALSO**

3583 *exit()*, *kill()*, *raise()*, *signal()*, *wait()*, *waitpid()*, the Base Definitions volume of
3584 IEEE Std 1003.1-2001, <stdlib.h>

3585 **CHANGE HISTORY**

3586 First released in Issue 1. Derived from Issue 1 of the SVID.

3587 **Issue 6**

3588 Extensions beyond the ISO C standard are marked.

3589 Changes are made to the DESCRIPTION for alignment with the ISO/IEC 9899:1999 standard.

3590 The Open Group Base Resolution bwg2002-003 is applied.

3591 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/10 is applied, changing the
3592 DESCRIPTION of abnormal termination processing and adding to the RATIONALE section.

3593 **NAME**

3594 abs — return an integer absolute value

3595 **SYNOPSIS**

3596 #include <stdlib.h>

3597 int abs(int i);

3598 **DESCRIPTION**

3599 cx The functionality described on this reference page is aligned with the ISO C standard. Any
3600 conflict between the requirements described here and the ISO C standard is unintentional. This
3601 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

3602 The *abs()* function shall compute the absolute value of its integer operand, *i*. If the result cannot
3603 be represented, the behavior is undefined.

3604 **RETURN VALUE**3605 The *abs()* function shall return the absolute value of its integer operand.3606 **ERRORS**

3607 No errors are defined.

3608 **EXAMPLES**

3609 None.

3610 **APPLICATION USAGE**

3611 In two's-complement representation, the absolute value of the negative integer with largest
3612 magnitude {INT_MIN} might not be representable.

3613 **RATIONALE**

3614 None.

3615 **FUTURE DIRECTIONS**

3616 None.

3617 **SEE ALSO**3618 *fabs()*, *labs()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>3619 **CHANGE HISTORY**

3620 First released in Issue 1. Derived from Issue 1 of the SVID.

3621 **Issue 6**

3622 Extensions beyond the ISO C standard are marked.

3623 **NAME**

3624 accept — accept a new connection on a socket

3625 **SYNOPSIS**

3626 #include <sys/socket.h>

3627 int accept(int *socket*, struct sockaddr *restrict *address*,
3628 socklen_t *restrict *address_len*);3629 **DESCRIPTION**3630 The *accept()* function shall extract the first connection on the queue of pending connections,
3631 create a new socket with the same socket type protocol and address family as the specified
3632 socket, and allocate a new file descriptor for that socket.3633 The *accept()* function takes the following arguments:3634 *socket* Specifies a socket that was created with *socket()*, has been bound to an address
3635 with *bind()*, and has issued a successful call to *listen()*.3636 *address* Either a null pointer, or a pointer to a **sockaddr** structure where the address of
3637 the connecting socket shall be returned.3638 *address_len* Points to a **socklen_t** structure which on input specifies the length of the
3639 supplied **sockaddr** structure, and on output specifies the length of the stored
3640 address.3641 If *address* is not a null pointer, the address of the peer for the accepted connection shall be stored
3642 in the **sockaddr** structure pointed to by *address*, and the length of this address shall be stored in
3643 the object pointed to by *address_len*.3644 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,
3645 the stored address shall be truncated.3646 If the protocol permits connections by unbound clients, and the peer is not bound, then the value
3647 stored in the object pointed to by *address* is unspecified.3648 If the listen queue is empty of connection requests and O_NONBLOCK is not set on the file
3649 descriptor for the socket, *accept()* shall block until a connection is present. If the *listen()* queue is
3650 empty of connection requests and O_NONBLOCK is set on the file descriptor for the socket,
3651 *accept()* shall fail and set *errno* to [EAGAIN] or [EWOULDBLOCK].3652 The accepted socket cannot itself accept more connections. The original socket remains open and
3653 can accept more connections.3654 **RETURN VALUE**3655 Upon successful completion, *accept()* shall return the non-negative file descriptor of the accepted
3656 socket. Otherwise, -1 shall be returned and *errno* set to indicate the error.3657 **ERRORS**3658 The *accept()* function shall fail if:

3659 [EAGAIN] or [EWOULDBLOCK]

3660 O_NONBLOCK is set for the socket file descriptor and no connections are
3661 present to be accepted.3662 [EBADF] The *socket* argument is not a valid file descriptor.

3663 [ECONNABORTED]

3664 A connection has been aborted.

3665	[EINTR]	The <i>accept()</i> function was interrupted by a signal that was caught before a valid connection arrived.
3666		
3667	[EINVAL]	The <i>socket</i> is not accepting connections.
3668	[EMFILE]	{OPEN_MAX} file descriptors are currently open in the calling process.
3669	[ENFILE]	The maximum number of file descriptors in the system are already open.
3670	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
3671	[EOPNOTSUPP]	The socket type of the specified socket does not support accepting connections.
3672		
3673		The <i>accept()</i> function may fail if:
3674	[ENOBUFS]	No buffer space is available.
3675	[ENOMEM]	There was insufficient memory available to complete the operation.
3676	XSR [EPROTO]	A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized.
3677		
3678	EXAMPLES	
3679		None.
3680	APPLICATION USAGE	
3681		When a connection is available, <i>select()</i> indicates that the file descriptor for the socket is ready for reading.
3682		
3683	RATIONALE	
3684		None.
3685	FUTURE DIRECTIONS	
3686		None.
3687	SEE ALSO	
3688		<i>bind()</i> , <i>connect()</i> , <i>listen()</i> , <i>socket()</i> , the Base Definitions volume of IEEE Std 1003.1-2001, <sys/socket.h>
3689		
3690	CHANGE HISTORY	
3691		First released in Issue 6. Derived from the XNS, Issue 5.2 specification.
3692		The restrict keyword is added to the <i>accept()</i> prototype for alignment with the ISO/IEC 9899:1999 standard.
3693		

3694 **NAME**

3695 access — determine accessibility of a file

3696 **SYNOPSIS**

3697 #include <unistd.h>

3698 int access(const char *path, int amode);

3699 **DESCRIPTION**3700 The `access()` function shall check the file named by the pathname pointed to by the `path`
3701 argument for accessibility according to the bit pattern contained in `amode`, using the real user ID
3702 in place of the effective user ID and the real group ID in place of the effective group ID.3703 The value of `amode` is either the bitwise-inclusive OR of the access permissions to be checked
3704 (R_OK, W_OK, X_OK) or the existence test (F_OK).3705 If any access permissions are checked, each shall be checked individually, as described in the
3706 Base Definitions volume of IEEE Std 1003.1-2001, Chapter 3, Definitions. If the process has
3707 appropriate privileges, an implementation may indicate success for X_OK even if none of the
3708 execute file permission bits are set.3709 **RETURN VALUE**3710 If the requested access is permitted, `access()` succeeds and shall return 0; otherwise, -1 shall be
3711 returned and `errno` shall be set to indicate the error.3712 **ERRORS**3713 The `access()` function shall fail if:3714 [EACCES] Permission bits of the file mode do not permit the requested access, or search
3715 permission is denied on a component of the path prefix.3716 [ELOOP] A loop exists in symbolic links encountered during resolution of the `path`
3717 argument.3718 [ENAMETOOLONG] The length of the `path` argument exceeds {PATH_MAX} or a pathname
3719 component is longer than {NAME_MAX}.
37203721 [ENOENT] A component of `path` does not name an existing file or `path` is an empty string.

3722 [ENOTDIR] A component of the path prefix is not a directory.

3723 [EROFS] Write access is requested for a file on a read-only file system.

3724 The `access()` function may fail if:3725 [EINVAL] The value of the `amode` argument is invalid.3726 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
3727 resolution of the `path` argument.3728 [ENAMETOOLONG] As a result of encountering a symbolic link in resolution of the `path` argument,
3729 the length of the substituted pathname string exceeded {PATH_MAX}.
37303731 [ETXTBSY] Write access is requested for a pure procedure (shared text) file that is being
3732 executed.

3733 **EXAMPLES**3734 **Testing for the Existence of a File**

3735 The following example tests whether a file named **myfile** exists in the **/tmp** directory.

```
3736 #include <unistd.h>
3737 ...
3738 int result;
3739 const char *filename = "/tmp/myfile";
3740 result = access (filename, F_OK);
```

3741 **APPLICATION USAGE**

3742 Additional values of *amode* other than the set defined in the description may be valid; for
3743 example, if a system has extended access controls.

3744 **RATIONALE**

3745 In early proposals, some inadequacies in the *access()* function led to the creation of an *eaccess()*
3746 function because:

- 3747 1. Historical implementations of *access()* do not test file access correctly when the process'
3748 real user ID is superuser. In particular, they always return zero when testing execute
3749 permissions without regard to whether the file is executable.
- 3750 2. The superuser has complete access to all files on a system. As a consequence, programs
3751 started by the superuser and switched to the effective user ID with lesser privileges cannot
3752 use *access()* to test their file access permissions.

3753 However, the historical model of *eaccess()* does not resolve problem (1), so this volume of
3754 IEEE Std 1003.1-2001 now allows *access()* to behave in the desired way because several
3755 implementations have corrected the problem. It was also argued that problem (2) is more easily
3756 solved by using *open()*, *chdir()*, or one of the *exec* functions as appropriate and responding to the
3757 error, rather than creating a new function that would not be as reliable. Therefore, *eaccess()* is not
3758 included in this volume of IEEE Std 1003.1-2001.

3759 The sentence concerning appropriate privileges and execute permission bits reflects the two
3760 possibilities implemented by historical implementations when checking superuser access for
3761 *X_OK*.

3762 New implementations are discouraged from returning *X_OK* unless at least one execution
3763 permission bit is set.

3764 **FUTURE DIRECTIONS**

3765 None.

3766 **SEE ALSO**

3767 *chmod()*, *stat()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

3768 **CHANGE HISTORY**

3769 First released in Issue 1. Derived from Issue 1 of the SVID.

3770 **Issue 6**

3771 The following new requirements on POSIX implementations derive from alignment with the
3772 Single UNIX Specification:

- 3773 • The [ELOOP] mandatory error condition is added.
- 3774 • A second [ENAMETOOLONG] is added as an optional error condition.

3775 • The [ETXTBSY] optional error condition is added.

3776 The following changes were made to align with the IEEE P1003.1a draft standard:

3777 • The [ELOOP] optional error condition is added.

3778 **NAME**

3779 acos, acosf, acosl — arc cosine functions

3780 **SYNOPSIS**

3781 #include <math.h>

3782 double acos(double x);

3783 float acosf(float x);

3784 long double acosl(long double x);

3785 **DESCRIPTION**

3786 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 3787 conflict between the requirements described here and the ISO C standard is unintentional. This
 3788 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

3789 These functions shall compute the principal value of the arc cosine of their argument *x*. The
 3790 value of *x* should be in the range $[-1,1]$.

3791 An application wishing to check for error situations should set *errno* to zero and call
 3792 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 3793 *fetetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 3794 zero, an error has occurred.

3795 **RETURN VALUE**

3796 Upon successful completion, these functions shall return the arc cosine of *x*, in the range $[0,\pi]$
 3797 radians.

3798 **MX** For finite values of *x* not in the range $[-1,1]$, a domain error shall occur, and either a NaN (if
 3799 supported), or an implementation-defined value shall be returned.

3800 **MX** If *x* is NaN, a NaN shall be returned.

3801 If *x* is $+1$, $+0$ shall be returned.

3802 If *x* is $\pm\text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 3803 defined value shall be returned.

3804 **ERRORS**

3805 These functions shall fail if:

3806 **MX** Domain Error The *x* argument is finite and is not in the range $[-1,1]$, or is $\pm\text{Inf}$.

3807 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 3808 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 3809 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 3810 shall be raised.

3811 **EXAMPLES**

3812 None.

3813 **APPLICATION USAGE**

3814 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 3815 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

3816 **RATIONALE**

3817 None.

3818 **FUTURE DIRECTIONS**

3819 None.

3820 **SEE ALSO**3821 *cos()*, *feclearexcept()*, *fetetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,
3822 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>3823 **CHANGE HISTORY**

3824 First released in Issue 1. Derived from Issue 1 of the SVID.

3825 **Issue 5**3826 The DESCRIPTION is updated to indicate how an application should check for an error. This
3827 text was previously published in the APPLICATION USAGE section.3828 **Issue 6**3829 The *acosf()* and *acosl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.3830 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
3831 revised to align with the ISO/IEC 9899:1999 standard.3832 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
3833 marked.

3834 **NAME**

3835 acosh, acoshf, acoshl — inverse hyperbolic cosine functions

3836 **SYNOPSIS**

3837 #include <math.h>

3838 double acosh(double x);

3839 float acoshf(float x);

3840 long double acoshl(long double x);

3841 **DESCRIPTION**

3842 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 3843 conflict between the requirements described here and the ISO C standard is unintentional. This
 3844 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

3845 These functions shall compute the inverse hyperbolic cosine of their argument *x*.

3846 An application wishing to check for error situations should set *errno* to zero and call
 3847 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 3848 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 3849 zero, an error has occurred.

3850 **RETURN VALUE**3851 Upon successful completion, these functions shall return the inverse hyperbolic cosine of their
3852 argument.

3853 **MX** For finite values of $x < 1$, a domain error shall occur, and either a NaN (if supported), or an
 3854 implementation-defined value shall be returned.

3855 **MX** If *x* is NaN, a NaN shall be returned.3856 If *x* is +1, +0 shall be returned.3857 If *x* is +Inf, +Inf shall be returned.

3858 If *x* is -Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-
 3859 defined value shall be returned.

3860 **ERRORS**

3861 These functions shall fail if:

3862 **MX** Domain Error The *x* argument is finite and less than +1.0, or is -Inf.

3863 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 3864 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 3865 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 3866 shall be raised.

3867 **EXAMPLES**

3868 None.

3869 **APPLICATION USAGE**

3870 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 3871 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

3872 **RATIONALE**

3873 None.

3874 **FUTURE DIRECTIONS**

3875 None.

3876 **SEE ALSO**3877 *cosh()*, *feclearexcept()*, *fetetestexcept()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section
3878 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>3879 **CHANGE HISTORY**

3880 First released in Issue 4, Version 2.

3881 **Issue 5**

3882 Moved from X/OPEN UNIX extension to BASE.

3883 **Issue 6**3884 The *acosh()* function is no longer marked as an extension.3885 The *acoshf()* and *acoshl()* functions are added for alignment with the ISO/IEC 9899:1999
3886 standard.3887 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
3888 revised to align with the ISO/IEC 9899:1999 standard.3889 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
3890 marked.

3891 **NAME**3892 *acosl* — arc cosine functions3893 **SYNOPSIS**

3894 #include <math.h>

3895 long double *acosl*(long double *x*);3896 **DESCRIPTION**3897 Refer to *acos*().

3898 **NAME**3899 aio_cancel — cancel an asynchronous I/O request (**REALTIME**)3900 **SYNOPSIS**

3901 AIO #include <aio.h>

3902 int aio_cancel(int *fildes*, struct aiocb **aiocbp*);

3903

3904 **DESCRIPTION**

3905 The *aio_cancel()* function shall attempt to cancel one or more asynchronous I/O requests
 3906 currently outstanding against file descriptor *fildes*. The *aiocbp* argument points to the
 3907 asynchronous I/O control block for a particular request to be canceled. If *aiocbp* is NULL, then
 3908 all outstanding cancelable asynchronous I/O requests against *fildes* shall be canceled.

3909 Normal asynchronous notification shall occur for asynchronous I/O operations that are
 3910 successfully canceled. If there are requests that cannot be canceled, then the normal
 3911 asynchronous completion process shall take place for those requests when they are completed.

3912 For requested operations that are successfully canceled, the associated error status shall be set to
 3913 [ECANCELED] and the return status shall be -1. For requested operations that are not
 3914 successfully canceled, the *aiocbp* shall not be modified by *aio_cancel()*.

3915 If *aiocbp* is not NULL, then if *fildes* does not have the same value as the file descriptor with which
 3916 the asynchronous operation was initiated, unspecified results occur.

3917 Which operations are cancelable is implementation-defined.

3918 **RETURN VALUE**

3919 The *aio_cancel()* function shall return the value AIO_CANCELED to the calling process if the
 3920 requested operation(s) were canceled. The value AIO_NOTCANCELED shall be returned if at
 3921 least one of the requested operation(s) cannot be canceled because it is in progress. In this case,
 3922 the state of the other operations, if any, referenced in the call to *aio_cancel()* is not indicated by
 3923 the return value of *aio_cancel()*. The application may determine the state of affairs for these
 3924 operations by using *aio_error()*. The value AIO_ALLDONE is returned if all of the operations
 3925 have already completed. Otherwise, the function shall return -1 and set *errno* to indicate the
 3926 error.

3927 **ERRORS**3928 The *aio_cancel()* function shall fail if:3929 [EBADF] The *fildes* argument is not a valid file descriptor.3930 **EXAMPLES**

3931 None.

3932 **APPLICATION USAGE**

3933 The *aio_cancel()* function is part of the Asynchronous Input and Output option and need not be
 3934 available on all implementations.

3935 **RATIONALE**

3936 None.

3937 **FUTURE DIRECTIONS**

3938 None.

3939 **SEE ALSO**3940 *aio_read()*, *aio_write()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**aio.h**>3941 **CHANGE HISTORY**

3942 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

3943 **Issue 6**3944 The [ENOSYS] error condition has been removed as stubs need not be provided if an
3945 implementation does not support the Asynchronous Input and Output option.

3946 The APPLICATION USAGE section is added.

3947 **NAME**3948 aio_error — retrieve errors status for an asynchronous I/O operation (**REALTIME**)3949 **SYNOPSIS**

3950 AIO #include <aio.h>

3951 int aio_error(const struct aiocb *aiocbp);

3952

3953 **DESCRIPTION**

3954 The *aio_error()* function shall return the error status associated with the **aiocb** structure
 3955 referenced by the *aiocbp* argument. The error status for an asynchronous I/O operation is the
 3956 SIO *errno* value that would be set by the corresponding *read()*, *write()*, *fdatasync()*, or *fsync()*
 3957 operation. If the operation has not yet completed, then the error status shall be equal to
 3958 [EINPROGRESS].

3959 **RETURN VALUE**

3960 If the asynchronous I/O operation has completed successfully, then 0 shall be returned. If the
 3961 asynchronous operation has completed unsuccessfully, then the error status, as described for
 3962 SIO *read()*, *write()*, *fdatasync()*, and *fsync()*, shall be returned. If the asynchronous I/O operation has
 3963 not yet completed, then [EINPROGRESS] shall be returned.

3964 **ERRORS**3965 The *aio_error()* function may fail if:

3966 [EINVAL] The *aiocbp* argument does not refer to an asynchronous operation whose
 3967 return status has not yet been retrieved.

3968 **EXAMPLES**

3969 None.

3970 **APPLICATION USAGE**

3971 The *aio_error()* function is part of the Asynchronous Input and Output option and need not be
 3972 available on all implementations.

3973 **RATIONALE**

3974 None.

3975 **FUTURE DIRECTIONS**

3976 None.

3977 **SEE ALSO**

3978 *aio_cancel()*, *aio_fsync()*, *aio_read()*, *aio_return()*, *aio_write()*, *close()*, *exec*, *exit()*, *fork()*, *lio_listio()*,
 3979 *lseek()*, *read()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**aio.h**>

3980 **CHANGE HISTORY**

3981 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

3982 **Issue 6**

3983 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 3984 implementation does not support the Asynchronous Input and Output option.

3985 The APPLICATION USAGE section is added.

3986 **NAME**3987 aio_fsync — asynchronous file synchronization (**REALTIME**)3988 **SYNOPSIS**

3989 AIO #include <aio.h>

3990 int aio_fsync(int op, struct aiocb *aiocbp);

3991

3992 **DESCRIPTION**

3993 The *aio_fsync()* function shall asynchronously force all I/O operations associated with the file
 3994 indicated by the file descriptor *aio_fildes* member of the **aiocb** structure referenced by the *aiocbp*
 3995 argument and queued at the time of the call to *aio_fsync()* to the synchronized I/O completion
 3996 state. The function call shall return when the synchronization request has been initiated or
 3997 queued to the file or device (even when the data cannot be synchronized immediately).

3998 If *op* is O_DSYNC, all currently queued I/O operations shall be completed as if by a call to
 3999 *fdatasync()*; that is, as defined for synchronized I/O data integrity completion. If *op* is O_SYNC,
 4000 all currently queued I/O operations shall be completed as if by a call to *fsync()*; that is, as
 4001 defined for synchronized I/O file integrity completion. If the *aio_fsync()* function fails, or if the
 4002 operation queued by *aio_fsync()* fails, then, as for *fsync()* and *fdatasync()*, outstanding I/O
 4003 operations are not guaranteed to have been completed.

4004 If *aio_fsync()* succeeds, then it is only the I/O that was queued at the time of the call to
 4005 *aio_fsync()* that is guaranteed to be forced to the relevant completion state. The completion of
 4006 subsequent I/O on the file descriptor is not guaranteed to be completed in a synchronized
 4007 fashion.

4008 The *aiocbp* argument refers to an asynchronous I/O control block. The *aiocbp* value may be used
 4009 as an argument to *aio_error()* and *aio_return()* in order to determine the error status and return
 4010 status, respectively, of the asynchronous operation while it is proceeding. When the request is
 4011 queued, the error status for the operation is [EINPROGRESS]. When all data has been
 4012 successfully transferred, the error status shall be reset to reflect the success or failure of the
 4013 operation. If the operation does not complete successfully, the error status for the operation shall
 4014 be set to indicate the error. The *aio_sigevent* member determines the asynchronous notification to
 4015 occur as specified in Section 2.4.1 (on page 28) when all operations have achieved synchronized
 4016 I/O completion. All other members of the structure referenced by *aiocbp* are ignored. If the
 4017 control block referenced by *aiocbp* becomes an illegal address prior to asynchronous I/O
 4018 completion, then the behavior is undefined.

4019 If the *aio_fsync()* function fails or *aiocbp* indicates an error condition, data is not guaranteed to
 4020 have been successfully transferred.

4021 **RETURN VALUE**

4022 The *aio_fsync()* function shall return the value 0 to the calling process if the I/O operation is
 4023 successfully queued; otherwise, the function shall return the value -1 and set *errno* to indicate
 4024 the error.

4025 **ERRORS**4026 The *aio_fsync()* function shall fail if:

4027 [EAGAIN] The requested asynchronous operation was not queued due to temporary
 4028 resource limitations.

4029 [EBADF] The *aio_fildes* member of the **aiocb** structure referenced by the *aiocbp* argument
 4030 is not a valid file descriptor open for writing.

- 4031 [EINVAL] This implementation does not support synchronized I/O for this file.
- 4032 [EINVAL] A value of *op* other than O_DSYNC or O_SYNC was specified.
- 4033 In the event that any of the queued I/O operations fail, *aio_fsync()* shall return the error
4034 condition defined for *read()* and *write()*. The error is returned in the error status for the
4035 asynchronous *fsync()* operation, which can be retrieved using *aio_error()*.
- 4036 **EXAMPLES**
- 4037 None.
- 4038 **APPLICATION USAGE**
- 4039 The *aio_fsync()* function is part of the Asynchronous Input and Output option and need not be
4040 available on all implementations.
- 4041 **RATIONALE**
- 4042 None.
- 4043 **FUTURE DIRECTIONS**
- 4044 None.
- 4045 **SEE ALSO**
- 4046 *fcntl()*, *fdatasync()*, *fsync()*, *open()*, *read()*, *write()*, the Base Definitions volume of
4047 IEEE Std 1003.1-2001, <**aio.h**>
- 4048 **CHANGE HISTORY**
- 4049 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
- 4050 **Issue 6**
- 4051 The [ENOSYS] error condition has been removed as stubs need not be provided if an
4052 implementation does not support the Asynchronous Input and Output option.
- 4053 The APPLICATION USAGE section is added.

4054 **NAME**4055 aio_read — asynchronous read from a file (**REALTIME**)4056 **SYNOPSIS**

4057 AIO #include <aio.h>

4058 int aio_read(struct aiocb *aiocbp);

4059

4060 **DESCRIPTION**

4061 The *aio_read()* function shall read *aiocbp->aio_nbytes* from the file associated with
 4062 *aiocbp->aio_fildes* into the buffer pointed to by *aiocbp->aio_buf*. The function call shall return when
 4063 the read request has been initiated or queued to the file or device (even when the data cannot be
 4064 delivered immediately).

4065 PIO If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted
 4066 at a priority equal to the scheduling priority of the process minus *aiocbp->aio_reqprio*.

4067 The *aiocbp* value may be used as an argument to *aio_error()* and *aio_return()* in order to
 4068 determine the error status and return status, respectively, of the asynchronous operation while it
 4069 is proceeding. If an error condition is encountered during queuing, the function call shall return
 4070 without having initiated or queued the request. The requested operation takes place at the
 4071 absolute position in the file as given by *aio_offset*, as if *lseek()* were called immediately prior to
 4072 the operation with an *offset* equal to *aio_offset* and a *whence* equal to *SEEK_SET*. After a
 4073 successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file
 4074 is unspecified.

4075 The *aiocbp->aio_lio_opcode* field shall be ignored by *aio_read()*.

4076 The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp->aio_buf* or
 4077 the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O
 4078 completion, then the behavior is undefined.

4079 Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

4080 SIO If synchronized I/O is enabled on the file associated with *aiocbp->aio_fildes*, the behavior of this
 4081 function shall be according to the definitions of synchronized I/O data integrity completion and
 4082 synchronized I/O file integrity completion.

4083 For any system action that changes the process memory space while an asynchronous I/O is
 4084 outstanding to the address range being changed, the result of that action is undefined.

4085 For regular files, no data transfer shall occur past the offset maximum established in the open
 4086 file description associated with *aiocbp->aio_fildes*.

4087 **RETURN VALUE**

4088 The *aio_read()* function shall return the value zero to the calling process if the I/O operation is
 4089 successfully queued; otherwise, the function shall return the value -1 and set *errno* to indicate
 4090 the error.

4091 **ERRORS**

4092 The *aio_read()* function shall fail if:

4093 [EAGAIN] The requested asynchronous I/O operation was not queued due to system
 4094 resource limitations.

4095 Each of the following conditions may be detected synchronously at the time of the call to
 4096 *aio_read()*, or asynchronously. If any of the conditions below are detected synchronously, the
 4097 *aio_read()* function shall return -1 and set *errno* to the corresponding value. If any of the
 4098 conditions below are detected asynchronously, the return status of the asynchronous operation

- 4099 is set to `-1`, and the error status of the asynchronous operation is set to the corresponding value.
- 4100 [EBADF] The `aiocbp->aio_fildes` argument is not a valid file descriptor open for reading.
- 4101 [EINVAL] The file offset value implied by `aiocbp->aio_offset` would be invalid,
4102 `aiocbp->aio_reqprio` is not a valid value, or `aiocbp->aio_nbytes` is an invalid
4103 value.
- 4104 In the case that the `aio_read()` successfully queues the I/O operation but the operation is
4105 subsequently canceled or encounters an error, the return status of the asynchronous operation is
4106 one of the values normally returned by the `read()` function call. In addition, the error status of
4107 the asynchronous operation is set to one of the error statuses normally set by the `read()` function
4108 call, or one of the following values:
- 4109 [EBADF] The `aiocbp->aio_fildes` argument is not a valid file descriptor open for reading.
- 4110 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit
4111 `aio_cancel()` request.
- 4112 [EINVAL] The file offset value implied by `aiocbp->aio_offset` would be invalid.
- 4113 The following condition may be detected synchronously or asynchronously:
- 4114 [EOVERFLOW] The file is a regular file, `aiocbp->aio_nbytes` is greater than 0, and the starting
4115 offset in `aiocbp->aio_offset` is before the end-of-file and is at or beyond the offset
4116 maximum in the open file description associated with `aiocbp->aio_fildes`.

4117 EXAMPLES

4118 None.

4119 APPLICATION USAGE

4120 The `aio_read()` function is part of the Asynchronous Input and Output option and need not be
4121 available on all implementations.

4122 RATIONALE

4123 None.

4124 FUTURE DIRECTIONS

4125 None.

4126 SEE ALSO

4127 `aio_cancel()`, `aio_error()`, `lio_listio()`, `aio_return()`, `aio_write()`, `close()`, `exec`, `exit()`, `fork()`, `lseek()`,
4128 `read()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<aio.h>`

4129 CHANGE HISTORY

4130 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4131 Large File Summit extensions are added.

4132 Issue 6

4133 The [ENOSYS] error condition has been removed as stubs need not be provided if an
4134 implementation does not support the Asynchronous Input and Output option.

4135 The APPLICATION USAGE section is added.

4136 The following new requirements on POSIX implementations derive from alignment with the
4137 Single UNIX Specification:

- 4138 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file
4139 description. This change is to support large files.
- 4140 • In the ERRORS section, the [Eoverflow] condition is added. This change is to support
4141 large files.

4142 **NAME**4143 aio_return — retrieve return status of an asynchronous I/O operation (**REALTIME**)4144 **SYNOPSIS**

4145 AIO #include <aio.h>

4146 ssize_t aio_return(struct aiocb *aiocbp);

4147

4148 **DESCRIPTION**

4149 The *aio_return()* function shall return the return status associated with the **aiocb** structure
 4150 referenced by the *aiocbp* argument. The return status for an asynchronous I/O operation is the
 4151 value that would be returned by the corresponding *read()*, *write()*, or *fsync()* function call. If the
 4152 error status for the operation is equal to [EINPROGRESS], then the return status for the
 4153 operation is undefined. The *aio_return()* function may be called exactly once to retrieve the
 4154 return status of a given asynchronous operation; thereafter, if the same **aiocb** structure is used in
 4155 a call to *aio_return()* or *aio_error()*, an error may be returned. When the **aiocb** structure referred
 4156 to by *aiocbp* is used to submit another asynchronous operation, then *aio_return()* may be
 4157 successfully used to retrieve the return status of that operation.

4158 **RETURN VALUE**

4159 If the asynchronous I/O operation has completed, then the return status, as described for *read()*,
 4160 *write()*, and *fsync()*, shall be returned. If the asynchronous I/O operation has not yet completed,
 4161 the results of *aio_return()* are undefined.

4162 **ERRORS**4163 The *aio_return()* function may fail if:

4164 [EINVAL] The *aiocbp* argument does not refer to an asynchronous operation whose
 4165 return status has not yet been retrieved.

4166 **EXAMPLES**

4167 None.

4168 **APPLICATION USAGE**

4169 The *aio_return()* function is part of the Asynchronous Input and Output option and need not be
 4170 available on all implementations.

4171 **RATIONALE**

4172 None.

4173 **FUTURE DIRECTIONS**

4174 None.

4175 **SEE ALSO**

4176 *aio_cancel()*, *aio_error()*, *aio_fsync()*, *aio_read()*, *aio_write()*, *close()*, *exec*, *exit()*, *fork()*, *lio_listio()*,
 4177 *lseek()*, *read()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**aio.h**>

4178 **CHANGE HISTORY**

4179 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4180 **Issue 6**

4181 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 4182 implementation does not support the Asynchronous Input and Output option.

4183 The APPLICATION USAGE section is added.

4184 The [EINVAL] error condition is updated as a “may fail”. This is for consistency with the
 4185 DESCRIPTION.

4186 NAME

4187 aio_suspend — wait for an asynchronous I/O request (**REALTIME**)

4188 SYNOPSIS

4189 AIO #include <aio.h>

```
4190 int aio_suspend(const struct aiocb * const list[], int nent,
4191               const struct timespec *timeout);
4192
```

4193 DESCRIPTION

4194 The *aio_suspend()* function shall suspend the calling thread until at least one of the asynchronous
 4195 I/O operations referenced by the *list* argument has completed, until a signal interrupts the
 4196 function, or, if *timeout* is not NULL, until the time interval specified by *timeout* has passed. If any
 4197 of the **aiocb** structures in the list correspond to completed asynchronous I/O operations (that is,
 4198 the error status for the operation is not equal to [EINPROGRESS]) at the time of the call, the
 4199 function shall return without suspending the calling thread. The *list* argument is an array of
 4200 pointers to asynchronous I/O control blocks. The *nent* argument indicates the number of
 4201 elements in the array. Each **aiocb** structure pointed to has been used in initiating an
 4202 asynchronous I/O request via *aio_read()*, *aio_write()*, or *lio_listio()*. This array may contain
 4203 NULL pointers, which are ignored. If this array contains pointers that refer to **aiocb** structures
 4204 that have not been used in submitting asynchronous I/O, the effect is undefined.

4205 If the time interval indicated in the **timespec** structure pointed to by *timeout* passes before any of
 4206 the I/O operations referenced by *list* are completed, then *aio_suspend()* shall return with an
 4207 error. If the Monotonic Clock option is supported, the clock that shall be used to measure this
 4208 time interval shall be the CLOCK_MONOTONIC clock.

4209 RETURN VALUE

4210 If the *aio_suspend()* function returns after one or more asynchronous I/O operations have
 4211 completed, the function shall return zero. Otherwise, the function shall return a value of -1 and
 4212 set *errno* to indicate the error.

4213 The application may determine which asynchronous I/O completed by scanning the associated
 4214 error and return status using *aio_error()* and *aio_return()*, respectively.

4215 ERRORS

4216 The *aio_suspend()* function shall fail if:

4217 [EAGAIN] No asynchronous I/O indicated in the list referenced by *list* completed in the
 4218 time interval indicated by *timeout*.

4219 [EINTR] A signal interrupted the *aio_suspend()* function. Note that, since each
 4220 asynchronous I/O operation may possibly provoke a signal when it
 4221 completes, this error return may be caused by the completion of one (or more)
 4222 of the very I/O operations being awaited.

4223 EXAMPLES

4224 None.

4225 APPLICATION USAGE

4226 The *aio_suspend()* function is part of the Asynchronous Input and Output option and need not
 4227 be available on all implementations.

4228 RATIONALE

4229 None.

4230 **FUTURE DIRECTIONS**

4231 None.

4232 **SEE ALSO**4233 *aio_read()*, *aio_write()*, *lio_listio()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**aio.h**>4234 **CHANGE HISTORY**

4235 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4236 **Issue 6**4237 The [ENOSYS] error condition has been removed as stubs need not be provided if an
4238 implementation does not support the Asynchronous Input and Output option.

4239 The APPLICATION USAGE section is added.

4240 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the
4241 CLOCK_MONOTONIC clock, if supported, is used.

4242 NAME

4243 aio_write — asynchronous write to a file (**REALTIME**)

4244 SYNOPSIS

4245 AIO #include <aio.h>

4246 int aio_write(struct aiocb *aiocbp);

4247

4248 DESCRIPTION

4249 The *aio_write()* function shall write *aiocbp->aio_nbytes* to the file associated with *aiocbp->aio_fildes*
 4250 from the buffer pointed to by *aiocbp->aio_buf*. The function shall return when the write request
 4251 has been initiated or, at a minimum, queued to the file or device.

4252 PIO If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted
 4253 at a priority equal to the scheduling priority of the process minus *aiocbp->aio_reqprio*.

4254 The *aiocbp* argument may be used as an argument to *aio_error()* and *aio_return()* in order to
 4255 determine the error status and return status, respectively, of the asynchronous operation while it
 4256 is proceeding.

4257 The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp->aio_buf* or
 4258 the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O
 4259 completion, then the behavior is undefined.

4260 If **O_APPEND** is not set for the file descriptor *aio_fildes*, then the requested operation shall take
 4261 place at the absolute position in the file as given by *aio_offset*, as if *lseek()* were called
 4262 immediately prior to the operation with an *offset* equal to *aio_offset* and a *whence* equal to
 4263 **SEEK_SET**. If **O_APPEND** is set for the file descriptor, write operations append to the file in the
 4264 same order as the calls were made. After a successful call to enqueue an asynchronous I/O
 4265 operation, the value of the file offset for the file is unspecified.

4266 The *aiocbp->aio_lio_opcode* field shall be ignored by *aio_write()*.

4267 Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

4268 SIO If synchronized I/O is enabled on the file associated with *aiocbp->aio_fildes*, the behavior of this
 4269 function shall be according to the definitions of synchronized I/O data integrity completion, and
 4270 synchronized I/O file integrity completion.

4271 For any system action that changes the process memory space while an asynchronous I/O is
 4272 outstanding to the address range being changed, the result of that action is undefined.

4273 For regular files, no data transfer shall occur past the offset maximum established in the open
 4274 file description associated with *aiocbp->aio_fildes*.

4275 RETURN VALUE

4276 The *aio_write()* function shall return the value zero to the calling process if the I/O operation is
 4277 successfully queued; otherwise, the function shall return the value -1 and set *errno* to indicate
 4278 the error.

4279 ERRORS

4280 The *aio_write()* function shall fail if:

4281 [EAGAIN] The requested asynchronous I/O operation was not queued due to system
 4282 resource limitations.

4283 Each of the following conditions may be detected synchronously at the time of the call to
 4284 *aio_write()*, or asynchronously. If any of the conditions below are detected synchronously, the
 4285 *aio_write()* function shall return -1 and set *errno* to the corresponding value. If any of the

4286 conditions below are detected asynchronously, the return status of the asynchronous operation
 4287 shall be set to -1, and the error status of the asynchronous operation is set to the corresponding
 4288 value.

4289 [EBADF] The *aiocbp->aio_fildes* argument is not a valid file descriptor open for writing.

4290 [EINVAL] The file offset value implied by *aiocbp->aio_offset* would be invalid,
 4291 *aiocbp->aio_reqprio* is not a valid value, or *aiocbp->aio_nbytes* is an invalid
 4292 value.

4293 In the case that the *aio_write()* successfully queues the I/O operation, the return status of the
 4294 asynchronous operation shall be one of the values normally returned by the *write()* function call.
 4295 If the operation is successfully queued but is subsequently canceled or encounters an error, the
 4296 error status for the asynchronous operation contains one of the values normally set by the
 4297 *write()* function call, or one of the following:

4298 [EBADF] The *aiocbp->aio_fildes* argument is not a valid file descriptor open for writing.

4299 [EINVAL] The file offset value implied by *aiocbp->aio_offset* would be invalid.

4300 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit
 4301 *aio_cancel()* request.

4302 The following condition may be detected synchronously or asynchronously:

4303 [EFBIG] The file is a regular file, *aiocbp->aio_nbytes* is greater than 0, and the starting
 4304 offset in *aiocbp->aio_offset* is at or beyond the offset maximum in the open file
 4305 description associated with *aiocbp->aio_fildes*.

4306 EXAMPLES

4307 None.

4308 APPLICATION USAGE

4309 The *aio_write()* function is part of the Asynchronous Input and Output option and need not be
 4310 available on all implementations.

4311 RATIONALE

4312 None.

4313 FUTURE DIRECTIONS

4314 None.

4315 SEE ALSO

4316 *aio_cancel()*, *aio_error()*, *aio_read()*, *aio_return()*, *close()*, *exec*, *exit()*, *fork()*, *lio_listio()*, *lseek()*,
 4317 *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**aio.h**>

4318 CHANGE HISTORY

4319 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4320 Large File Summit extensions are added.

4321 Issue 6

4322 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 4323 implementation does not support the Asynchronous Input and Output option.

4324 The APPLICATION USAGE section is added.

4325 The following new requirements on POSIX implementations derive from alignment with the
 4326 Single UNIX Specification:

- 4327 • In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs
 4328 past the offset maximum established in the open file description associated with

4329

aiocbp->aiowrites.

4330

- The [EFBIG] error is added as part of the large file support extensions.

4331 **NAME**

4332 alarm — schedule an alarm signal

4333 **SYNOPSIS**

4334 #include <unistd.h>

4335 unsigned alarm(unsigned *seconds*);4336 **DESCRIPTION**

4337 The *alarm()* function shall cause the system to generate a SIGALRM signal for the process after
4338 the number of realtime seconds specified by *seconds* have elapsed. Processor scheduling delays
4339 may prevent the process from handling the signal as soon as it is generated.

4340 If *seconds* is 0, a pending alarm request, if any, is canceled.

4341 Alarm requests are not stacked; only one SIGALRM generation can be scheduled in this manner.
4342 If the SIGALRM signal has not yet been generated, the call shall result in rescheduling the time
4343 at which the SIGALRM signal is generated.

4344 XSI Interactions between *alarm()* and any of *setitimer()*, *ualarm()*, or *usleep()* are unspecified.

4345 **RETURN VALUE**

4346 If there is a previous *alarm()* request with time remaining, *alarm()* shall return a non-zero value
4347 that is the number of seconds until the previous request would have generated a SIGALRM
4348 signal. Otherwise, *alarm()* shall return 0.

4349 **ERRORS**

4350 The *alarm()* function is always successful, and no return value is reserved to indicate an error.

4351 **EXAMPLES**

4352 None.

4353 **APPLICATION USAGE**

4354 The *fork()* function clears pending alarms in the child process. A new process image created by
4355 one of the *exec* functions inherits the time left to an alarm signal in the old process' image.

4356 Application writers should note that the type of the argument *seconds* and the return value of
4357 *alarm()* is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces
4358 Application cannot pass a value greater than the minimum guaranteed value for {UINT_MAX},
4359 which the ISO C standard sets as 65 535, and any application passing a larger value is restricting
4360 its portability. A different type was considered, but historical implementations, including those
4361 with a 16-bit **int** type, consistently use either **unsigned** or **int**.

4362 Application writers should be aware of possible interactions when the same process uses both
4363 the *alarm()* and *sleep()* functions.

4364 **RATIONALE**

4365 Many historical implementations (including Version 7 and System V) allow an alarm to occur up
4366 to a second early. Other implementations allow alarms up to half a second or one clock tick
4367 early or do not allow them to occur early at all. The latter is considered most appropriate, since it
4368 gives the most predictable behavior, especially since the signal can always be delayed for an
4369 indefinite amount of time due to scheduling. Applications can thus choose the *seconds* argument
4370 as the minimum amount of time they wish to have elapse before the signal.

4371 The term “realtime” here and elsewhere (*sleep()*, *times()*) is intended to mean “wall clock” time
4372 as common English usage, and has nothing to do with “realtime operating systems”. It is in
4373 contrast to *virtual time*, which could be misinterpreted if just *time* were used.

4374 In some implementations, including 4.3 BSD, very large values of the *seconds* argument are
4375 silently rounded down to an implementation-defined maximum value. This maximum is large

4376 enough (to the order of several months) that the effect is not noticeable.

4377 There were two possible choices for alarm generation in multi-threaded applications: generation
4378 for the calling thread or generation for the process. The first option would not have been
4379 particularly useful since the alarm state is maintained on a per-process basis and the alarm that
4380 is established by the last invocation of *alarm()* is the only one that would be active.

4381 Furthermore, allowing generation of an asynchronous signal for a thread would have introduced
4382 an exception to the overall signal model. This requires a compelling reason in order to be
4383 justified.

4384 **FUTURE DIRECTIONS**

4385 None.

4386 **SEE ALSO**

4387 *alarm()*, *exec*, *fork()*, *getitimer()*, *pause()*, *sigaction()*, *sleep()*, *ualarm()*, *usleep()*, the Base
4388 Definitions volume of IEEE Std 1003.1-2001, <signal.h>, <unistd.h>

4389 **CHANGE HISTORY**

4390 First released in Issue 1. Derived from Issue 1 of the SVID.

4391 **Issue 6**

4392 The following new requirements on POSIX implementations derive from alignment with the
4393 Single UNIX Specification:

- 4394 • The DESCRIPTION is updated to indicate that interactions with the *setitimer()*, *ualarm()*, and
4395 *usleep()* functions are unspecified.

4396 NAME

4397 asctime, asctime_r — convert date and time to a string

4398 SYNOPSIS

4399 #include <time.h>

4400 char *asctime(const struct tm *timeptr);

4401 TSF char *asctime_r(const struct tm *restrict tm, char *restrict buf);

4402

4403 DESCRIPTION

4404 CX For *asctime()*: The functionality described on this reference page is aligned with the ISO C
 4405 standard. Any conflict between the requirements described here and the ISO C standard is
 4406 unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

4407 The *asctime()* function shall convert the broken-down time in the structure pointed to by *timeptr*
 4408 into a string in the form:

4409 Sun Sep 16 01:03:52 1973\n\0

4410 using the equivalent of the following algorithm:

```

4411 char *asctime(const struct tm *timeptr)
4412 {
4413     static char wday_name[7][3] = {
4414         "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
4415     };
4416     static char mon_name[12][3] = {
4417         "Jan", "Feb", "Mar", "Apr", "May", "Jun",
4418         "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
4419     };
4420     static char result[26];
4421     sprintf(result, "%.3s %.3s%3d %.2d:%.2d:%.2d %d\n",
4422             wday_name[timeptr->tm_wday],
4423             mon_name[timeptr->tm_mon],
4424             timeptr->tm_mday, timeptr->tm_hour,
4425             timeptr->tm_min, timeptr->tm_sec,
4426             1900 + timeptr->tm_year);
4427     return result;
4428 }
```

4429 The **tm** structure is defined in the <time.h> header.

4430 CX The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static
 4431 objects: a broken-down time structure and an array of type **char**. Execution of any of the
 4432 functions may overwrite the information returned in either of these objects by any of the other
 4433 functions.

4434 The *asctime()* function need not be reentrant. A function that is not required to be reentrant is not
 4435 required to be thread-safe.

4436 TSF The *asctime_r()* function shall convert the broken-down time in the structure pointed to by *tm*
 4437 into a string (of the same form as that returned by *asctime()*) that is placed in the user-supplied
 4438 buffer pointed to by *buf* (which shall contain at least 26 bytes) and then return *buf*.

4439 **RETURN VALUE**

4440 Upon successful completion, *asctime()* shall return a pointer to the string.

4441 TSF Upon successful completion, *asctime_r()* shall return a pointer to a character string containing
4442 the date and time. This string is pointed to by the argument *buf*. If the function is unsuccessful,
4443 it shall return NULL.

4444 **ERRORS**

4445 No errors are defined.

4446 **EXAMPLES**

4447 None.

4448 **APPLICATION USAGE**

4449 Values for the broken-down time structure can be obtained by calling *gmtime()* or *localtime()*.
4450 This function is included for compatibility with older implementations, and does not support
4451 localized date and time formats. Applications should use *strptime()* to achieve maximum
4452 portability.

4453 The *asctime_r()* function is thread-safe and shall return values in a user-supplied buffer instead
4454 of possibly using a static data area that may be overwritten by each call.

4455 **RATIONALE**

4456 None.

4457 **FUTURE DIRECTIONS**

4458 None.

4459 **SEE ALSO**

4460 *clock()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strptime()*, *strptime()*, *time()*, *utime()*,
4461 the Base Definitions volume of IEEE Std 1003.1-2001, <**time.h**>

4462 **CHANGE HISTORY**

4463 First released in Issue 1. Derived from Issue 1 of the SVID.

4464 **Issue 5**

4465 Normative text previously in the APPLICATION USAGE section is moved to the
4466 DESCRIPTION.

4467 The *asctime_r()* function is included for alignment with the POSIX Threads Extension.

4468 A note indicating that the *asctime()* function need not be reentrant is added to the
4469 DESCRIPTION.

4470 **Issue 6**

4471 The *asctime_r()* function is marked as part of the Thread-Safe Functions option.

4472 Extensions beyond the ISO C standard are marked.

4473 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
4474 its avoidance of possibly using a static data area.

4475 The DESCRIPTION of *asctime_r()* is updated to describe the format of the string returned.

4476 The **restrict** keyword is added to the *asctime_r()* prototype for alignment with the
4477 ISO/IEC 9899:1999 standard.

4478 **NAME**

4479 asin, asinf, asinl — arc sine function

4480 **SYNOPSIS**

4481 #include <math.h>

4482 double asin(double x);

4483 float asinf(float x);

4484 long double asinl(long double x);

4485 **DESCRIPTION**

4486 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 4487 conflict between the requirements described here and the ISO C standard is unintentional. This
 4488 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

4489 These functions shall compute the principal value of the arc sine of their argument *x*. The value
 4490 of *x* should be in the range $[-1,1]$.

4491 An application wishing to check for error situations should set *errno* to zero and call
 4492 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 4493 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 4494 zero, an error has occurred.

4495 **RETURN VALUE**

4496 Upon successful completion, these functions shall return the arc sine of *x*, in the range
 4497 $[-\pi/2, \pi/2]$ radians.

4498 **MX** For finite values of *x* not in the range $[-1,1]$, a domain error shall occur, and either a NaN (if
 4499 supported), or an implementation-defined value shall be returned.

4500 **MX** If *x* is NaN, a NaN shall be returned.

4501 If *x* is ± 0 , *x* shall be returned.

4502 If *x* is $\pm \text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 4503 defined value shall be returned.

4504 If *x* is subnormal, a range error may occur and *x* should be returned.

4505 **ERRORS**

4506 These functions shall fail if:

4507 **MX** **Domain Error** The *x* argument is finite and is not in the range $[-1,1]$, or is $\pm \text{Inf}$.

4508 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 4509 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 4510 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 4511 shall be raised.

4512 These functions may fail if:

4513 **MX** **Range Error** The value of *x* is subnormal.

4514 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 4515 then *errno* shall be set to [ERANGE]. If the integer expression
 4516 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 4517 floating-point exception shall be raised.

4518 **EXAMPLES**

4519 None.

4520 **APPLICATION USAGE**

4521 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`
4522 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

4523 **RATIONALE**

4524 None.

4525 **FUTURE DIRECTIONS**

4526 None.

4527 **SEE ALSO**

4528 *feclearexcept()*, *fetestexcept()*, *isnan()*, *sin()*, the Base Definitions volume of IEEE Std 1003.1-2001,
4529 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <**math.h**>

4530 **CHANGE HISTORY**

4531 First released in Issue 1. Derived from Issue 1 of the SVID.

4532 **Issue 5**

4533 The DESCRIPTION is updated to indicate how an application should check for an error. This
4534 text was previously published in the APPLICATION USAGE section.

4535 **Issue 6**4536 The *asinf()* and *asinl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

4537 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
4538 revised to align with the ISO/IEC 9899:1999 standard.

4539 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
4540 marked.

4541 **NAME**

4542 asinh, asinhf, asinhl — inverse hyperbolic sine functions

4543 **SYNOPSIS**

4544 #include <math.h>

4545 double asinh(double x);

4546 float asinhf(float x);

4547 long double asinhl(long double x);

4548 **DESCRIPTION**4549 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
4550 conflict between the requirements described here and the ISO C standard is unintentional. This
4551 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.4552 These functions shall compute the inverse hyperbolic sine of their argument *x*.4553 An application wishing to check for error situations should set *errno* to zero and call
4554 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
4555 *fetetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
4556 zero, an error has occurred.4557 **RETURN VALUE**4558 Upon successful completion, these functions shall return the inverse hyperbolic sine of their
4559 argument.4560 **MX** If *x* is NaN, a NaN shall be returned.4561 If *x* is ± 0 , or $\pm \text{Inf}$, *x* shall be returned.4562 If *x* is subnormal, a range error may occur and *x* should be returned.4563 **ERRORS**

4564 These functions may fail if:

4565 **MX** **Range Error** The value of *x* is subnormal.4566 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
4567 then *errno* shall be set to [ERANGE]. If the integer expression
4568 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
4569 floating-point exception shall be raised.4570 **EXAMPLES**

4571 None.

4572 **APPLICATION USAGE**4573 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
4574 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.4575 **RATIONALE**

4576 None.

4577 **FUTURE DIRECTIONS**

4578 None.

4579 **SEE ALSO**4580 *feclearexcept*(), *fetetestexcept*(), *sinh*(), the Base Definitions volume of IEEE Std 1003.1-2001, Section
4581 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

4582 **CHANGE HISTORY**

4583 First released in Issue 4, Version 2.

4584 **Issue 5**

4585 Moved from X/OPEN UNIX extension to BASE.

4586 **Issue 6**

4587 The *asinh()* function is no longer marked as an extension.

4588 The *asinhf()* and *asinhll()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

4590 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

4592 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

4594 **NAME**

4595 *asinl* — arc sine function

4596 **SYNOPSIS**

4597 #include <math.h>

4598 long double *asinl*(long double *x*);

4599 **DESCRIPTION**

4600 Refer to *asin*().

4601 **NAME**

4602 assert — insert program diagnostics

4603 **SYNOPSIS**

4604 #include <assert.h>

4605 void assert(*scalar expression*);4606 **DESCRIPTION**4607 **cx** The functionality described on this reference page is aligned with the ISO C standard. Any
4608 conflict between the requirements described here and the ISO C standard is unintentional. This
4609 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.4610 The *assert()* macro shall insert diagnostics into programs; it shall expand to a **void** expression.
4611 When it is executed, if *expression* (which shall have a **scalar** type) is false (that is, compares equal
4612 to 0), *assert()* shall write information about the particular call that failed on *stderr* and shall call
4613 *abort()*.4614 The information written about the call that failed shall include the text of the argument, the
4615 name of the source file, the source file line number, and the name of the enclosing function; the
4616 latter are, respectively, the values of the preprocessing macros `__FILE__` and `__LINE__` and of
4617 the identifier `__func__`.4618 Forcing a definition of the name `NDEBUG`, either from the compiler command line or with the
4619 preprocessor control statement `#define NDEBUG` ahead of the `#include <assert.h>` statement,
4620 shall stop assertions from being compiled into the program.4621 **RETURN VALUE**4622 The *assert()* macro shall not return a value.4623 **ERRORS**

4624 No errors are defined.

4625 **EXAMPLES**

4626 None.

4627 **APPLICATION USAGE**

4628 None.

4629 **RATIONALE**

4630 None.

4631 **FUTURE DIRECTIONS**

4632 None.

4633 **SEE ALSO**4634 *abort()*, *stderr*, the Base Definitions volume of IEEE Std 1003.1-2001, `<assert.h>`4635 **CHANGE HISTORY**

4636 First released in Issue 1. Derived from Issue 1 of the SVID.

4637 **Issue 6**4638 The prototype for the *expression* argument to *assert()* is changed from **int** to **scalar** for alignment
4639 with the ISO/IEC 9899:1999 standard.4640 The DESCRIPTION of *assert()* is updated for alignment with the ISO/IEC 9899:1999 standard.

4641 **NAME**

4642 atan, atanf, atanl — arc tangent function

4643 **SYNOPSIS**

4644 #include <math.h>

4645 double atan(double x);

4646 float atanf(float x);

4647 long double atanl(long double x);

4648 **DESCRIPTION**

4649 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 4650 conflict between the requirements described here and the ISO C standard is unintentional. This
 4651 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

4652 These functions shall compute the principal value of the arc tangent of their argument x .

4653 An application wishing to check for error situations should set *errno* to zero and call
 4654 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 4655 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 4656 zero, an error has occurred.

4657 **RETURN VALUE**4658 Upon successful completion, these functions shall return the arc tangent of x in the range
4659 $[-\pi/2, \pi/2]$ radians.4660 MX If x is NaN, a NaN shall be returned.4661 If x is ± 0 , x shall be returned.4662 If x is $\pm\text{Inf}$, $\pm\pi/2$ shall be returned.4663 If x is subnormal, a range error may occur and x should be returned.4664 **ERRORS**

4665 These functions may fail if:

4666 MX **Range Error** The value of x is subnormal.

4667 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 4668 then *errno* shall be set to [ERANGE]. If the integer expression
 4669 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 4670 floating-point exception shall be raised.

4671 **EXAMPLES**

4672 None.

4673 **APPLICATION USAGE**4674 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
4675 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.4676 **RATIONALE**

4677 None.

4678 **FUTURE DIRECTIONS**

4679 None.

4680 **SEE ALSO**4681 *atan2()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tan()*, the Base Definitions volume of
4682 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
4683 <math.h>

4684 **CHANGE HISTORY**

4685 First released in Issue 1. Derived from Issue 1 of the SVID.

4686 **Issue 5**

4687 The DESCRIPTION is updated to indicate how an application should check for an error. This
4688 text was previously published in the APPLICATION USAGE section.

4689 **Issue 6**

4690 The *atanf()* and *atanl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

4691 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
4692 revised to align with the ISO/IEC 9899:1999 standard.

4693 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
4694 marked.

4695 **NAME**

4696 atan2, atan2f, atan2l — arc tangent functions

4697 **SYNOPSIS**

4698 #include <math.h>

4699 double atan2(double y, double x);

4700 float atan2f(float y, float x);

4701 long double atan2l(long double y, long double x);

4702 **DESCRIPTION**

4703 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 4704 conflict between the requirements described here and the ISO C standard is unintentional. This
 4705 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

4706 These functions shall compute the principal value of the arc tangent of y/x , using the signs of
 4707 both arguments to determine the quadrant of the return value.

4708 An application wishing to check for error situations should set *errno* to zero and call
 4709 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 4710 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 4711 zero, an error has occurred.

4712 **RETURN VALUE**

4713 Upon successful completion, these functions shall return the arc tangent of y/x in the range
 4714 $[-\pi, \pi]$ radians.

4715 If y is ± 0 and x is < 0 , $\pm\pi$ shall be returned.

4716 If y is ± 0 and x is > 0 , ± 0 shall be returned.

4717 If y is < 0 and x is ± 0 , $-\pi/2$ shall be returned.

4718 If y is > 0 and x is ± 0 , $\pi/2$ shall be returned.

4719 If x is 0, a pole error shall not occur.

4720 **MX** If either x or y is NaN, a NaN shall be returned.

4721 If the result underflows, a range error may occur and y/x should be returned.

4722 If y is ± 0 and x is -0 , $\pm\pi$ shall be returned.

4723 If y is ± 0 and x is $+0$, ± 0 shall be returned.

4724 For finite values of $\pm y > 0$, if x is $-\text{Inf}$, $\pm\pi$ shall be returned.

4725 For finite values of $\pm y > 0$, if x is $+\text{Inf}$, ± 0 shall be returned.

4726 For finite values of x , if y is $\pm\text{Inf}$, $\pm\pi/2$ shall be returned.

4727 If y is $\pm\text{Inf}$ and x is $-\text{Inf}$, $\pm 3\pi/4$ shall be returned.

4728 If y is $\pm\text{Inf}$ and x is $+\text{Inf}$, $\pm\pi/4$ shall be returned.

4729 If both arguments are 0, a domain error shall not occur.

4730 **ERRORS**

4731 These functions may fail if:

4732 **MX** **Range Error** The result underflows.

4733 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 4734 then *errno* shall be set to [ERANGE]. If the integer expression

4735 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
4736 floating-point exception shall be raised.

4737 **EXAMPLES**

4738 None.

4739 **APPLICATION USAGE**

4740 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
4741 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

4742 **RATIONALE**

4743 None.

4744 **FUTURE DIRECTIONS**

4745 None.

4746 **SEE ALSO**

4747 *atan()*, *feclearexcept()*, *fetetestexcept()*, *isnan()*, *tan()*, the Base Definitions volume of
4748 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
4749 <math.h>

4750 **CHANGE HISTORY**

4751 First released in Issue 1. Derived from Issue 1 of the SVID.

4752 **Issue 5**

4753 The DESCRIPTION is updated to indicate how an application should check for an error. This
4754 text was previously published in the APPLICATION USAGE section.

4755 **Issue 6**

4756 The *atan2f()* and *atan2l()* functions are added for alignment with the ISO/IEC 9899:1999
4757 standard.

4758 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
4759 revised to align with the ISO/IEC 9899:1999 standard.

4760 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
4761 marked.

4762 **NAME**

4763 atanf — arc tangent function

4764 **SYNOPSIS**

4765 #include <math.h>

4766 float atanf(float x);

4767 **DESCRIPTION**

4768 Refer to *atan()*.

4769 **NAME**

4770 atanh, atanhf, atanh1 — inverse hyperbolic tangent functions

4771 **SYNOPSIS**

4772 #include <math.h>

4773 double atanh(double x);

4774 float atanhf(float x);

4775 long double atanh1(long double x);

4776 **DESCRIPTION**4777 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
4778 conflict between the requirements described here and the ISO C standard is unintentional. This
4779 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.4780 These functions shall compute the inverse hyperbolic tangent of their argument *x*.4781 An application wishing to check for error situations should set *errno* to zero and call
4782 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
4783 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
4784 zero, an error has occurred.4785 **RETURN VALUE**4786 Upon successful completion, these functions shall return the inverse hyperbolic tangent of their
4787 argument.4788 If *x* is ± 1 , a pole error shall occur, and *atanh()*, *atanhf()*, and *atanh1()* shall return the value of the
4789 macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively, with the same sign as the
4790 correct value of the function.4791 **MX** For finite $|x| > 1$, a domain error shall occur, and either a NaN (if supported), or an
4792 implementation-defined value shall be returned.4793 **MX** If *x* is NaN, a NaN shall be returned.4794 If *x* is ± 0 , *x* shall be returned.4795 If *x* is $\pm \text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
4796 defined value shall be returned.4797 If *x* is subnormal, a range error may occur and *x* should be returned.4798 **ERRORS**

4799 These functions shall fail if:

4800 **MX** Domain Error The *x* argument is finite and not in the range $[-1, 1]$, or is $\pm \text{Inf}$.4801 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
4802 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
4803 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
4804 shall be raised.4805 Pole Error The *x* argument is ± 1 .4806 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
4807 then *errno* shall be set to [ERANGE]. If the integer expression
4808 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the divide-by-
4809 zero floating-point exception shall be raised.

4810 These functions may fail if:

4811 MX **Range Error** The value of x is subnormal.

4812 If the integer expression (`math_errhandling & MATH_ERRNO`) is non-zero,
 4813 then *errno* shall be set to [ERANGE]. If the integer expression
 4814 (`math_errhandling & MATH_ERREXCEPT`) is non-zero, then the underflow
 4815 floating-point exception shall be raised.

4816 **EXAMPLES**

4817 None.

4818 **APPLICATION USAGE**

4819 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`
 4820 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

4821 **RATIONALE**

4822 None.

4823 **FUTURE DIRECTIONS**

4824 None.

4825 **SEE ALSO**

4826 *feclearexcept()*, *fetestexcept()*, *tanh()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section
 4827 4.18, Treatment of Error Conditions for Mathematical Functions, <**math.h**>

4828 **CHANGE HISTORY**

4829 First released in Issue 4, Version 2.

4830 **Issue 5**

4831 Moved from X/OPEN UNIX extension to BASE.

4832 **Issue 6**

4833 The *atanh()* function is no longer marked as an extension.

4834 The *atanhf()* and *atanhl()* functions are added for alignment with the ISO/IEC 9899:1999
 4835 standard.

4836 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 4837 revised to align with the ISO/IEC 9899:1999 standard.

4838 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 4839 marked.

4840 **NAME**4841 **atanl** — arc tangent function4842 **SYNOPSIS**

4843 #include <math.h>

4844 long double atanl(long double x);

4845 **DESCRIPTION**4846 Refer to *atan()*.

4847 **NAME**

4848 atexit — register a function to run at process termination

4849 **SYNOPSIS**

4850 #include <stdlib.h>

4851 int atexit(void (*func)(void));

4852 **DESCRIPTION**4853 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
4854 conflict between the requirements described here and the ISO C standard is unintentional. This
4855 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.4856 The *atexit()* function shall register the function pointed to by *func*, to be called without
4857 arguments at normal program termination. At normal program termination, all functions
4858 registered by the *atexit()* function shall be called, in the reverse order of their registration, except
4859 that a function is called after any previously registered functions that had already been called at
4860 the time it was registered. Normal termination occurs either by a call to *exit()* or a return from
4861 *main()*.4862 At least 32 functions can be registered with *atexit()*.4863 **CX** After a successful call to any of the *exec* functions, any functions previously registered by *atexit()*
4864 shall no longer be registered.4865 **RETURN VALUE**4866 Upon successful completion, *atexit()* shall return 0; otherwise, it shall return a non-zero value.4867 **ERRORS**

4868 No errors are defined.

4869 **EXAMPLES**

4870 None.

4871 **APPLICATION USAGE**4872 The functions registered by a call to *atexit()* must return to ensure that all registered functions
4873 are called.4874 The application should call *sysconf()* to obtain the value of {ATEXIT_MAX}, the number of
4875 functions that can be registered. There is no way for an application to tell how many functions
4876 have already been registered with *atexit()*.4877 **RATIONALE**

4878 None.

4879 **FUTURE DIRECTIONS**

4880 None.

4881 **SEE ALSO**4882 *exit()*, *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>4883 **CHANGE HISTORY**

4884 First released in Issue 4. Derived from the ANSI C standard.

4885 **Issue 6**

4886 Extensions beyond the ISO C standard are marked.

4887 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

4888 **NAME**

4889 atof — convert a string to a double-precision number

4890 **SYNOPSIS**

4891 #include <stdlib.h>

4892 double atof(const char *str);

4893 **DESCRIPTION**

4894 cx The functionality described on this reference page is aligned with the ISO C standard. Any
4895 conflict between the requirements described here and the ISO C standard is unintentional. This
4896 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

4897 The call *atof(str)* shall be equivalent to:4898 strtod(*str*, (char **)NULL),

4899 except that the handling of errors may differ. If the value cannot be represented, the behavior is
4900 undefined.

4901 **RETURN VALUE**4902 The *atof()* function shall return the converted value if the value can be represented.4903 **ERRORS**

4904 No errors are defined.

4905 **EXAMPLES**

4906 None.

4907 **APPLICATION USAGE**

4908 The *atof()* function is subsumed by *strtod()* but is retained because it is used extensively in
4909 existing code. If the number is not known to be in range, *strtod()* should be used because *atof()* is
4910 not required to perform any error checking.

4911 **RATIONALE**

4912 None.

4913 **FUTURE DIRECTIONS**

4914 None.

4915 **SEE ALSO**4916 *strtod()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>4917 **CHANGE HISTORY**

4918 First released in Issue 1. Derived from Issue 1 of the SVID.

4919 **NAME**

4920 atoi — convert a string to an integer

4921 **SYNOPSIS**

4922 #include <stdlib.h>

4923 int atoi(const char *str);

4924 **DESCRIPTION**

4925 cx The functionality described on this reference page is aligned with the ISO C standard. Any
4926 conflict between the requirements described here and the ISO C standard is unintentional. This
4927 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

4928 The call *atoi(str)* shall be equivalent to:

4929 (int) strtol(str, (char **)NULL, 10)

4930 except that the handling of errors may differ. If the value cannot be represented, the behavior is
4931 undefined.4932 **RETURN VALUE**4933 The *atoi()* function shall return the converted value if the value can be represented.4934 **ERRORS**

4935 No errors are defined.

4936 **EXAMPLES**4937 **Converting an Argument**

4938 The following example checks for proper usage of the program. If there is an argument and the
4939 decimal conversion of this argument (obtained using *atoi()*) is greater than 0, then the program
4940 has a valid number of minutes to wait for an event.

```
4941         #include <stdlib.h>
4942         #include <stdio.h>
4943         ...
4944         int minutes_to_event;
4945         ...
4946         if (argc < 2 || ((minutes_to_event = atoi (argv[1]))) <= 0) {
4947             fprintf(stderr, "Usage: %s minutes\n", argv[0]); exit(1);
4948         }
4949         ...
```

4950 **APPLICATION USAGE**

4951 The *atoi()* function is subsumed by *strtol()* but is retained because it is used extensively in
4952 existing code. If the number is not known to be in range, *strtol()* should be used because *atoi()* is
4953 not required to perform any error checking.

4954 **RATIONALE**

4955 None.

4956 **FUTURE DIRECTIONS**

4957 None.

4958 **SEE ALSO**4959 *strtol()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>

4960 **CHANGE HISTORY**

4961 First released in Issue 1. Derived from Issue 1 of the SVID.

4962 **NAME**

4963 atol, atoll — convert a string to a long integer

4964 **SYNOPSIS**

4965 #include <stdlib.h>

4966 long atol(const char *str);

4967 long long atoll(const char *nptr);

4968 **DESCRIPTION**4969 cx The functionality described on this reference page is aligned with the ISO C standard. Any
4970 conflict between the requirements described here and the ISO C standard is unintentional. This
4971 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.4972 The call *atol(str)* shall be equivalent to:

4973 strtol(str, (char **)NULL, 10)

4974 The call *atoll(str)* shall be equivalent to:

4975 strtoll(nptr, (char **)NULL, 10)

4976 except that the handling of errors may differ. If the value cannot be represented, the behavior is
4977 undefined.4978 **RETURN VALUE**

4979 These functions shall return the converted value if the value can be represented.

4980 **ERRORS**

4981 No errors are defined.

4982 **EXAMPLES**

4983 None.

4984 **APPLICATION USAGE**4985 The *atol()* function is subsumed by *strtol()* but is retained because it is used extensively in
4986 existing code. If the number is not known to be in range, *strtol()* should be used because *atol()* is
4987 not required to perform any error checking.4988 **RATIONALE**

4989 None.

4990 **FUTURE DIRECTIONS**

4991 None.

4992 **SEE ALSO**4993 *strtol()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>4994 **CHANGE HISTORY**

4995 First released in Issue 1. Derived from Issue 1 of the SVID.

4996 **Issue 6**4997 The *atoll()* function is added for alignment with the ISO/IEC 9899:1999 standard.

4998 **NAME**

4999 `basename` — return the last component of a pathname

5000 **SYNOPSIS**

```
5001 xSI #include <libgen.h>
5002 char *basename(char *path);
5003
```

5004 **DESCRIPTION**

5005 The *basename()* function shall take the pathname pointed to by *path* and return a pointer to the
5006 final component of the pathname, deleting any trailing '/' characters.

5007 If the string consists entirely of the '/' character, *basename()* shall return a pointer to the string
5008 "/". If the string is exactly "///", it is implementation-defined whether '/' or "///" is
5009 returned.

5010 If *path* is a null pointer or points to an empty string, *basename()* shall return a pointer to the
5011 string ".".

5012 The *basename()* function may modify the string pointed to by *path*, and may return a pointer to
5013 static storage that may then be overwritten by a subsequent call to *basename()*.

5014 The *basename()* function need not be reentrant. A function that is not required to be reentrant is
5015 not required to be thread-safe.

5016 **RETURN VALUE**

5017 The *basename()* function shall return a pointer to the final component of *path*.

5018 **ERRORS**

5019 No errors are defined.

5020 **EXAMPLES**

5021 **Using basename()**

5022 The following program fragment returns a pointer to the value *lib*, which is the base name of
5023 */usr/lib*.

```
5024 #include <libgen.h>
5025 ...
5026 char *name = "/usr/lib";
5027 char *base;
5028 base = basename(name);
5029 ...
```

5030 **Sample Input and Output Strings for basename()**

5031 In the following table, the input string is the value pointed to by *path*, and the output string is
5032 the return value of the *basename()* function.

Input String	Output String
"/usr/lib"	"lib"
"/usr/"	"usr"
"/"	"/"
"/" / "///"	"/"
"/usr//lib/"	"lib"

5039 **APPLICATION USAGE**

5040 None.

5041 **RATIONALE**

5042 None.

5043 **FUTURE DIRECTIONS**

5044 None.

5045 **SEE ALSO**5046 *dirname()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**libgen.h**>, the Shell and
5047 Utilities volume of IEEE Std 1003.1-2001, *basename*5048 **CHANGE HISTORY**

5049 First released in Issue 4, Version 2.

5050 **Issue 5**

5051 Moved from X/OPEN UNIX extension to BASE.

5052 Normative text previously in the APPLICATION USAGE section is moved to the
5053 DESCRIPTION.

5054 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

5055 **Issue 6**

5056 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

5057 **NAME**5058 bcmp — memory operations (**LEGACY**)5059 **SYNOPSIS**

5060 xSI #include <strings.h>

5061 int bcmp(const void *s1, const void *s2, size_t n);

5062

5063 **DESCRIPTION**5064 The *bcmp()* function shall compare the first *n* bytes of the area pointed to by *s1* with the area pointed to by *s2*.5066 **RETURN VALUE**5067 The *bcmp()* function shall return 0 if *s1* and *s2* are identical; otherwise, it shall return non-zero.5068 Both areas are assumed to be *n* bytes long. If the value of *n* is 0, *bcmp()* shall return 0.5069 **ERRORS**

5070 No errors are defined.

5071 **EXAMPLES**

5072 None.

5073 **APPLICATION USAGE**5074 The *memcmp()* function is preferred over this function.5075 For maximum portability, it is recommended to replace the function call to *bcmp()* as follows:

5076 #define bcmp(b1,b2,len) memcmp((b1), (b2), (size_t)(len))

5077 **RATIONALE**

5078 None.

5079 **FUTURE DIRECTIONS**

5080 This function may be withdrawn in a future version.

5081 **SEE ALSO**5082 *memcmp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <strings.h>5083 **CHANGE HISTORY**

5084 First released in Issue 4, Version 2.

5085 **Issue 5**

5086 Moved from X/OPEN UNIX extension to BASE.

5087 **Issue 6**

5088 This function is marked LEGACY.

5089 **NAME**5090 `bcopy` — memory operations (**LEGACY**)5091 **SYNOPSIS**5092 XSI `#include <strings.h>`5093 `void bcopy(const void *s1, void *s2, size_t n);`

5094

5095 **DESCRIPTION**5096 The `bcopy()` function shall copy `n` bytes from the area pointed to by `s1` to the area pointed to by
5097 `s2`.5098 The bytes are copied correctly even if the area pointed to by `s1` overlaps the area pointed to by
5099 `s2`.5100 **RETURN VALUE**5101 The `bcopy()` function shall not return a value.5102 **ERRORS**

5103 No errors are defined.

5104 **EXAMPLES**

5105 None.

5106 **APPLICATION USAGE**5107 The `memmove()` function is preferred over this function.

5108 The following are approximately equivalent (note the order of the arguments):

5109 `bcopy(s1, s2, n) ~ memmove(s2, s1, n)`5110 For maximum portability, it is recommended to replace the function call to `bcopy()` as follows:5111 `#define bcopy(b1,b2,len) memmove((b2), (b1), (len)), (void) 0`5112 **RATIONALE**

5113 None.

5114 **FUTURE DIRECTIONS**

5115 This function may be withdrawn in a future version.

5116 **SEE ALSO**5117 `memmove()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<strings.h>`5118 **CHANGE HISTORY**

5119 First released in Issue 4, Version 2.

5120 **Issue 5**

5121 Moved from X/OPEN UNIX extension to BASE.

5122 **Issue 6**

5123 This function is marked LEGACY.

5124 **NAME**

5125 bind — bind a name to a socket

5126 **SYNOPSIS**

5127 #include <sys/socket.h>

5128 int bind(int *socket*, const struct sockaddr **address*,
5129 socklen_t *address_len*);5130 **DESCRIPTION**5131 The *bind()* function shall assign a local socket address *address* to a socket identified by descriptor
5132 *socket* that has no local socket address assigned. Sockets created with the *socket()* function are
5133 initially unnamed; they are identified only by their address family.5134 The *bind()* function takes the following arguments:5135 *socket* Specifies the file descriptor of the socket to be bound.5136 *address* Points to a **sockaddr** structure containing the address to be bound to the
5137 socket. The length and format of the address depend on the address family of
5138 the socket.5139 *address_len* Specifies the length of the **sockaddr** structure pointed to by the *address*
5140 argument.5141 The socket specified by *socket* may require the process to have appropriate privileges to use the
5142 *bind()* function.5143 **RETURN VALUE**5144 Upon successful completion, *bind()* shall return 0; otherwise, -1 shall be returned and *errno* set
5145 to indicate the error.5146 **ERRORS**5147 The *bind()* function shall fail if:

5148 [EADDRINUSE] The specified address is already in use.

5149 [EADDRNOTAVAIL]

5150 The specified address is not available from the local machine.

5151 [EAFNOSUPPORT]

5152 The specified address is not a valid address for the address family of the
5153 specified socket.5154 [EBADF] The *socket* argument is not a valid file descriptor.5155 [EINVAL] The socket is already bound to an address, and the protocol does not support
5156 binding to a new address; or the socket has been shut down.5157 [ENOTSOCK] The *socket* argument does not refer to a socket.5158 [EOPNOTSUPP] The socket type of the specified socket does not support binding to an
5159 address.5160 If the address family of the socket is AF_UNIX, then *bind()* shall fail if:5161 [EACCES] A component of the path prefix denies search permission, or the requested
5162 name requires writing in a directory with a mode that denies write
5163 permission.

5164 [EDESTADDRREQ] or [EISDIR]

5165 The *address* argument is a null pointer.

5166	[EIO]	An I/O error occurred.
5167	[ELOOP]	A loop exists in symbolic links encountered during resolution of the pathname in <i>address</i> .
5168		
5169	[ENAMETOOLONG]	
5170		A component of a pathname exceeded {NAME_MAX} characters, or an entire pathname exceeded {PATH_MAX} characters.
5171		
5172	[ENOENT]	A component of the pathname does not name an existing file or the pathname is an empty string.
5173		
5174	[ENOTDIR]	A component of the path prefix of the pathname in <i>address</i> is not a directory.
5175	[EROFS]	The name would reside on a read-only file system.
5176		The <i>bind()</i> function may fail if:
5177	[EACCES]	The specified address is protected and the current user does not have permission to bind to it.
5178		
5179	[EINVAL]	The <i>address_len</i> argument is not a valid length for the address family.
5180	[EISCONN]	The socket is already connected.
5181	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in <i>address</i> .
5182		
5183	[ENAMETOOLONG]	
5184		Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.
5185		
5186	[ENOBUFS]	Insufficient resources were available to complete the call.

5187 EXAMPLES

5188 None.

5189 APPLICATION USAGE

5190 An application program can retrieve the assigned socket name with the *getsockname()* function.

5191 RATIONALE

5192 None.

5193 FUTURE DIRECTIONS

5194 None.

5195 SEE ALSO

5196 *connect()*, *getsockname()*, *listen()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-2001,
5197 <sys/socket.h>

5198 CHANGE HISTORY

5199 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

5200 **NAME**

5201 bsd_signal — simplified signal facilities

5202 **SYNOPSIS**

5203 OB XSI #include <signal.h>

5204 void (*bsd_signal(int sig, void (*func)(int)))(int);

5205

5206 **DESCRIPTION**5207 The *bsd_signal()* function provides a partially compatible interface for programs written to
5208 historical system interfaces (see APPLICATION USAGE).5209 The function call *bsd_signal(sig, func)* shall be equivalent to the following:

5210 void (*bsd_signal(int sig, void (*func)(int)))(int)

5211 {
5212 struct sigaction act, oact;
5213 act.sa_handler = func;
5214 act.sa_flags = SA_RESTART;
5215 sigemptyset(&act.sa_mask);
5216 sigaddset(&act.sa_mask, sig);
5217 if (sigaction(sig, &act, &oact) == -1)
5218 return(SIG_ERR);
5219 return(oact.sa_handler);
5220 }

5221 The handler function should be declared:

5222 void handler(int sig);

5223 where *sig* is the signal number. The behavior is undefined if *func* is a function that takes more
5224 than one argument, or an argument of a different type.5225 **RETURN VALUE**5226 Upon successful completion, *bsd_signal()* shall return the previous action for *sig*. Otherwise,
5227 SIG_ERR shall be returned and *errno* shall be set to indicate the error.5228 **ERRORS**5229 Refer to *sigaction()*.5230 **EXAMPLES**

5231 None.

5232 **APPLICATION USAGE**5233 This function is a direct replacement for the BSD *signal()* function for simple applications that
5234 are installing a single-argument signal handler function. If a BSD signal handler function is being
5235 installed that expects more than one argument, the application has to be modified to use
5236 *sigaction()*. The *bsd_signal()* function differs from *signal()* in that the SA_RESTART flag is set
5237 and the SA_RESETHAND is clear when *bsd_signal()* is used. The state of these flags is not
5238 specified for *signal()*.5239 It is recommended that new applications use the *sigaction()* function.5240 **RATIONALE**

5241 None.

5242 **FUTURE DIRECTIONS**

5243 None.

5244 **SEE ALSO**5245 *sigaction()*, *sigaddset()*, *sigemptyset()*, *signal()*, the Base Definitions volume of
5246 IEEE Std 1003.1-2001, <**signal.h**>5247 **CHANGE HISTORY**

5248 First released in Issue 4, Version 2.

5249 **Issue 5**

5250 Moved from X/OPEN UNIX extension to BASE.

5251 **Issue 6**

5252 This function is marked obsolescent.

5253 **NAME**5254 **bsearch** — binary search a sorted table5255 **SYNOPSIS**

5256 #include <stdlib.h>

5257 void *bsearch(const void *key, const void *base, size_t nel,
5258 size_t width, int (*compar)(const void *, const void *));5259 **DESCRIPTION**5260 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
5261 conflict between the requirements described here and the ISO C standard is unintentional. This
5262 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.5263 The *bsearch()* function shall search an array of *nel* objects, the initial element of which is pointed
5264 to by *base*, for an element that matches the object pointed to by *key*. The size of each element in
5265 the array is specified by *width*. If the *nel* argument has the value zero, the comparison function
5266 pointed to by *compar* shall not be called and no match shall be found.5267 The comparison function pointed to by *compar* shall be called with two arguments that point to
5268 the *key* object and to an array element, in that order.5269 The application shall ensure that the comparison function pointed to by *compar* does not alter the
5270 contents of the array. The implementation may reorder elements of the array between calls to the
5271 comparison function, but shall not alter the contents of any individual element.

5272 The implementation shall ensure that the first argument is always a pointer to the key.

5273 When the same objects (consisting of *width* bytes, irrespective of their current positions in the
5274 array) are passed more than once to the comparison function, the results shall be consistent with
5275 one another. That is, the same object shall always compare the same way with the key.5276 The application shall ensure that the function returns an integer less than, equal to, or greater
5277 than 0 if the *key* object is considered, respectively, to be less than, to match, or to be greater than
5278 the array element. The application shall ensure that the array consists of all the elements that
5279 compare less than, all the elements that compare equal to, and all the elements that compare
5280 greater than the *key* object, in that order.5281 **RETURN VALUE**5282 The *bsearch()* function shall return a pointer to a matching member of the array, or a null pointer
5283 if no match is found. If two or more members compare equal, which member is returned is
5284 unspecified.5285 **ERRORS**

5286 No errors are defined.

5287 **EXAMPLES**5288 The example below searches a table containing pointers to nodes consisting of a string and its
5289 length. The table is ordered alphabetically on the string in the node pointed to by each entry.5290 The code fragment below reads in strings and either finds the corresponding node and prints out
5291 the string and its length, or prints an error message.

5292 #include <stdio.h>

5293 #include <stdlib.h>

5294 #include <string.h>

5295 #define TABSIZE 1000

```

5296     struct node {                               /* These are stored in the table. */
5297         char *string;
5298         int length;
5299     };
5300     struct node table[TABSIZE];    /* Table to be searched. */
5301     .
5302     .
5303     .
5304     {
5305         struct node *node_ptr, node;
5306         /* Routine to compare 2 nodes. */
5307         int node_compare(const void *, const void *);
5308         char str_space[20];    /* Space to read string into. */
5309         .
5310         .
5311         .
5312         node.string = str_space;
5313         while (scanf("%s", node.string) != EOF) {
5314             node_ptr = (struct node *)bsearch((void *)&node,
5315                 (void *)table, TABSIZE,
5316                 sizeof(struct node), node_compare);
5317             if (node_ptr != NULL) {
5318                 (void)printf("string = %20s, length = %d\n",
5319                     node_ptr->string, node_ptr->length);
5320             } else {
5321                 (void)printf("not found: %s\n", node.string);
5322             }
5323         }
5324     }
5325     /*
5326     This routine compares two nodes based on an
5327     alphabetical ordering of the string field.
5328     */
5329     int
5330     node_compare(const void *node1, const void *node2)
5331     {
5332         return strcoll(((const struct node *)node1)->string,
5333             ((const struct node *)node2)->string);
5334     }

```

5335 APPLICATION USAGE

5336 The pointers to the key and the element at the base of the table should be of type pointer-to-
5337 element.

5338 The comparison function need not compare every byte, so arbitrary data may be contained in
5339 the elements in addition to the values being compared.

5340 In practice, the array is usually sorted according to the comparison function.

5341 RATIONALE

5342 The requirement that the second argument (hereafter referred to as *p*) to the comparison
5343 function is a pointer to an element of the array implies that for every call all of the following
5344 expressions are non-zero: |

```
5345     ((char *)p - (char *(base) % width == 0
5346     (char *)p >= (char *)base
5347     (char *)p < (char *)base + nel * width
```

5348 FUTURE DIRECTIONS

5349 None.

5350 SEE ALSO

5351 *hcreate()*, *lsearch()*, *qsort()*, *tsearch()*, the Base Definitions volume of IEEE Std 1003.1-2001,
5352 <stdlib.h>

5353 CHANGE HISTORY

5354 First released in Issue 1. Derived from Issue 1 of the SVID.

5355 Issue 6

5356 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

5357 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/11 is applied, adding to the
5358 DESCRIPTION the last sentence of the first non-shaded paragraph, and the following three
5359 paragraphs. The RATIONALE section is also updated. These changes are for alignment with the
5360 ISO C standard.

5361 **NAME**

5362 btowc — single byte to wide character conversion

5363 **SYNOPSIS**

5364 #include <stdio.h>

5365 #include <wchar.h>

5366 wint_t btowc(int c);

5367 **DESCRIPTION**

5368 cx The functionality described on this reference page is aligned with the ISO C standard. Any
5369 conflict between the requirements described here and the ISO C standard is unintentional. This
5370 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5371 The *btowc()* function shall determine whether *c* constitutes a valid (one-byte) character in the
5372 initial shift state.

5373 The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

5374 **RETURN VALUE**

5375 The *btowc()* function shall return WEOF if *c* has the value EOF or if (**unsigned char**) *c* does not
5376 constitute a valid (one-byte) character in the initial shift state. Otherwise, it shall return the
5377 wide-character representation of that character.

5378 **ERRORS**

5379 No errors are defined.

5380 **EXAMPLES**

5381 None.

5382 **APPLICATION USAGE**

5383 None.

5384 **RATIONALE**

5385 None.

5386 **FUTURE DIRECTIONS**

5387 None.

5388 **SEE ALSO**5389 *wctob()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>5390 **CHANGE HISTORY**

5391 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
5392 (E).

5393 **NAME**5394 **bzero** — memory operations (**LEGACY**)5395 **SYNOPSIS**5396 XSI `#include <strings.h>`5397 `void bzero(void *s, size_t n);`

5398

5399 **DESCRIPTION**5400 The *bzero()* function shall place *n* zero-valued bytes in the area pointed to by *s*.5401 **RETURN VALUE**5402 The *bzero()* function shall not return a value.5403 **ERRORS**

5404 No errors are defined.

5405 **EXAMPLES**

5406 None.

5407 **APPLICATION USAGE**5408 The *memset()* function is preferred over this function.5409 For maximum portability, it is recommended to replace the function call to *bzero()* as follows:5410 `#define bzero(b,len) (memset((b), '\0', (len)), (void) 0)`5411 **RATIONALE**

5412 None.

5413 **FUTURE DIRECTIONS**

5414 This function may be withdrawn in a future version.

5415 **SEE ALSO**5416 *memset()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<strings.h>**5417 **CHANGE HISTORY**

5418 First released in Issue 4, Version 2.

5419 **Issue 5**

5420 Moved from X/OPEN UNIX extension to BASE.

5421 **Issue 6**

5422 This function is marked LEGACY.

5423 **NAME**

5424 cabs, cabsf, cabsl — return a complex absolute value

5425 **SYNOPSIS**

5426 #include <complex.h>

5427 double cabs(double complex z);

5428 float cabsf(float complex z);

5429 long double cabsl(long double complex z);

5430 **DESCRIPTION**

5431 cx The functionality described on this reference page is aligned with the ISO C standard. Any
5432 conflict between the requirements described here and the ISO C standard is unintentional. This
5433 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5434 These functions shall compute the complex absolute value (also called norm, modulus, or
5435 magnitude) of *z*.

5436 **RETURN VALUE**

5437 These functions shall return the complex absolute value.

5438 **ERRORS**

5439 No errors are defined.

5440 **EXAMPLES**

5441 None.

5442 **APPLICATION USAGE**

5443 None.

5444 **RATIONALE**

5445 None.

5446 **FUTURE DIRECTIONS**

5447 None.

5448 **SEE ALSO**

5449 The Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>

5450 **CHANGE HISTORY**

5451 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5452 **NAME**

5453 cacos, cacosf, cacosl — complex arc cosine functions

5454 **SYNOPSIS**

5455 #include <complex.h>

5456 double complex cacos(double complex z);

5457 float complex cacosf(float complex z);

5458 long double complex cacosl(long double complex z);

5459 **DESCRIPTION**

5460 cx The functionality described on this reference page is aligned with the ISO C standard. Any
5461 conflict between the requirements described here and the ISO C standard is unintentional. This
5462 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5463 These functions shall compute the complex arc cosine of z , with branch cuts outside the interval
5464 $[-1, +1]$ along the real axis.

5465 **RETURN VALUE**

5466 These functions shall return the complex arc cosine value, in the range of a strip mathematically
5467 unbounded along the imaginary axis and in the interval $[0, \pi]$ along the real axis.

5468 **ERRORS**

5469 No errors are defined.

5470 **EXAMPLES**

5471 None.

5472 **APPLICATION USAGE**

5473 None.

5474 **RATIONALE**

5475 None.

5476 **FUTURE DIRECTIONS**

5477 None.

5478 **SEE ALSO**5479 *ccos()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>5480 **CHANGE HISTORY**

5481 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5482 **NAME**5483 `cacosh`, `cacoshf`, `cacoshl` — complex arc hyperbolic cosine functions5484 **SYNOPSIS**5485 `#include <complex.h>`5486 `double complex cacosh(double complex z);`5487 `float complex cacoshf(float complex z);`5488 `long double complex cacoshl(long double complex z);`5489 **DESCRIPTION**

5490 `CX` The functionality described on this reference page is aligned with the ISO C standard. Any
5491 conflict between the requirements described here and the ISO C standard is unintentional. This
5492 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5493 These functions shall compute the complex arc hyperbolic cosine of z , with a branch cut at
5494 values less than 1 along the real axis.

5495 **RETURN VALUE**

5496 These functions shall return the complex arc hyperbolic cosine value, in the range of a half-strip
5497 of non-negative values along the real axis and in the interval $[-i\pi, +i\pi]$ along the imaginary axis.

5498 **ERRORS**

5499 No errors are defined.

5500 **EXAMPLES**

5501 None.

5502 **APPLICATION USAGE**

5503 None.

5504 **RATIONALE**

5505 None.

5506 **FUTURE DIRECTIONS**

5507 None.

5508 **SEE ALSO**5509 `ccosh()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<complex.h>`5510 **CHANGE HISTORY**

5511 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5512 **NAME**

5513 cacosl — complex arc cosine functions

5514 **SYNOPSIS**

5515 #include <complex.h>

5516 long double complex cacosl(long double complex z);

5517 **DESCRIPTION**

5518 Refer to *cacos()*.

5519 **NAME**5520 *calloc* — a memory allocator5521 **SYNOPSIS**

5522 #include <stdlib.h>

5523 void *calloc(size_t *nelem*, size_t *elsize*);5524 **DESCRIPTION**5525 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
5526 conflict between the requirements described here and the ISO C standard is unintentional. This
5527 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.5528 The *calloc()* function shall allocate unused space for an array of *nelem* elements each of whose
5529 size in bytes is *elsize*. The space shall be initialized to all bits 0.5530 The order and contiguity of storage allocated by successive calls to *calloc()* is unspecified. The
5531 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to
5532 a pointer to any type of object and then used to access such an object or an array of such objects
5533 in the space allocated (until the space is explicitly freed or reallocated). Each such allocation
5534 shall yield a pointer to an object disjoint from any other object. The pointer returned shall point
5535 to the start (lowest byte address) of the allocated space. If the space cannot be allocated, a null
5536 pointer shall be returned. If the size of the space requested is 0, the behavior is implementation-
5537 defined: the value returned shall be either a null pointer or a unique pointer.5538 **RETURN VALUE**5539 Upon successful completion with both *nelem* and *elsize* non-zero, *calloc()* shall return a pointer to
5540 the allocated space. If either *nelem* or *elsize* is 0, then either a null pointer or a unique pointer
5541 value that can be successfully passed to *free()* shall be returned. Otherwise, it shall return a null
5542 **CX** pointer and set *errno* to indicate the error.5543 **ERRORS**5544 The *calloc()* function shall fail if:5545 **CX** [ENOMEM] Insufficient memory is available.5546 **EXAMPLES**

5547 None.

5548 **APPLICATION USAGE**

5549 There is now no requirement for the implementation to support the inclusion of <malloc.h>.

5550 **RATIONALE**

5551 None.

5552 **FUTURE DIRECTIONS**

5553 None.

5554 **SEE ALSO**5555 *free()*, *malloc()*, *realloc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>5556 **CHANGE HISTORY**

5557 First released in Issue 1. Derived from Issue 1 of the SVID.

5558 **Issue 6**

5559 Extensions beyond the ISO C standard are marked.

5560
5561
5562
5563

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The setting of *errno* and the [ENOMEM] error condition are mandatory if an insufficient memory condition occurs.

5564 **NAME**

5565 carg, cargf, cargl — complex argument functions

5566 **SYNOPSIS**

5567 #include <complex.h>

5568 double carg(double complex z);

5569 float cargf(float complex z);

5570 long double cargl(long double complex z);

5571 **DESCRIPTION**

5572 cx The functionality described on this reference page is aligned with the ISO C standard. Any
5573 conflict between the requirements described here and the ISO C standard is unintentional. This
5574 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5575 These functions shall compute the argument (also called phase angle) of z , with a branch cut
5576 along the negative real axis.

5577 **RETURN VALUE**5578 These functions shall return the value of the argument in the interval $[-\pi, +\pi]$.5579 **ERRORS**

5580 No errors are defined.

5581 **EXAMPLES**

5582 None.

5583 **APPLICATION USAGE**

5584 None.

5585 **RATIONALE**

5586 None.

5587 **FUTURE DIRECTIONS**

5588 None.

5589 **SEE ALSO**5590 *cimag()*, *conj()*, *cproj()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>5591 **CHANGE HISTORY**

5592 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5593 **NAME**

5594 `casin`, `casinf`, `casinl` — complex arc sine functions

5595 **SYNOPSIS**

5596 `#include <complex.h>`

5597 `double complex casin(double complex z);`

5598 `float complex casinf(float complex z);`

5599 `long double complex casinl(long double complex z);`

5600 **DESCRIPTION**

5601 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 5602 conflict between the requirements described here and the ISO C standard is unintentional. This
 5603 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5604 These functions shall compute the complex arc sine of z , with branch cuts outside the interval
 5605 $[-1, +1]$ along the real axis.

5606 **RETURN VALUE**

5607 These functions shall return the complex arc sine value, in the range of a strip mathematically
 5608 unbounded along the imaginary axis and in the interval $[-\pi/2, +\pi/2]$ along the real axis.

5609 **ERRORS**

5610 No errors are defined.

5611 **EXAMPLES**

5612 None.

5613 **APPLICATION USAGE**

5614 None.

5615 **RATIONALE**

5616 None.

5617 **FUTURE DIRECTIONS**

5618 None.

5619 **SEE ALSO**

5620 `csin()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<complex.h>`

5621 **CHANGE HISTORY**

5622 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5623 **NAME**

5624 casinh, casinhf, casinhl — complex arc hyperbolic sine functions

5625 **SYNOPSIS**

5626 #include <complex.h>

5627 double complex casinh(double complex z);

5628 float complex casinhf(float complex z);

5629 long double complex casinhl(long double complex z);

5630 **DESCRIPTION**

5631 cx The functionality described on this reference page is aligned with the ISO C standard. Any
5632 conflict between the requirements described here and the ISO C standard is unintentional. This
5633 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5634 These functions shall compute the complex arc hyperbolic sine of z , with branch cuts outside the
5635 interval $[-i, +i]$ along the imaginary axis.

5636 **RETURN VALUE**

5637 These functions shall return the complex arc hyperbolic sine value, in the range of a strip
5638 mathematically unbounded along the real axis and in the interval $[-i\pi/2, +i\pi/2]$ along the
5639 imaginary axis.

5640 **ERRORS**

5641 No errors are defined.

5642 **EXAMPLES**

5643 None.

5644 **APPLICATION USAGE**

5645 None.

5646 **RATIONALE**

5647 None.

5648 **FUTURE DIRECTIONS**

5649 None.

5650 **SEE ALSO**5651 `csinh()`, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>5652 **CHANGE HISTORY**

5653 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5654 **NAME**5655 **casinl** — complex arc sine functions5656 **SYNOPSIS**

5657 #include <complex.h>

5658 long double complex casinl(long double complex *z*);5659 **DESCRIPTION**5660 Refer to *casin()*.

5661 **NAME**

5662 catan, catanf, catanl — complex arc tangent functions

5663 **SYNOPSIS**

5664 #include <complex.h>

5665 double complex catan(double complex z);

5666 float complex catanf(float complex z);

5667 long double complex catanl(long double complex z);

5668 **DESCRIPTION**5669 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
5670 conflict between the requirements described here and the ISO C standard is unintentional. This
5671 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.5672 These functions shall compute the complex arc tangent of z , with branch cuts outside the
5673 interval $[-i, +i]$ along the imaginary axis.5674 **RETURN VALUE**5675 These functions shall return the complex arc tangent value, in the range of a strip
5676 mathematically unbounded along the imaginary axis and in the interval $[-\pi/2, +\pi/2]$ along the
5677 real axis.5678 **ERRORS**

5679 No errors are defined.

5680 **EXAMPLES**

5681 None.

5682 **APPLICATION USAGE**

5683 None.

5684 **RATIONALE**

5685 None.

5686 **FUTURE DIRECTIONS**

5687 None.

5688 **SEE ALSO**

5689 ctan(), the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>

5690 **CHANGE HISTORY**

5691 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5692 **NAME**

5693 catanh, catanhf, catanhl — complex arc hyperbolic tangent functions

5694 **SYNOPSIS**

5695 #include <complex.h>

5696 double complex catanh(double complex z);

5697 float complex catanhf(float complex z);

5698 long double complex catanhl(long double complex z);

5699 **DESCRIPTION**5700 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
5701 conflict between the requirements described here and the ISO C standard is unintentional. This
5702 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.5703 These functions shall compute the complex arc hyperbolic tangent of z , with branch cuts outside
5704 the interval $[-1, +1]$ along the real axis.5705 **RETURN VALUE**5706 These functions shall return the complex arc hyperbolic tangent value, in the range of a strip
5707 mathematically unbounded along the real axis and in the interval $[-i\pi/2, +i\pi/2]$ along the
5708 imaginary axis.5709 **ERRORS**

5710 No errors are defined.

5711 **EXAMPLES**

5712 None.

5713 **APPLICATION USAGE**

5714 None.

5715 **RATIONALE**

5716 None.

5717 **FUTURE DIRECTIONS**

5718 None.

5719 **SEE ALSO**5720 *ctanh()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>5721 **CHANGE HISTORY**

5722 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5723 **NAME**

5724 catanl — complex arc tangent functions

5725 **SYNOPSIS**

5726 #include <complex.h>

5727 long double complex catanl(long double complex z);

5728 **DESCRIPTION**5729 Refer to *catan()*.

5730 **NAME**

5731 catclose — close a message catalog descriptor

5732 **SYNOPSIS**

5733 XSI #include <nl_types.h>

5734 int catclose(nl_catd catd);

5735

5736 **DESCRIPTION**5737 The *catclose()* function shall close the message catalog identified by *catd*. If a file descriptor is
5738 used to implement the type **nl_catd**, that file descriptor shall be closed.5739 **RETURN VALUE**5740 Upon successful completion, *catclose()* shall return 0; otherwise, -1 shall be returned, and *errno*
5741 set to indicate the error.5742 **ERRORS**5743 The *catclose()* function may fail if:

5744 [EBADF] The catalog descriptor is not valid.

5745 [EINTR] The *catclose()* function was interrupted by a signal.5746 **EXAMPLES**

5747 None.

5748 **APPLICATION USAGE**

5749 None.

5750 **RATIONALE**

5751 None.

5752 **FUTURE DIRECTIONS**

5753 None.

5754 **SEE ALSO**5755 *catgets()*, *catopen()*, the Base Definitions volume of IEEE Std 1003.1-2001, <nl_types.h>5756 **CHANGE HISTORY**

5757 First released in Issue 2.

5758 **NAME**

5759 catgets — read a program message

5760 **SYNOPSIS**

5761 XSI #include <nl_types.h>

5762 char *catgets(nl_catd catd, int set_id, int msg_id, const char *s);

5763

5764 **DESCRIPTION**

5765 The *catgets()* function shall attempt to read message *msg_id*, in set *set_id*, from the message catalog identified by *catd*. The *catd* argument is a message catalog descriptor returned from an earlier call to *catopen()*. The *s* argument points to a default message string which shall be returned by *catgets()* if it cannot retrieve the identified message.

5769 The *catgets()* function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

5771 **RETURN VALUE**

5772 If the identified message is retrieved successfully, *catgets()* shall return a pointer to an internal buffer area containing the null-terminated message string. If the call is unsuccessful for any reason, *s* shall be returned and *errno* may be set to indicate the error.

5775 **ERRORS**5776 The *catgets()* function may fail if:

5777 [EBADF] The *catd* argument is not a valid message catalog descriptor open for reading.

5778 [EBADMSG] The message identified by *set_id* and *msg_id* in the specified message catalog did not satisfy implementation-defined security criteria.

5780 [EINTR] The read operation was terminated due to the receipt of a signal, and no data was transferred.

5782 [EINVAL] The message catalog identified by *catd* is corrupted.

5783 [ENOMSG] The message identified by *set_id* and *msg_id* is not in the message catalog.

5784 **EXAMPLES**

5785 None.

5786 **APPLICATION USAGE**

5787 None.

5788 **RATIONALE**

5789 None.

5790 **FUTURE DIRECTIONS**

5791 None.

5792 **SEE ALSO**5793 *catclose()*, *catopen()*, the Base Definitions volume of IEEE Std 1003.1-2001, <nl_types.h>5794 **CHANGE HISTORY**

5795 First released in Issue 2.

5796 **Issue 5**

5797 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

5798 **Issue 6**

5799 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

5800 **NAME**5801 `catopen` — open a message catalog5802 **SYNOPSIS**5803 XSI `#include <nl_types.h>`5804 `nl_catd catopen(const char *name, int oflag);`

5805

5806 **DESCRIPTION**

5807 The `catopen()` function shall open a message catalog and return a message catalog descriptor.
 5808 The `name` argument specifies the name of the message catalog to be opened. If `name` contains a
 5809 `'/'`, then `name` specifies a complete name for the message catalog. Otherwise, the environment
 5810 variable `NLSPATH` is used with `name` substituted for the `%N` conversion specification (see the
 5811 Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables). If
 5812 `NLSPATH` exists in the environment when the process starts, then if the process has appropriate
 5813 privileges, the behavior of `catopen()` is undefined. If `NLSPATH` does not exist in the environment,
 5814 or if a message catalog cannot be found in any of the components specified by `NLSPATH`, then
 5815 an implementation-defined default path shall be used. This default may be affected by the
 5816 setting of `LC_MESSAGES` if the value of `oflag` is `NL_CAT_LOCALE`, or the `LANG` environment
 5817 variable if `oflag` is 0.

5818 A message catalog descriptor shall remain valid in a process until that process closes it, or a
 5819 successful call to one of the `exec` functions. A change in the setting of the `LC_MESSAGES`
 5820 category may invalidate existing open catalogs.

5821 If a file descriptor is used to implement message catalog descriptors, the `FD_CLOEXEC` flag
 5822 shall be set; see `<fcntl.h>`.

5823 If the value of the `oflag` argument is 0, the `LANG` environment variable is used to locate the
 5824 catalog without regard to the `LC_MESSAGES` category. If the `oflag` argument is
 5825 `NL_CAT_LOCALE`, the `LC_MESSAGES` category is used to locate the message catalog (see the
 5826 Base Definitions volume of IEEE Std 1003.1-2001, Section 8.2, Internationalization Variables).

5827 **RETURN VALUE**

5828 Upon successful completion, `catopen()` shall return a message catalog descriptor for use on
 5829 subsequent calls to `catgets()` and `catclose()`. Otherwise, `catopen()` shall return `(nl_catd) -1` and set
 5830 `errno` to indicate the error.

5831 **ERRORS**5832 The `catopen()` function may fail if:

5833 [EACCES] Search permission is denied for the component of the path prefix of the
 5834 message catalog or read permission is denied for the message catalog.

5835 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

5836 [ENAMETOOLONG]

5837 The length of a pathname of the message catalog exceeds {PATH_MAX} or a
 5838 pathname component is longer than {NAME_MAX}.

5839 [ENAMETOOLONG]

5840 Pathname resolution of a symbolic link produced an intermediate result
 5841 whose length exceeds {PATH_MAX}.

5842 [ENFILE] Too many files are currently open in the system.

5843 [ENOENT] The message catalog does not exist or the `name` argument points to an empty
 5844 string.

- 5845 [ENOMEM] Insufficient storage space is available.
5846 [ENOTDIR] A component of the path prefix of the message catalog is not a directory.

5847 EXAMPLES

5848 None.

5849 APPLICATION USAGE

5850 Some implementations of *catopen()* use *malloc()* to allocate space for internal buffer areas. The
5851 *catopen()* function may fail if there is insufficient storage space available to accommodate these
5852 buffers.

5853 Conforming applications must assume that message catalog descriptors are not valid after a call
5854 to one of the *exec* functions.

5855 Application writers should be aware that guidelines for the location of message catalogs have
5856 not yet been developed. Therefore they should take care to avoid conflicting with catalogs used
5857 by other applications and the standard utilities.

5858 RATIONALE

5859 None.

5860 FUTURE DIRECTIONS

5861 None.

5862 SEE ALSO

5863 *catclose()*, *catgets()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<fcntl.h>`,
5864 `<nl_types.h>`, the Shell and Utilities volume of IEEE Std 1003.1-2001

5865 CHANGE HISTORY

5866 First released in Issue 2.

5867 **NAME**

5868 cbrt, cbrtf, cbrtl — cube root functions

5869 **SYNOPSIS**

5870 #include <math.h>

5871 double cbrt(double x);

5872 float cbrtf(float x);

5873 long double cbrtl(long double x);

5874 **DESCRIPTION**

5875 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 5876 conflict between the requirements described here and the ISO C standard is unintentional. This
 5877 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5878 These functions shall compute the real cube root of their argument *x*.5879 **RETURN VALUE**5880 Upon successful completion, these functions shall return the cube root of *x*.5881 **MX** If *x* is NaN, a NaN shall be returned.5882 If *x* is ± 0 or $\pm\text{Inf}$, *x* shall be returned.5883 **ERRORS**

5884 No errors are defined.

5885 **EXAMPLES**

5886 None.

5887 **APPLICATION USAGE**

5888 None.

5889 **RATIONALE**

5890 For some applications, a true cube root function, which returns negative results for negative
 5891 arguments, is more appropriate than *pow(x, 1.0/3.0)*, which returns a NaN for *x* less than 0.

5892 **FUTURE DIRECTIONS**

5893 None.

5894 **SEE ALSO**

5895 The Base Definitions volume of IEEE Std 1003.1-2001, <math.h>

5896 **CHANGE HISTORY**

5897 First released in Issue 4, Version 2.

5898 **Issue 5**

5899 Moved from X/OPEN UNIX extension to BASE.

5900 **Issue 6**5901 The *cbrt()* function is no longer marked as an extension.5902 The *cbrtf()* and *cbrtl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

5903 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 5904 revised to align with the ISO/IEC 9899:1999 standard.

5905 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 5906 marked.

5907 **NAME**

5908 ccos, ccosf, ccosl — complex cosine functions

5909 **SYNOPSIS**

5910 #include <complex.h>

5911 double complex ccos(double complex z);

5912 float complex ccosf(float complex z);

5913 long double complex ccosl(long double complex z);

5914 **DESCRIPTION**5915 cx The functionality described on this reference page is aligned with the ISO C standard. Any
5916 conflict between the requirements described here and the ISO C standard is unintentional. This
5917 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5918 These functions shall compute the complex cosine of z.

5919 **RETURN VALUE**

5920 These functions shall return the complex cosine value.

5921 **ERRORS**

5922 No errors are defined.

5923 **EXAMPLES**

5924 None.

5925 **APPLICATION USAGE**

5926 None.

5927 **RATIONALE**

5928 None.

5929 **FUTURE DIRECTIONS**

5930 None.

5931 **SEE ALSO**5932 *ccos()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>5933 **CHANGE HISTORY**

5934 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5935 **NAME**5936 `ccosh`, `ccoshf`, `ccoshl` — complex hyperbolic cosine functions5937 **SYNOPSIS**5938 `#include <complex.h>`5939 `double complex ccosh(double complex z);`5940 `float complex ccoshf(float complex z);`5941 `long double complex ccoshl(long double complex z);`5942 **DESCRIPTION**

5943 `CX` The functionality described on this reference page is aligned with the ISO C standard. Any
5944 conflict between the requirements described here and the ISO C standard is unintentional. This
5945 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5946 These functions shall compute the complex hyperbolic cosine of z .5947 **RETURN VALUE**

5948 These functions shall return the complex hyperbolic cosine value.

5949 **ERRORS**

5950 No errors are defined.

5951 **EXAMPLES**

5952 None.

5953 **APPLICATION USAGE**

5954 None.

5955 **RATIONALE**

5956 None.

5957 **FUTURE DIRECTIONS**

5958 None.

5959 **SEE ALSO**5960 `cacosh()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<complex.h>`5961 **CHANGE HISTORY**

5962 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5963 **NAME**5964 `ccosl` — complex cosine functions5965 **SYNOPSIS**5966 `#include <complex.h>`5967 `long double complex ccosl(long double complex z);`5968 **DESCRIPTION**5969 Refer to `ccos()`.

5970 **NAME**

5971 ceil, ceilf, ceill — ceiling value function

5972 **SYNOPSIS**

5973 #include <math.h>

5974 double ceil(double x);

5975 float ceilf(float x);

5976 long double ceill(long double x);

5977 **DESCRIPTION**

5978 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 5979 conflict between the requirements described here and the ISO C standard is unintentional. This
 5980 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

5981 These functions shall compute the smallest integral value not less than *x*.

5982 An application wishing to check for error situations should set *errno* to zero and call
 5983 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 5984 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 5985 zero, an error has occurred.

5986 **RETURN VALUE**

5987 Upon successful completion, *ceil()*, *ceilf()*, and *ceill()* shall return the smallest integral value not
 5988 less than *x*, expressed as a type **double**, **float**, or **long double**, respectively.

5989 **MX** If *x* is NaN, a NaN shall be returned.5990 If *x* is ± 0 or $\pm \text{Inf}$, *x* shall be returned.

5991 **XSI** If the correct value would cause overflow, a range error shall occur and *ceil()*, *ceilf()*, and *ceill()*
 5992 shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

5993 **ERRORS**

5994 These functions shall fail if:

5995 **XSI** **Range Error** The result overflows.

5996 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 5997 then *errno* shall be set to [ERANGE]. If the integer expression
 5998 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 5999 floating-point exception shall be raised.

6000 **EXAMPLES**

6001 None.

6002 **APPLICATION USAGE**

6003 The integral value returned by these functions need not be expressible as an **int** or **long**. The
 6004 return value should be tested before assigning it to an integer type to avoid the undefined results
 6005 of an integer overflow.

6006 The *ceil()* function can only overflow when the floating-point representation has
 6007 DBL_MANT_DIG > DBL_MAX_EXP.

6008 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 6009 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

6010 **RATIONALE**

6011 None.

6012 **FUTURE DIRECTIONS**

6013 None.

6014 **SEE ALSO**6015 *feclearexcept()*, *fetestexcept()*, *floor()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,
6016 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>6017 **CHANGE HISTORY**

6018 First released in Issue 1. Derived from Issue 1 of the SVID.

6019 **Issue 5**6020 The DESCRIPTION is updated to indicate how an application should check for an error. This
6021 text was previously published in the APPLICATION USAGE section.6022 **Issue 6**6023 The *ceilf()* and *ceilll()* functions are added for alignment with the ISO/IEC 9899:1999 standard.6024 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
6025 revised to align with the ISO/IEC 9899:1999 standard.6026 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
6027 marked.

6028 **NAME**

6029 cexp, cexpf, cexpl — complex exponential functions

6030 **SYNOPSIS**

6031 #include <complex.h>

6032 double complex cexp(double complex z);

6033 float complex cexpf(float complex z);

6034 long double complex cexpl(long double complex z);

6035 **DESCRIPTION**6036 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
6037 conflict between the requirements described here and the ISO C standard is unintentional. This
6038 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.6039 These functions shall compute the complex exponent of z , defined as e^z .6040 **RETURN VALUE**6041 These functions shall return the complex exponential value of z .6042 **ERRORS**

6043 No errors are defined.

6044 **EXAMPLES**

6045 None.

6046 **APPLICATION USAGE**

6047 None.

6048 **RATIONALE**

6049 None.

6050 **FUTURE DIRECTIONS**

6051 None.

6052 **SEE ALSO**6053 *clog()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>6054 **CHANGE HISTORY**

6055 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

6056 NAME

6057 cfgetispeed — get input baud rate

6058 SYNOPSIS

6059 #include <termios.h>

6060 speed_t cfgetispeed(const struct termios *termios_p);

6061 DESCRIPTION

6062 The *cfgetispeed()* function shall extract the input baud rate from the **termios** structure to which
6063 the *termios_p* argument points.6064 This function shall return exactly the value in the **termios** data structure, without interpretation.

6065 RETURN VALUE

6066 Upon successful completion, *cfgetispeed()* shall return a value of type **speed_t** representing the
6067 input baud rate.

6068 ERRORS

6069 No errors are defined.

6070 EXAMPLES

6071 None.

6072 APPLICATION USAGE

6073 None.

6074 RATIONALE

6075 The term “baud” is used historically here, but is not technically correct. This is properly “bits
6076 per second”, which may not be the same as baud. However, the term is used because of the
6077 historical usage and understanding.6078 The *cfgetospeed()*, *cfgetispeed()*, *cfsetospeed()*, and *cfsetispeed()* functions do not take arguments as
6079 numbers, but rather as symbolic names. There are two reasons for this:

- 6080 1. Historically, numbers were not used because of the way the rate was stored in the data
-
- 6081 structure. This is retained even though a function is now used.
-
- 6082 2. More importantly, only a limited set of possible rates is at all portable, and this constrains
-
- 6083 the application to that set.

6084 There is nothing to prevent an implementation accepting as an extension a number (such as 126),
6085 and since the encoding of the Bxxx symbols is not specified, this can be done to avoid
6086 introducing ambiguity.6087 Setting the input baud rate to zero was a mechanism to allow for split baud rates. Clarifications
6088 in this volume of IEEE Std 1003.1-2001 have made it possible to determine whether split rates are
6089 supported and to support them without having to treat zero as a special case. Since this
6090 functionality is also confusing, it has been declared obsolescent. The 0 argument referred to is
6091 the literal constant 0, not the symbolic constant B0. This volume of IEEE Std 1003.1-2001 does not
6092 preclude B0 from being defined as the value 0; in fact, implementations would likely benefit
6093 from the two being equivalent. This volume of IEEE Std 1003.1-2001 does not fully specify
6094 whether the previous *cfsetispeed()* value is retained after a *tcgetattr()* as the actual value or as
6095 zero. Therefore, conforming applications should always set both the input speed and output
6096 speed when setting either.6097 In historical implementations, the baud rate information is traditionally kept in **c_cflag**.
6098 Applications should be written to presume that this might be the case (and thus not blindly copy
6099 **c_cflag**), but not to rely on it in case it is in some other field of the structure. Setting the **c_cflag**
6100 field absolutely after setting a baud rate is a non-portable action because of this. In general, the

6101 unused parts of the flag fields might be used by the implementation and should not be blindly
6102 copied from the descriptions of one terminal device to another.

6103 **FUTURE DIRECTIONS**

6104 None.

6105 **SEE ALSO**

6106 *cfgetospeed()*, *cfsetispeed()*, *cfsetospeed()*, *tcgetattr()*, the Base Definitions volume of
6107 IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, <termios.h>

6108 **CHANGE HISTORY**

6109 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

6110 **NAME**

6111 cfgetospeed — get output baud rate

6112 **SYNOPSIS**

6113 #include <termios.h>

6114 speed_t cfgetospeed(const struct termios *termios_p);

6115 **DESCRIPTION**

6116 The *cfgetospeed()* function shall extract the output baud rate from the **termios** structure to which
6117 the *termios_p* argument points.

6118 This function shall return exactly the value in the **termios** data structure, without interpretation.

6119 **RETURN VALUE**

6120 Upon successful completion, *cfgetospeed()* shall return a value of type **speed_t** representing the
6121 output baud rate.

6122 **ERRORS**

6123 No errors are defined.

6124 **EXAMPLES**

6125 None.

6126 **APPLICATION USAGE**

6127 None.

6128 **RATIONALE**

6129 Refer to *cfgetispeed()*.

6130 **FUTURE DIRECTIONS**

6131 None.

6132 **SEE ALSO**

6133 *cfgetispeed()*, *cfsetispeed()*, *cfsetospeed()*, *tcgetattr()*, the Base Definitions volume of
6134 IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, <**termios.h**>

6135 **CHANGE HISTORY**

6136 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

6137 **NAME**6138 `cfsetispeed` — set input baud rate6139 **SYNOPSIS**6140 `#include <termios.h>`6141 `int cfsetispeed(struct termios *termios_p, speed_t speed);`6142 **DESCRIPTION**6143 The `cfsetispeed()` function shall set the input baud rate stored in the structure pointed to by
6144 `termios_p` to `speed`.6145 There shall be no effect on the baud rates set in the hardware until a subsequent successful call
6146 to `tcsetattr()` with the same **termios** structure. Similarly, errors resulting from attempts to set
6147 baud rates not supported by the terminal device need not be detected until the `tcsetattr()`
6148 function is called.6149 **RETURN VALUE**6150 Upon successful completion, `cfsetispeed()` shall return 0; otherwise, -1 shall be returned, and
6151 `errno` may be set to indicate the error.6152 **ERRORS**6153 The `cfsetispeed()` function may fail if:6154 [EINVAL] The `speed` value is not a valid baud rate.6155 [EINVAL] The value of `speed` is outside the range of possible speed values as specified in
6156 `<termios.h>`.6157 **EXAMPLES**

6158 None.

6159 **APPLICATION USAGE**

6160 None.

6161 **RATIONALE**6162 Refer to `cfgetispeed()`.6163 **FUTURE DIRECTIONS**

6164 None.

6165 **SEE ALSO**6166 `cfgetispeed()`, `cfgetospeed()`, `cfsetospeed()`, `tcsetattr()`, the Base Definitions volume of
6167 IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, `<termios.h>`6168 **CHANGE HISTORY**

6169 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

6170 **Issue 6**6171 The following new requirements on POSIX implementations derive from alignment with the
6172 Single UNIX Specification:

- 6173
- The optional setting of `errno` and the [EINVAL] error conditions are added.

6174 **NAME**

6175 cfsetospeed — set output baud rate

6176 **SYNOPSIS**

6177 #include <termios.h>

6178 int cfsetospeed(struct termios *termios_p, speed_t speed);

6179 **DESCRIPTION**6180 The *cfsetospeed()* function shall set the output baud rate stored in the structure pointed to by
6181 *termios_p* to *speed*.6182 There shall be no effect on the baud rates set in the hardware until a subsequent successful call
6183 to *tcsetattr()* with the same **termios** structure. Similarly, errors resulting from attempts to set
6184 baud rates not supported by the terminal device need not be detected until the *tcsetattr()*
6185 function is called.6186 **RETURN VALUE**6187 Upon successful completion, *cfsetospeed()* shall return 0; otherwise, it shall return -1 and *errno*
6188 may be set to indicate the error.6189 **ERRORS**6190 The *cfsetospeed()* function may fail if:6191 [EINVAL] The *speed* value is not a valid baud rate.6192 [EINVAL] The value of *speed* is outside the range of possible speed values as specified in
6193 <**termios.h**>.6194 **EXAMPLES**

6195 None.

6196 **APPLICATION USAGE**

6197 None.

6198 **RATIONALE**6199 Refer to *cfgetispeed()*.6200 **FUTURE DIRECTIONS**

6201 None.

6202 **SEE ALSO**6203 *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, *tcsetattr()*, the Base Definitions volume of
6204 IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, <**termios.h**>6205 **CHANGE HISTORY**

6206 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

6207 **Issue 6**6208 The following new requirements on POSIX implementations derive from alignment with the
6209 Single UNIX Specification:

- 6210
- The optional setting of *errno* and the [EINVAL] error conditions are added.

6211 **NAME**

6212 chdir — change working directory

6213 **SYNOPSIS**

6214 #include <unistd.h>

6215 int chdir(const char *path);

6216 **DESCRIPTION**

6217 The *chdir()* function shall cause the directory named by the pathname pointed to by the *path*
6218 argument to become the current working directory; that is, the starting point for path searches
6219 for pathnames not beginning with *'/'*.

6220 **RETURN VALUE**

6221 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned, the current
6222 working directory shall remain unchanged, and *errno* shall be set to indicate the error.

6223 **ERRORS**6224 The *chdir()* function shall fail if:

6225 [EACCES] Search permission is denied for any component of the pathname.

6226 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
6227 argument.

6228 [ENAMETOOLONG]

6229 The length of the *path* argument exceeds {PATH_MAX} or a pathname
6230 component is longer than {NAME_MAX}.

6231 [ENOENT] A component of *path* does not name an existing directory or *path* is an empty
6232 string.

6233 [ENOTDIR] A component of the pathname is not a directory.

6234 The *chdir()* function may fail if:6235 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
6236 resolution of the *path* argument.

6237 [ENAMETOOLONG]

6238 As a result of encountering a symbolic link in resolution of the *path* argument,
6239 the length of the substituted pathname string exceeded {PATH_MAX}.

6240 **EXAMPLES**6241 **Changing the Current Working Directory**

6242 The following example makes the value pointed to by **directory**, **/tmp**, the current working
6243 directory.

6244 #include <unistd.h>

6245 ...

6246 char *directory = "/tmp";

6247 int ret;

6248 ret = chdir (directory);

6249 **APPLICATION USAGE**

6250 None.

6251 **RATIONALE**6252 The *chdir()* function only affects the working directory of the current process.6253 **FUTURE DIRECTIONS**

6254 None.

6255 **SEE ALSO**6256 *getcwd()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>6257 **CHANGE HISTORY**

6258 First released in Issue 1. Derived from Issue 1 of the SVID.

6259 **Issue 6**

6260 The APPLICATION USAGE section is added.

6261 The following new requirements on POSIX implementations derive from alignment with the
6262 Single UNIX Specification:

- 6263 • The [ELOOP] mandatory error condition is added.
- 6264 • A second [ENAMETOOLONG] is added as an optional error condition.

6265 The following changes were made to align with the IEEE P1003.1a draft standard:

- 6266 • The [ELOOP] optional error condition is added.

6267 **NAME**

6268 chmod — change mode of a file

6269 **SYNOPSIS**

6270 #include <sys/stat.h>

6271 int chmod(const char *path, mode_t mode);

6272 **DESCRIPTION**

6273 XSI The *chmod()* function shall change S_ISUID, S_ISGID, S_ISVTX, and the file permission bits of
 6274 the file named by the pathname pointed to by the *path* argument to the corresponding bits in the
 6275 *mode* argument. The application shall ensure that the effective user ID of the process matches the
 6276 owner of the file or the process has appropriate privileges in order to do this.

6277 XSI S_ISUID, S_ISGID, S_ISVTX, and the file permission bits are described in <sys/stat.h>.

6278 If the calling process does not have appropriate privileges, and if the group ID of the file does
 6279 not match the effective group ID or one of the supplementary group IDs and if the file is a
 6280 regular file, bit S_ISGID (set-group-ID on execution) in the file's mode shall be cleared upon
 6281 successful return from *chmod()*.

6282 Additional implementation-defined restrictions may cause the S_ISUID and S_ISGID bits in
 6283 *mode* to be ignored.

6284 The effect on file descriptors for files open at the time of a call to *chmod()* is implementation-
 6285 defined.

6286 Upon successful completion, *chmod()* shall mark for update the *st_ctime* field of the file.

6287 **RETURN VALUE**

6288 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
 6289 indicate the error. If -1 is returned, no change to the file mode occurs.

6290 **ERRORS**

6291 The *chmod()* function shall fail if:

6292 [EACCES] Search permission is denied on a component of the path prefix.

6293 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 6294 argument.

6295 [ENAMETOOLONG]

6296 The length of the *path* argument exceeds {PATH_MAX} or a pathname
 6297 component is longer than {NAME_MAX}.

6298 [ENOTDIR] A component of the path prefix is not a directory.

6299 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

6300 [EPERM] The effective user ID does not match the owner of the file and the process
 6301 does not have appropriate privileges.

6302 [EROFS] The named file resides on a read-only file system.

6303 The *chmod()* function may fail if:

6304 [EINTR] A signal was caught during execution of the function.

6305 [EINVAL] The value of the *mode* argument is invalid.

6306 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 6307 resolution of the *path* argument.

6308 [ENAMETOOLONG]
 6309 As a result of encountering a symbolic link in resolution of the *path* argument,
 6310 the length of the substituted pathname strings exceeded {PATH_MAX}.

6311 EXAMPLES

6312 Setting Read Permissions for User, Group, and Others

6313 The following example sets read permissions for the owner, group, and others.

```
6314 #include <sys/stat.h>
6315 const char *path;
6316 ...
6317 chmod(path, S_IRUSR|S_IRGRP|S_IROTH);
```

6318 Setting Read, Write, and Execute Permissions for the Owner Only

6319 The following example sets read, write, and execute permissions for the owner, and no
 6320 permissions for group and others.

```
6321 #include <sys/stat.h>
6322 const char *path;
6323 ...
6324 chmod(path, S_IRWXU);
```

6325 Setting Different Permissions for Owner, Group, and Other

6326 The following example sets owner permissions for CHANGEFILE to read, write, and execute,
 6327 group permissions to read and execute, and other permissions to read.

```
6328 #include <sys/stat.h>
6329 #define CHANGEFILE "/etc/myfile"
6330 ...
6331 chmod(CHANGEFILE, S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH);
```

6332 Setting and Checking File Permissions

6333 The following example sets the file permission bits for a file named */home/cnd/mod1*, then calls
 6334 the *stat()* function to verify the permissions.

```
6335 #include <sys/types.h>
6336 #include <sys/stat.h>
6337 int status;
6338 struct stat buffer
6339 ...
6340 chmod("home/cnd/mod1", S_IRWXU|S_IRWXG|S_IROTH|S_IWOTH);
6341 status = stat("home/cnd/mod1", &buffer);
```

6342 APPLICATION USAGE

6343 In order to ensure that the *S_ISUID* and *S_ISGID* bits are set, an application requiring this should
 6344 use *stat()* after a successful *chmod()* to verify this.

6345 Any file descriptors currently open by any process on the file could possibly become invalid if
 6346 the mode of the file is changed to a value which would deny access to that process. One

6347 situation where this could occur is on a stateless file system. This behavior will not occur in a
6348 conforming environment.

6349 RATIONALE

6350 This volume of IEEE Std 1003.1-2001 specifies that the S_ISGID bit is cleared by *chmod()* on a
6351 regular file under certain conditions. This is specified on the assumption that regular files may
6352 be executed, and the system should prevent users from making executable *setgid()* files perform
6353 with privileges that the caller does not have. On implementations that support execution of
6354 other file types, the S_ISGID bit should be cleared for those file types under the same
6355 circumstances.

6356 Implementations that use the S_ISUID bit to indicate some other function (for example,
6357 mandatory record locking) on non-executable files need not clear this bit on writing. They
6358 should clear the bit for executable files and any other cases where the bit grants special powers
6359 to processes that change the file contents. Similar comments apply to the S_ISGID bit.

6360 FUTURE DIRECTIONS

6361 None.

6362 SEE ALSO

6363 *chown()*, *mkdir()*, *mkfifo()*, *open()*, *stat()*, *statvfs()*, the Base Definitions volume of
6364 IEEE Std 1003.1-2001, `<sys/stat.h>`, `<sys/types.h>`

6365 CHANGE HISTORY

6366 First released in Issue 1. Derived from Issue 1 of the SVID.

6367 Issue 6

6368 The following new requirements on POSIX implementations derive from alignment with the
6369 Single UNIX Specification:

6370 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
6371 required for conforming implementations of previous POSIX specifications, it was not
6372 required for UNIX applications.

6373 • The [EINVAL] and [EINTR] optional error conditions are added.

6374 • A second [ENAMETOOLONG] is added as an optional error condition.

6375 The following changes were made to align with the IEEE P1003.1a draft standard:

6376 • The [ELOOP] optional error condition is added.

6377 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

6378 **NAME**

6379 chown — change owner and group of a file

6380 **SYNOPSIS**

6381 #include <unistd.h>

6382 int chown(const char *path, uid_t owner, gid_t group);

6383 **DESCRIPTION**6384 The *chown()* function shall change the user and group ownership of a file.6385 The *path* argument points to a pathname naming a file. The user ID and group ID of the named
6386 file shall be set to the numeric values contained in *owner* and *group*, respectively.6387 Only processes with an effective user ID equal to the user ID of the file or with appropriate
6388 privileges may change the ownership of a file. If `_POSIX_CHOWN_RESTRICTED` is in effect for
6389 *path*:

- 6390
- Changing the user ID is restricted to processes with appropriate privileges.
 - Changing the group ID is permitted to a process with an effective user ID equal to the user
6391 ID of the file, but without appropriate privileges, if and only if *owner* is equal to the file's user
6392 ID or `(uid_t)-1` and *group* is equal either to the calling process' effective group ID or to one of
6393 its supplementary group IDs.
- 6394

6395 If the specified file is a regular file, one or more of the `S_IXUSR`, `S_IXGRP`, or `S_IXOTH` bits of
6396 the file mode are set, and the process does not have appropriate privileges, the set-user-ID
6397 (`S_ISUID`) and set-group-ID (`S_ISGID`) bits of the file mode shall be cleared upon successful
6398 return from *chown()*. If the specified file is a regular file, one or more of the `S_IXUSR`, `S_IXGRP`,
6399 or `S_IXOTH` bits of the file mode are set, and the process has appropriate privileges, it is
6400 implementation-defined whether the set-user-ID and set-group-ID bits are altered. If the *chown()*
6401 function is successfully invoked on a file that is not a regular file and one or more of the
6402 `S_IXUSR`, `S_IXGRP`, or `S_IXOTH` bits of the file mode are set, the set-user-ID and set-group-ID
6403 bits may be cleared.6404 If *owner* or *group* is specified as `(uid_t)-1` or `(gid_t)-1`, respectively, the corresponding ID of the
6405 file shall not be changed. If both owner and group are `-1`, the times need not be updated.6406 Upon successful completion, *chown()* shall mark for update the *st_ctime* field of the file.6407 **RETURN VALUE**6408 Upon successful completion, 0 shall be returned; otherwise, `-1` shall be returned and *errno* set to
6409 indicate the error. If `-1` is returned, no changes are made in the user ID and group ID of the file.6410 **ERRORS**6411 The *chown()* function shall fail if:

- 6412 [EACCES] Search permission is denied on a component of the path prefix.
-
- 6413 [ELOOP] A loop exists in symbolic links encountered during resolution of the
- path*
-
- 6414 argument.
-
- 6415 [ENAMETOOLONG] The length of the
- path*
- argument exceeds
- `{PATH_MAX}`
- or a pathname
-
- 6416 component is longer than
- `{NAME_MAX}`
- .
-
- 6417 [ENOTDIR] A component of the path prefix is not a directory.
-
- 6418 [ENOENT] A component of
- path*
- does not name an existing file or
- path*
- is an empty string.
-
- 6419

6420 [EPERM] The effective user ID does not match the owner of the file, or the calling
 6421 process does not have appropriate privileges and
 6422 `_POSIX_CHOWN_RESTRICTED` indicates that such privilege is required.

6423 [EROFS] The named file resides on a read-only file system.

6424 The `chown()` function may fail if:

6425 [EIO] An I/O error occurred while reading or writing to the file system.

6426 [EINTR] The `chown()` function was interrupted by a signal which was caught.

6427 [EINVAL] The owner or group ID supplied is not a value supported by the
 6428 implementation.

6429 [ELOOP] More than `{SYMLOOP_MAX}` symbolic links were encountered during
 6430 resolution of the *path* argument.

6431 [ENAMETOOLONG]

6432 As a result of encountering a symbolic link in resolution of the *path* argument,
 6433 the length of the substituted pathname string exceeded `{PATH_MAX}`.

6434 EXAMPLES

6435 None.

6436 APPLICATION USAGE

6437 Although `chown()` can be used on some implementations by the file owner to change the owner
 6438 and group to any desired values, the only portable use of this function is to change the group of
 6439 a file to the effective GID of the calling process or to a member of its group set.

6440 RATIONALE

6441 System III and System V allow a user to give away files; that is, the owner of a file may change
 6442 its user ID to anything. This is a serious problem for implementations that are intended to meet
 6443 government security regulations. Version 7 and 4.3 BSD permit only the superuser to change the
 6444 user ID of a file. Some government agencies (usually not ones concerned directly with security)
 6445 find this limitation too confining. This volume of IEEE Std 1003.1-2001 uses *may* to permit secure
 6446 implementations while not disallowing System V.

6447 System III and System V allow the owner of a file to change the group ID to anything. Version 7
 6448 permits only the superuser to change the group ID of a file. 4.3 BSD permits the owner to
 6449 change the group ID of a file to its effective group ID or to any of the groups in the list of
 6450 supplementary group IDs, but to no others.

6451 The POSIX.1-1990 standard requires that the `chown()` function invoked by a non-appropriate
 6452 privileged process clear the `S_ISGID` and the `S_ISUID` bits for regular files, and permits them to
 6453 be cleared for other types of files. This is so that changes in accessibility do not accidentally
 6454 cause files to become security holes. Unfortunately, requiring these bits to be cleared on non-
 6455 executable data files also clears the mandatory file locking bit (shared with `S_ISGID`), which is
 6456 an extension on many implementations (it first appeared in System V). These bits should only be
 6457 required to be cleared on regular files that have one or more of their execute bits set.

6458 FUTURE DIRECTIONS

6459 None.

6460 SEE ALSO

6461 `chmod()`, `pathconf()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<sys/types.h>`,
 6462 `<unistd.h>`

6463 **CHANGE HISTORY**

6464 First released in Issue 1. Derived from Issue 1 of the SVID.

6465 **Issue 6**

6466 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 6467 • The wording describing the optional dependency on `_POSIX_CHOWN_RESTRICTED` is
6468 restored.
- 6469 • The `[EPERM]` error is restored as an error dependent on `_POSIX_CHOWN_RESTRICTED`.
6470 This is since its operand is a pathname and applications should be aware that the error may
6471 not occur for that pathname if the file system does not support
6472 `_POSIX_CHOWN_RESTRICTED`.

6473 The following new requirements on POSIX implementations derive from alignment with the
6474 Single UNIX Specification:

- 6475 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
6476 required for conforming implementations of previous POSIX specifications, it was not
6477 required for UNIX applications.
- 6478 • The value for *owner* of `(uid_t)-1` allows the use of `-1` by the owner of a file to change the
6479 group ID only. A corresponding change is made for group.
- 6480 • The `[ELOOP]` mandatory error condition is added.
- 6481 • The `[EIO]` and `[EINTR]` optional error conditions are added.
- 6482 • A second `[ENAMETOOLONG]` is added as an optional error condition.

6483 The following changes were made to align with the IEEE P1003.1a draft standard:

- 6484 • Clarification is added that the `S_ISUID` and `S_ISGID` bits do not need to be cleared when the
6485 process has appropriate privileges.
- 6486 • The `[ELOOP]` optional error condition is added.

6487 **NAME**

6488 cimag, cimagf, cimagl — complex imaginary functions

6489 **SYNOPSIS**

6490 #include <complex.h>

6491 double cimag(double complex z);

6492 float cimagf(float complex z);

6493 long double cimagl(long double complex z);

6494 **DESCRIPTION**

6495 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
6496 conflict between the requirements described here and the ISO C standard is unintentional. This
6497 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

6498 These functions shall compute the imaginary part of *z*.6499 **RETURN VALUE**

6500 These functions shall return the imaginary part value (as a real).

6501 **ERRORS**

6502 No errors are defined.

6503 **EXAMPLES**

6504 None.

6505 **APPLICATION USAGE**6506 For a variable *z* of complex type:6507 `z == creal(z) + cimag(z)*I`6508 **RATIONALE**

6509 None.

6510 **FUTURE DIRECTIONS**

6511 None.

6512 **SEE ALSO**6513 *carg()*, *conj()*, *cproj()*, *creal()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>6514 **CHANGE HISTORY**

6515 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

6516 **NAME**

6517 clearerr — clear indicators on a stream

6518 **SYNOPSIS**

6519 #include <stdio.h>

6520 void clearerr(FILE *stream);

6521 **DESCRIPTION**

6522 cx The functionality described on this reference page is aligned with the ISO C standard. Any
6523 conflict between the requirements described here and the ISO C standard is unintentional. This
6524 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

6525 The *clearerr()* function shall clear the end-of-file and error indicators for the stream to which
6526 *stream* points.

6527 **RETURN VALUE**

6528 The *clearerr()* function shall not return a value.

6529 **ERRORS**

6530 No errors are defined.

6531 **EXAMPLES**

6532 None.

6533 **APPLICATION USAGE**

6534 None.

6535 **RATIONALE**

6536 None.

6537 **FUTURE DIRECTIONS**

6538 None.

6539 **SEE ALSO**

6540 The Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

6541 **CHANGE HISTORY**

6542 First released in Issue 1. Derived from Issue 1 of the SVID.

6543 **NAME**

6544 clock — report CPU time used

6545 **SYNOPSIS**

6546 #include <time.h>

6547 clock_t clock(void);

6548 **DESCRIPTION**

6549 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
6550 conflict between the requirements described here and the ISO C standard is unintentional. This
6551 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

6552 The *clock()* function shall return the implementation's best approximation to the processor time
6553 used by the process since the beginning of an implementation-defined era related only to the
6554 process invocation.

6555 **RETURN VALUE**

6556 To determine the time in seconds, the value returned by *clock()* should be divided by the value
6557 **XSI** of the macro `CLOCKS_PER_SEC`. `CLOCKS_PER_SEC` is defined to be one million in `<time.h>`.
6558 If the processor time used is not available or its value cannot be represented, the function shall
6559 return the value `(clock_t)-1`.

6560 **ERRORS**

6561 No errors are defined.

6562 **EXAMPLES**

6563 None.

6564 **APPLICATION USAGE**

6565 In order to measure the time spent in a program, *clock()* should be called at the start of the
6566 program and its return value subtracted from the value returned by subsequent calls. The value
6567 returned by *clock()* is defined for compatibility across systems that have clocks with different
6568 resolutions. The resolution on any particular system need not be to microsecond accuracy.

6569 The value returned by *clock()* may wrap around on some implementations. For example, on a
6570 machine with 32-bit values for `clock_t`, it wraps after 2 147 seconds or 36 minutes.

6571 **RATIONALE**

6572 None.

6573 **FUTURE DIRECTIONS**

6574 None.

6575 **SEE ALSO**

6576 *asctime()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*,
6577 the Base Definitions volume of IEEE Std 1003.1-2001, `<time.h>`

6578 **CHANGE HISTORY**

6579 First released in Issue 1. Derived from Issue 1 of the SVID.

6580 **NAME**6581 clock_getcpuclockid — access a process CPU-time clock (**ADVANCED REALTIME**)6582 **SYNOPSIS**

6583 CPT #include <time.h>

6584 int clock_getcpuclockid(pid_t pid, clockid_t *clock_id);

6585

6586 **DESCRIPTION**

6587 The *clock_getcpuclockid()* function shall return the clock ID of the CPU-time clock of the process
6588 specified by *pid*. If the process described by *pid* exists and the calling process has permission,
6589 the clock ID of this clock shall be returned in *clock_id*.

6590 If *pid* is zero, the *clock_getcpuclockid()* function shall return the clock ID of the CPU-time clock of
6591 the process making the call, in *clock_id*.

6592 The conditions under which one process has permission to obtain the CPU-time clock ID of
6593 other processes are implementation-defined.

6594 **RETURN VALUE**

6595 Upon successful completion, *clock_getcpuclockid()* shall return zero; otherwise, an error number
6596 shall be returned to indicate the error.

6597 **ERRORS**

6598 The *clock_getcpuclockid()* function shall fail if:

6599 [EPERM] The requesting process does not have permission to access the CPU-time
6600 clock for the process.

6601 The *clock_getcpuclockid()* function may fail if:

6602 [ESRCH] No process can be found corresponding to the process specified by *pid*.

6603 **EXAMPLES**

6604 None.

6605 **APPLICATION USAGE**

6606 The *clock_getcpuclockid()* function is part of the Process CPU-Time Clocks option and need not
6607 be provided on all implementations.

6608 **RATIONALE**

6609 None.

6610 **FUTURE DIRECTIONS**

6611 None.

6612 **SEE ALSO**

6613 *clock_getres()*, *timer_create()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**time.h**>

6614 **CHANGE HISTORY**

6615 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

6616 In the SYNOPSIS, the inclusion of <**sys/types.h**> is no longer required.

6617 **NAME**6618 clock_getres, clock_gettime, clock_settime — clock and timer functions (**REALTIME**)6619 **SYNOPSIS**

6620 TMR #include <time.h>

```
6621 int clock_getres(clockid_t clock_id, struct timespec *res);
6622 int clock_gettime(clockid_t clock_id, struct timespec *tp);
6623 int clock_settime(clockid_t clock_id, const struct timespec *tp);
6624
```

6625 **DESCRIPTION**

6626 The *clock_getres()* function shall return the resolution of any clock. Clock resolutions are
 6627 implementation-defined and cannot be set by a process. If the argument *res* is not NULL, the
 6628 resolution of the specified clock shall be stored in the location pointed to by *res*. If *res* is NULL,
 6629 the clock resolution is not returned. If the *time* argument of *clock_settime()* is not a multiple of *res*,
 6630 then the value is truncated to a multiple of *res*.

6631 The *clock_gettime()* function shall return the current value *tp* for the specified clock, *clock_id*.

6632 The *clock_settime()* function shall set the specified clock, *clock_id*, to the value specified by *tp*.
 6633 Time values that are between two consecutive non-negative integer multiples of the resolution
 6634 of the specified clock shall be truncated down to the smaller multiple of the resolution.

6635 A clock may be system-wide (that is, visible to all processes) or per-process (measuring time that
 6636 is meaningful only within a process). All implementations shall support a *clock_id* of
 6637 CLOCK_REALTIME as defined in <time.h>. This clock represents the realtime clock for the
 6638 system. For this clock, the values returned by *clock_gettime()* and specified by *clock_settime()*
 6639 represent the amount of time (in seconds and nanoseconds) since the Epoch. An implementation
 6640 may also support additional clocks. The interpretation of time values for these clocks is
 6641 unspecified.

6642 If the value of the CLOCK_REALTIME clock is set via *clock_settime()*, the new value of the clock
 6643 shall be used to determine the time of expiration for absolute time services based upon the
 6644 CLOCK_REALTIME clock. This applies to the time at which armed absolute timers expire. If the
 6645 absolute time requested at the invocation of such a time service is before the new value of the
 6646 clock, the time service shall expire immediately as if the clock had reached the requested time
 6647 normally.

6648 Setting the value of the CLOCK_REALTIME clock via *clock_settime()* shall have no effect on
 6649 threads that are blocked waiting for a relative time service based upon this clock, including the
 6650 *nanosleep()* function; nor on the expiration of relative timers based upon this clock.
 6651 Consequently, these time services shall expire when the requested relative interval elapses,
 6652 independently of the new or old value of the clock.

6653 MON If the Monotonic Clock option is supported, all implementations shall support a *clock_id* of
 6654 CLOCK_MONOTONIC defined in <time.h>. This clock represents the monotonic clock for the
 6655 system. For this clock, the value returned by *clock_gettime()* represents the amount of time (in
 6656 seconds and nanoseconds) since an unspecified point in the past (for example, system start-up
 6657 time, or the Epoch). This point does not change after system start-up time. The value of the
 6658 CLOCK_MONOTONIC clock cannot be set via *clock_settime()*. This function shall fail if it is
 6659 invoked with a *clock_id* argument of CLOCK_MONOTONIC.

6660 The effect of setting a clock via *clock_settime()* on armed per-process timers associated with a
 6661 clock other than CLOCK_REALTIME is implementation-defined.

6662 CS If the value of the CLOCK_REALTIME clock is set via *clock_settime()*, the new value of the clock
 6663 shall be used to determine the time at which the system shall awaken a thread blocked on an

- 6664 absolute *clock_nanosleep()* call based upon the CLOCK_REALTIME clock. If the absolute time
 6665 requested at the invocation of such a time service is before the new value of the clock, the call
 6666 shall return immediately as if the clock had reached the requested time normally.
- 6667 Setting the value of the CLOCK_REALTIME clock via *clock_settime()* shall have no effect on any
 6668 thread that is blocked on a relative *clock_nanosleep()* call. Consequently, the call shall return
 6669 when the requested relative interval elapses, independently of the new or old value of the clock.
- 6670 The appropriate privilege to set a particular clock is implementation-defined.
- 6671 CPT If `_POSIX_CPUTIME` is defined, implementations shall support clock ID values obtained by
 6672 invoking *clock_getcpuclockid()*, which represent the CPU-time clock of a given process.
 6673 Implementations shall also support the special `clockid_t` value
 6674 `CLOCK_PROCESS_CPUTIME_ID`, which represents the CPU-time clock of the calling process
 6675 when invoking one of the *clock_**() or *timer_**() functions. For these clock IDs, the values
 6676 returned by *clock_gettime()* and specified by *clock_settime()* represent the amount of execution
 6677 time of the process associated with the clock. Changing the value of a CPU-time clock via
 6678 *clock_settime()* shall have no effect on the behavior of the sporadic server scheduling policy (see
 6679 **Scheduling Policies** (on page 44)).
- 6680 TCT If `_POSIX_THREAD_CPUTIME` is defined, implementations shall support clock ID values
 6681 obtained by invoking *pthread_getcpuclockid()*, which represent the CPU-time clock of a given
 6682 thread. Implementations shall also support the special `clockid_t` value
 6683 `CLOCK_THREAD_CPUTIME_ID`, which represents the CPU-time clock of the calling thread
 6684 when invoking one of the *clock_**() or *timer_**() functions. For these clock IDs, the values
 6685 returned by *clock_gettime()* and specified by *clock_settime()* shall represent the amount of
 6686 execution time of the thread associated with the clock. Changing the value of a CPU-time clock
 6687 via *clock_settime()* shall have no effect on the behavior of the sporadic server scheduling policy
 6688 (see **Scheduling Policies** (on page 44)).
- 6689 **RETURN VALUE**
- 6690 A return value of 0 shall indicate that the call succeeded. A return value of -1 shall indicate that
 6691 an error occurred, and *errno* shall be set to indicate the error.
- 6692 **ERRORS**
- 6693 The *clock_getres()*, *clock_gettime()*, and *clock_settime()* functions shall fail if:
- 6694 [EINVAL] The *clock_id* argument does not specify a known clock.
- 6695 The *clock_settime()* function shall fail if:
- 6696 [EINVAL] The *tp* argument to *clock_settime()* is outside the range for the given clock ID.
- 6697 [EINVAL] The *tp* argument specified a nanosecond value less than zero or greater than
 6698 or equal to 1 000 million.
- 6699 MON [EINVAL] The value of the *clock_id* argument is `CLOCK_MONOTONIC`.
- 6700 The *clock_settime()* function may fail if:
- 6701 [EPERM] The requesting process does not have the appropriate privilege to set the
 6702 specified clock.

6703 **EXAMPLES**

6704 None.

6705 **APPLICATION USAGE**

6706 These functions are part of the Timers option and need not be available on all implementations.

6707 Note that the absolute value of the monotonic clock is meaningless (because its origin is
 6708 arbitrary), and thus there is no need to set it. Furthermore, realtime applications can rely on the
 6709 fact that the value of this clock is never set and, therefore, that time intervals measured with this
 6710 clock will not be affected by calls to *clock_settime()*.

6711 **RATIONALE**

6712 None.

6713 **FUTURE DIRECTIONS**

6714 None.

6715 **SEE ALSO**

6716 *clock_getcpuclockid()*, *clock_nanosleep()*, *ctime()*, *mq_timedreceive()*, *mq_timedsend()*, *nanosleep()*,
 6717 *pthread_mutex_timedlock()*, *sem_timedwait()*, *time()*, *timer_create()*, *timer_getoverrun()*, the Base
 6718 Definitions volume of IEEE Std 1003.1-2001, <**time.h**>

6719 **CHANGE HISTORY**

6720 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

6721 **Issue 6**

6722 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 6723 implementation does not support the Timers option.

6724 The APPLICATION USAGE section is added.

6725 The following changes were made to align with the IEEE P1003.1a draft standard:

- 6726 • Clarification is added of the effect of resetting the clock resolution.

6727 CPU-time clocks and the *clock_getcpuclockid()* function are added for alignment with
 6728 IEEE Std 1003.1d-1999.

6729 The following changes are added for alignment with IEEE Std 1003.1j-2000:

- 6730 • The DESCRIPTION is updated as follows:

- 6731 — The value returned by *clock_gettime()* for CLOCK_MONOTONIC is specified.

- 6732 — The *clock_settime()* function failing for CLOCK_MONOTONIC is specified.

- 6733 — The effects of *clock_settime()* on the *clock_nanosleep()* function with respect to
 6734 CLOCK_REALTIME are specified.

- 6735 • An [EINVAL] error is added to the ERRORS section, indicating that *clock_settime()* fails for
 6736 CLOCK_MONOTONIC.

- 6737 • The APPLICATION USAGE section notes that the CLOCK_MONOTONIC clock need not
 6738 and shall not be set by *clock_settime()* since the absolute value of the CLOCK_MONOTONIC
 6739 clock is meaningless.

- 6740 • The *clock_nanosleep()*, *mq_timedreceive()*, *mq_timedsend()*, *pthread_mutex_timedlock()*,
 6741 *sem_timedwait()*, *timer_create()*, and *timer_settime()* functions are added to the SEE ALSO
 6742 section.

6743 NAME

6744 clock_nanosleep — high resolution sleep with specifiable clock (**ADVANCED REALTIME**)

6745 SYNOPSIS

```
6746 cs #include <time.h>
6747 int clock_nanosleep(clockid_t clock_id, int flags,
6748     const struct timespec *rqtp, struct timespec *rmtp);
6749
```

6750 DESCRIPTION

6751 If the flag `TIMER_ABSTIME` is not set in the *flags* argument, the `clock_nanosleep()` function shall
 6752 cause the current thread to be suspended from execution until either the time interval specified
 6753 by the *rqtp* argument has elapsed, or a signal is delivered to the calling thread and its action is to
 6754 invoke a signal-catching function, or the process is terminated. The clock used to measure the
 6755 time shall be the clock specified by *clock_id*.

6756 If the flag `TIMER_ABSTIME` is set in the *flags* argument, the `clock_nanosleep()` function shall
 6757 cause the current thread to be suspended from execution until either the time value of the clock
 6758 specified by *clock_id* reaches the absolute time specified by the *rqtp* argument, or a signal is
 6759 delivered to the calling thread and its action is to invoke a signal-catching function, or the
 6760 process is terminated. If, at the time of the call, the time value specified by *rqtp* is less than or
 6761 equal to the time value of the specified clock, then `clock_nanosleep()` shall return immediately
 6762 and the calling process shall not be suspended.

6763 The suspension time caused by this function may be longer than requested because the
 6764 argument value is rounded up to an integer multiple of the sleep resolution, or because of the
 6765 scheduling of other activity by the system. But, except for the case of being interrupted by a
 6766 signal, the suspension time for the relative `clock_nanosleep()` function (that is, with the
 6767 `TIMER_ABSTIME` flag not set) shall not be less than the time interval specified by *rqtp*, as
 6768 measured by the corresponding clock. The suspension for the absolute `clock_nanosleep()` function
 6769 (that is, with the `TIMER_ABSTIME` flag set) shall be in effect at least until the value of the
 6770 corresponding clock reaches the absolute time specified by *rqtp*, except for the case of being
 6771 interrupted by a signal.

6772 The use of the `clock_nanosleep()` function shall have no effect on the action or blockage of any
 6773 signal.

6774 The `clock_nanosleep()` function shall fail if the *clock_id* argument refers to the CPU-time clock of
 6775 the calling thread. It is unspecified whether *clock_id* values of other CPU-time clocks are
 6776 allowed.

6777 RETURN VALUE

6778 If the `clock_nanosleep()` function returns because the requested time has elapsed, its return value
 6779 shall be zero.

6780 If the `clock_nanosleep()` function returns because it has been interrupted by a signal, it shall return
 6781 the corresponding error value. For the relative `clock_nanosleep()` function, if the *rmtp* argument is
 6782 non-NULL, the **timespec** structure referenced by it shall be updated to contain the amount of
 6783 time remaining in the interval (the requested time minus the time actually slept). If the *rmtp*
 6784 argument is NULL, the remaining time is not returned. The absolute `clock_nanosleep()` function
 6785 has no effect on the structure referenced by *rmtp*.

6786 If `clock_nanosleep()` fails, it shall return the corresponding error value.

6787 **ERRORS**6788 The `clock_nanosleep()` function shall fail if:

- 6789 [EINTR] The `clock_nanosleep()` function was interrupted by a signal.
- 6790 [EINVAL] The `rqt` argument specified a nanosecond value less than zero or greater than
6791 or equal to 1 000 million; or the `TIMER_ABSTIME` flag was specified in `flags`
6792 and the `rqt` argument is outside the range for the clock specified by `clock_id`;
6793 or the `clock_id` argument does not specify a known clock, or specifies the
6794 CPU-time clock of the calling thread.
- 6795 [ENOTSUP] The `clock_id` argument specifies a clock for which `clock_nanosleep()` is not
6796 supported, such as a CPU-time clock.

6797 **EXAMPLES**

6798 None.

6799 **APPLICATION USAGE**

6800 Calling `clock_nanosleep()` with the value `TIMER_ABSTIME` not set in the `flags` argument and with
6801 a `clock_id` of `CLOCK_REALTIME` is equivalent to calling `nanosleep()` with the same `rqt` and `rmt`
6802 arguments.

6803 **RATIONALE**

6804 The `nanosleep()` function specifies that the system-wide clock `CLOCK_REALTIME` is used to
6805 measure the elapsed time for this time service. However, with the introduction of the monotonic
6806 clock `CLOCK_MONOTONIC` a new relative sleep function is needed to allow an application to
6807 take advantage of the special characteristics of this clock.

6808 There are many applications in which a process needs to be suspended and then activated
6809 multiple times in a periodic way; for example, to poll the status of a non-interrupting device or
6810 to refresh a display device. For these cases, it is known that precise periodic activation cannot be
6811 achieved with a relative `sleep()` or `nanosleep()` function call. Suppose, for example, a periodic
6812 process that is activated at time T_0 , executes for a while, and then wants to suspend itself until
6813 time T_0+T , the period being T . If this process wants to use the `nanosleep()` function, it must first
6814 call `clock_gettime()` to get the current time, then calculate the difference between the current time
6815 and T_0+T and, finally, call `nanosleep()` using the computed interval. However, the process could
6816 be preempted by a different process between the two function calls, and in this case the interval
6817 computed would be wrong; the process would wake up later than desired. This problem would
6818 not occur with the absolute `clock_nanosleep()` function, since only one function call would be
6819 necessary to suspend the process until the desired time. In other cases, however, a relative sleep
6820 is needed, and that is why both functionalities are required.

6821 Although it is possible to implement periodic processes using the timers interface, this
6822 implementation would require the use of signals, and the reservation of some signal numbers. In
6823 this regard, the reasons for including an absolute version of the `clock_nanosleep()` function in
6824 IEEE Std 1003.1-2001 are the same as for the inclusion of the relative `nanosleep()`.

6825 It is also possible to implement precise periodic processes using `pthread_cond_timedwait()`, in
6826 which an absolute timeout is specified that takes effect if the condition variable involved is
6827 never signaled. However, the use of this interface is unnatural, and involves performing other
6828 operations on mutexes and condition variables that imply an unnecessary overhead.
6829 Furthermore, `pthread_cond_timedwait()` is not available in implementations that do not support
6830 threads.

6831 Although the interface of the relative and absolute versions of the new high resolution sleep
6832 service is the same `clock_nanosleep()` function, the `rmt` argument is only used in the relative
6833 sleep. This argument is needed in the relative `clock_nanosleep()` function to reissue the function

6834 call if it is interrupted by a signal, but it is not needed in the absolute *clock_nanosleep()* function
6835 call; if the call is interrupted by a signal, the absolute *clock_nanosleep()* function can be invoked
6836 again with the same *rqt* argument used in the interrupted call.

6837 **FUTURE DIRECTIONS**

6838 None.

6839 **SEE ALSO**

6840 *clock_getres()*, *nanosleep()*, *pthread_cond_timedwait()*, *sleep()*, the Base Definitions volume of
6841 IEEE Std 1003.1-2001, <**time.h**>

6842 **CHANGE HISTORY**

6843 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

6844 **NAME**6845 clock_settime — clock and timer functions (**REALTIME**)6846 **SYNOPSIS**

6847 TMR #include <time.h>

6848 int clock_settime(clockid_t *clock_id*, const struct timespec **tp*);

6849

6850 **DESCRIPTION**6851 Refer to *clock_getres()*.

6852 NAME

6853 clog, clogf, clogl — complex natural logarithm functions

6854 SYNOPSIS

6855 #include <complex.h>

6856 double complex clog(double complex z);

6857 float complex clogf(float complex z);

6858 long double complex clogl(long double complex z);

6859 DESCRIPTION

6860 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
6861 conflict between the requirements described here and the ISO C standard is unintentional. This
6862 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

6863 These functions shall compute the complex natural (base *e*) logarithm of *z*, with a branch cut
6864 along the negative real axis.

6865 RETURN VALUE

6866 These functions shall return the complex natural logarithm value, in the range of a strip
6867 mathematically unbounded along the real axis and in the interval $[-i\pi, +i\pi]$ along the imaginary
6868 axis.

6869 ERRORS

6870 No errors are defined.

6871 EXAMPLES

6872 None.

6873 APPLICATION USAGE

6874 None.

6875 RATIONALE

6876 None.

6877 FUTURE DIRECTIONS

6878 None.

6879 SEE ALSO

6880 *cexp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>

6881 CHANGE HISTORY

6882 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

6883 **NAME**

6884 close — close a file descriptor

6885 **SYNOPSIS**

6886 #include <unistd.h>

6887 int close(int *fildes*);6888 **DESCRIPTION**

6889 The *close()* function shall deallocate the file descriptor indicated by *fildes*. To deallocate means
 6890 to make the file descriptor available for return by subsequent calls to *open()* or other functions
 6891 that allocate file descriptors. All outstanding record locks owned by the process on the file
 6892 associated with the file descriptor shall be removed (that is, unlocked).

6893 If *close()* is interrupted by a signal that is to be caught, it shall return -1 with *errno* set to [EINTR]
 6894 and the state of *fildes* is unspecified. If an I/O error occurred while reading from or writing to the
 6895 file system during *close()*, it may return -1 with *errno* set to [EIO]; if this error is returned, the
 6896 state of *fildes* is unspecified.

6897 When all file descriptors associated with a pipe or FIFO special file are closed, any data
 6898 remaining in the pipe or FIFO shall be discarded.

6899 When all file descriptors associated with an open file description have been closed, the open file
 6900 description shall be freed.

6901 If the link count of the file is 0, when all file descriptors associated with the file are closed, the
 6902 space occupied by the file shall be freed and the file shall no longer be accessible.

6903 **XSR** If a STREAMS-based *fildes* is closed and the calling process was previously registered to receive
 6904 a SIGPOLL signal for events associated with that STREAM, the calling process shall be
 6905 unregistered for events associated with the STREAM. The last *close()* for a STREAM shall cause
 6906 the STREAM associated with *fildes* to be dismantled. If O_NONBLOCK is not set and there have
 6907 been no signals posted for the STREAM, and if there is data on the module's write queue, *close()*
 6908 shall wait for an unspecified time (for each module and driver) for any output to drain before
 6909 dismantling the STREAM. The time delay can be changed via an I_SETCLTIME *ioctl()* request. If
 6910 the O_NONBLOCK flag is set, or if there are any pending signals, *close()* shall not wait for
 6911 output to drain, and shall dismantle the STREAM immediately.

6912 If the implementation supports STREAMS-based pipes, and *fildes* is associated with one end of a
 6913 pipe, the last *close()* shall cause a hangup to occur on the other end of the pipe. In addition, if the
 6914 other end of the pipe has been named by *fattach()*, then the last *close()* shall force the named end
 6915 to be detached by *fdetach()*. If the named end has no open file descriptors associated with it and
 6916 gets detached, the STREAM associated with that end shall also be dismantled.

6917 **XSI** If *fildes* refers to the master side of a pseudo-terminal, and this is the last close, a SIGHUP signal
 6918 shall be sent to the controlling process, if any, for which the slave side of the pseudo-terminal is
 6919 the controlling terminal. It is unspecified whether closing the master side of the pseudo-terminal
 6920 flushes all queued input and output.

6921 **XSR** If *fildes* refers to the slave side of a STREAMS-based pseudo-terminal, a zero-length message
 6922 may be sent to the master.

6923 **AIO** When there is an outstanding cancelable asynchronous I/O operation against *fildes* when *close()*
 6924 is called, that I/O operation may be canceled. An I/O operation that is not canceled completes
 6925 as if the *close()* operation had not yet occurred. All operations that are not canceled shall
 6926 complete as if the *close()* blocked until the operations completed. The *close()* operation itself
 6927 need not block awaiting such I/O completion. Whether any I/O operation is canceled, and
 6928 which I/O operation may be canceled upon *close()*, is implementation-defined.

6929 MF|SHM If a shared memory object or a memory mapped file remains referenced at the last close (that is,
 6930 a process has it mapped), then the entire contents of the memory object shall persist until the
 6931 memory object becomes unreferenced. If this is the last close of a shared memory object or a
 6932 memory mapped file and the close results in the memory object becoming unreferenced, and the
 6933 memory object has been unlinked, then the memory object shall be removed.

6934 If *fdes* refers to a socket, *close()* shall cause the socket to be destroyed. If the socket is in
 6935 connection-mode, and the `SO_LINGER` option is set for the socket with non-zero linger time,
 6936 and the socket has untransmitted data, then *close()* shall block for up to the current linger
 6937 interval until all data is transmitted.

6938 RETURN VALUE

6939 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
 6940 indicate the error.

6941 ERRORS

6942 The *close()* function shall fail if:

6943 [EBADF] The *fdes* argument is not a valid file descriptor.

6944 [EINTR] The *close()* function was interrupted by a signal.

6945 The *close()* function may fail if:

6946 [EIO] An I/O error occurred while reading from or writing to the file system.

6947 EXAMPLES

6948 Reassigning a File Descriptor

6949 The following example closes the file descriptor associated with standard output for the current
 6950 process, re-assigns standard output to a new file descriptor, and closes the original file
 6951 descriptor to clean up. This example assumes that the file descriptor 0 (which is the descriptor
 6952 for standard input) is not closed.

```
6953 #include <unistd.h>
6954 ...
6955 int pfd;
6956 ...
6957 close(1);
6958 dup(pfd);
6959 close(pfd);
6960 ...
```

6961 Incidentally, this is exactly what could be achieved using:

```
6962 dup2(pfd, 1);
6963 close(pfd);
```

6964 Closing a File Descriptor

6965 In the following example, *close()* is used to close a file descriptor after an unsuccessful attempt is
 6966 made to associate that file descriptor with a stream.

```
6967 #include <stdio.h>
6968 #include <unistd.h>
6969 #include <stdlib.h>
```

```

6970     #define LOCKFILE "/etc/ptmp"
6971     ...
6972     int pfd;
6973     FILE *fpfd;
6974     ...
6975     if ((fpfd = fdopen (pfd, "w")) == NULL) {
6976         close(pfd);
6977         unlink(LOCKFILE);
6978         exit(1);
6979     }
6980     ...

```

6981 APPLICATION USAGE

6982 An application that had used the *stdio* routine *fopen()* to open a file should use the
 6983 corresponding *fclose()* routine rather than *close()*. Once a file is closed, the file descriptor no
 6984 longer exists, since the integer corresponding to it no longer refers to a file.

6985 RATIONALE

6986 The use of interruptible device close routines should be discouraged to avoid problems with the
 6987 implicit closes of file descriptors by *exec* and *exit()*. This volume of IEEE Std 1003.1-2001 only
 6988 intends to permit such behavior by specifying the [EINTR] error condition.

6989 FUTURE DIRECTIONS

6990 None.

6991 SEE ALSO

6992 Section 2.6 (on page 38), *fattach()*, *fclose()*, *fdetach()*, *fopen()*, *ioctl()*, *open()*, the Base Definitions
 6993 volume of IEEE Std 1003.1-2001, <*unistd.h*>

6994 CHANGE HISTORY

6995 First released in Issue 1. Derived from Issue 1 of the SVID.

6996 Issue 5

6997 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

6998 Issue 6

6999 The DESCRIPTION related to a STREAMS-based file or pseudo-terminal is marked as part of the
 7000 XSI STREAMS Option Group.

7001 The following new requirements on POSIX implementations derive from alignment with the
 7002 Single UNIX Specification:

- 7003 • The [EIO] error condition is added as an optional error.
- 7004 • The DESCRIPTION is updated to describe the state of the *fildev* file descriptor as unspecified
 7005 if an I/O error occurs and an [EIO] error condition is returned.

7006 Text referring to sockets is added to the DESCRIPTION.

7007 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
 7008 shared memory objects and memory mapped files (and not typed memory objects) are the types
 7009 of memory objects to which the paragraph on last closes applies.

7010 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/12 is applied, correcting the XSH shaded
 7011 text relating to the master side of a pseudo-terminal. The reason for the change is that the
 7012 behavior of pseudo-terminals and regular terminals should be as much alike as possible in this
 7013 case; the change achieves that and matches historical behavior.

7014 **NAME**

7015 closedir — close a directory stream

7016 **SYNOPSIS**

7017 #include <dirent.h>

7018 int closedir(DIR *dirp);

7019 **DESCRIPTION**

7020 The *closedir()* function shall close the directory stream referred to by the argument *dirp*. Upon
7021 return, the value of *dirp* may no longer point to an accessible object of the type **DIR**. If a file
7022 descriptor is used to implement type **DIR**, that file descriptor shall be closed.

7023 **RETURN VALUE**

7024 Upon successful completion, *closedir()* shall return 0; otherwise, -1 shall be returned and *errno*
7025 set to indicate the error.

7026 **ERRORS**7027 The *closedir()* function may fail if:7028 [EBADF] The *dirp* argument does not refer to an open directory stream.7029 [EINTR] The *closedir()* function was interrupted by a signal.7030 **EXAMPLES**7031 **Closing a Directory Stream**7032 The following program fragment demonstrates how the *closedir()* function is used.

```
7033        ...  
7034        DIR *dir;  
7035        struct dirent *dp;  
7036        ...  
7037        if ((dir = opendir(".")) == NULL) {  
7038        ...  
7039        }  
7040        while ((dp = readdir(dir)) != NULL) {  
7041        ...  
7042        }  
7043        closedir(dir);  
7044        ...
```

7045 **APPLICATION USAGE**

7046 None.

7047 **RATIONALE**

7048 None.

7049 **FUTURE DIRECTIONS**

7050 None.

7051 **SEE ALSO**7052 *opendir()*, the Base Definitions volume of IEEE Std 1003.1-2001, <dirent.h>

7053 **CHANGE HISTORY**

7054 First released in Issue 2.

7055 **Issue 6**7056 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.7057 The following new requirements on POSIX implementations derive from alignment with the
7058 Single UNIX Specification:

- 7059 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
7060 required for conforming implementations of previous POSIX specifications, it was not
7061 required for UNIX applications.
- 7062 • The [EINTR] error condition is added as an optional error condition.

7063 **NAME**

7064 closelog, openlog, setlogmask, syslog — control system log

7065 **SYNOPSIS**

```
7066 xSI #include <syslog.h>
7067
7068 void closelog(void);
7069 void openlog(const char *ident, int logopt, int facility);
7070 int setlogmask(int maskpri);
7071 void syslog(int priority, const char *message, ... /* arguments */);
```

7072 **DESCRIPTION**

7073 The *syslog()* function shall send a message to an implementation-defined logging facility, which
 7074 may log it in an implementation-defined system log, write it to the system console, forward it to
 7075 a list of users, or forward it to the logging facility on another host over the network. The logged
 7076 message shall include a message header and a message body. The message header contains at
 7077 least a timestamp and a tag string.

7078 The message body is generated from the *message* and following arguments in the same manner
 7079 as if these were arguments to *printf()*, except that the additional conversion specification *%m*
 7080 shall be recognized; it shall convert no arguments, shall cause the output of the error message
 7081 string associated with the value of *errno* on entry to *syslog()*, and may be mixed with argument
 7082 specifications of the "*%n\$*" form. If a complete conversion specification with the *m* conversion
 7083 specifier character is not just *%m*, the behavior is undefined. A trailing *<newline>* may be added
 7084 if needed.

7085 Values of the *priority* argument are formed by OR'ing together a severity-level value and an
 7086 optional facility value. If no facility value is specified, the current default facility value is used.

7087 Possible values of severity level include:

7088	LOG_EMERG	A panic condition.
7089	LOG_ALERT	A condition that should be corrected immediately, such as a corrupted system 7090 database.
7091	LOG_CRIT	Critical conditions, such as hard device errors.
7092	LOG_ERR	Errors.
7093	LOG_WARNING	Warning messages.
7094		
7095	LOG_NOTICE	Conditions that are not error conditions, but that may require special 7096 handling.
7097	LOG_INFO	Informational messages.
7098	LOG_DEBUG	Messages that contain information normally of use only when debugging a 7099 program.

7100 The facility indicates the application or system component generating the message. Possible
 7101 facility values include:

7102	LOG_USER	Messages generated by arbitrary processes. This is the default facility 7103 identifier if none is specified.
7104	LOG_LOCAL0	Reserved for local use.

- 7105 LOG_LOCAL1 Reserved for local use.
- 7106 LOG_LOCAL2 Reserved for local use.
- 7107 LOG_LOCAL3 Reserved for local use.
- 7108 LOG_LOCAL4 Reserved for local use.
- 7109 LOG_LOCAL5 Reserved for local use.
- 7110 LOG_LOCAL6 Reserved for local use.
- 7111 LOG_LOCAL7 Reserved for local use.
- 7112 The *openlog()* function shall set process attributes that affect subsequent calls to *syslog()*. The
- 7113 *ident* argument is a string that is prepended to every message. The *logopt* argument indicates
- 7114 logging options. Values for *logopt* are constructed by a bitwise-inclusive OR of zero or more of
- 7115 the following:
- 7116 LOG_PID Log the process ID with each message. This is useful for identifying specific
- 7117 processes.
- 7118 LOG_CONS Write messages to the system console if they cannot be sent to the logging
- 7119 facility. The *syslog()* function ensures that the process does not acquire the
- 7120 console as a controlling terminal in the process of writing the message.
- 7121 LOG_NDELAY Open the connection to the logging facility immediately. Normally the open is
- 7122 delayed until the first message is logged. This is useful for programs that need
- 7123 to manage the order in which file descriptors are allocated.
- 7124 LOG_ODELAY Delay open until *syslog()* is called.
- 7125 LOG_NOWAIT Do not wait for child processes that may have been created during the course
- 7126 of logging the message. This option should be used by processes that enable
- 7127 notification of child termination using SIGCHLD, since *syslog()* may
- 7128 otherwise block waiting for a child whose exit status has already been
- 7129 collected.
- 7130 The *facility* argument encodes a default facility to be assigned to all messages that do not have
- 7131 an explicit facility already encoded. The initial default facility is LOG_USER.
- 7132 The *openlog()* and *syslog()* functions may allocate a file descriptor. It is not necessary to call
- 7133 *openlog()* prior to calling *syslog()*.
- 7134 The *closelog()* function shall close any open file descriptors allocated by previous calls to
- 7135 *openlog()* or *syslog()*.
- 7136 The *setlogmask()* function shall set the log priority mask for the current process to *maskpri* and
- 7137 return the previous mask. If the *maskpri* argument is 0, the current log mask is not modified.
- 7138 Calls by the current process to *syslog()* with a priority not set in *maskpri* shall be rejected. The
- 7139 default log mask allows all priorities to be logged. A call to *openlog()* is not required prior to
- 7140 calling *setlogmask()*.
- 7141 Symbolic constants for use as values of the *logopt*, *facility*, *priority*, and *maskpri* arguments are
- 7142 defined in the `<syslog.h>` header.
- 7143 **RETURN VALUE**
- 7144 The *setlogmask()* function shall return the previous log priority mask. The *closelog()*, *openlog()*,
- 7145 and *syslog()* functions shall not return a value.

7146 **ERRORS**

7147 None.

7148 **EXAMPLES**7149 **Using openlog()**

7150 The following example causes subsequent calls to *syslog()* to log the process ID with each message, and to write messages to the system console if they cannot be sent to the logging facility.

```
7153       #include <syslog.h>
7154       char *ident = "Process demo";
7155       int logopt = LOG_PID | LOG_CONS;
7156       int facility = LOG_USER;
7157       ...
7158       openlog(ident, logopt, facility);
```

7159 **Using setlogmask()**

7160 The following example causes subsequent calls to *syslog()* to accept error messages, and to reject all other messages.

```
7162       #include <syslog.h>
7163       int result;
7164       int mask = LOG_MASK (LOG_ERR);
7165       ...
7166       result = setlogmask(mask);
```

7167 **Using syslog**

7168 The following example sends the message "This is a message" to the default logging facility, marking the message as an error message generated by random processes.

```
7170       #include <syslog.h>
7171       char *message = "This is a message";
7172       int priority = LOG_ERR | LOG_USER;
7173       ...
7174       syslog(priority, message);
```

7175 **APPLICATION USAGE**

7176 None.

7177 **RATIONALE**

7178 None.

7179 **FUTURE DIRECTIONS**

7180 None.

7181 **SEE ALSO**7182 *printf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**syslog.h**>

7183 **CHANGE HISTORY**

7184 First released in Issue 4, Version 2.

7185 **Issue 5**

7186 Moved from X/OPEN UNIX extension to BASE.

7187 **Issue 6**

7188 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/13 is applied, correcting the EXAMPLES |
7189 section. |

7190 NAME

7191 confstr — get configurable variables

7192 SYNOPSIS

7193 #include <unistd.h>

7194 size_t confstr(int name, char *buf, size_t len);

7195 DESCRIPTION

7196 The *confstr()* function shall return configuration-defined string values. Its use and purpose are similar to *sysconf()*, but it is used where string values rather than numeric values are returned.7198 The *name* argument represents the system variable to be queried. The implementation shall support the following name values, defined in <unistd.h>. It may support others:

7200 _CS_PATH
 7201 _CS_POSIX_V6_ILP32_OFF32_CFLAGS
 7202 _CS_POSIX_V6_ILP32_OFF32_LDFLAGS
 7203 _CS_POSIX_V6_ILP32_OFF32_LIBS
 7204 _CS_POSIX_V6_ILP32_OFFBIG_CFLAGS
 7205 _CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS
 7206 _CS_POSIX_V6_ILP32_OFFBIG_LIBS
 7207 _CS_POSIX_V6_LP64_OFF64_CFLAGS
 7208 _CS_POSIX_V6_LP64_OFF64_LDFLAGS
 7209 _CS_POSIX_V6_LP64_OFF64_LIBS
 7210 _CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS
 7211 _CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS
 7212 _CS_POSIX_V6_LPBIG_OFFBIG_LIBS
 7213 _CS_POSIX_V6_WIDTH_RESTRICTED_ENVS
 7214 XSI CS_XBS5_ILP32_OFF32_CFLAGS (LEGACY)
 7215 CS_XBS5_ILP32_OFF32_LDFLAGS (LEGACY)
 7216 CS_XBS5_ILP32_OFF32_LIBS (LEGACY)
 7217 CS_XBS5_ILP32_OFF32_LINTFLAGS (LEGACY)
 7218 CS_XBS5_ILP32_OFFBIG_CFLAGS (LEGACY)
 7219 CS_XBS5_ILP32_OFFBIG_LDFLAGS (LEGACY)
 7220 CS_XBS5_ILP32_OFFBIG_LIBS (LEGACY)
 7221 CS_XBS5_ILP32_OFFBIG_LINTFLAGS (LEGACY)
 7222 CS_XBS5_LP64_OFF64_CFLAGS (LEGACY)
 7223 CS_XBS5_LP64_OFF64_LDFLAGS (LEGACY)
 7224 CS_XBS5_LP64_OFF64_LIBS (LEGACY)
 7225 CS_XBS5_LP64_OFF64_LINTFLAGS (LEGACY)
 7226 CS_XBS5_LPBIG_OFFBIG_CFLAGS (LEGACY)
 7227 CS_XBS5_LPBIG_OFFBIG_LDFLAGS (LEGACY)
 7228 CS_XBS5_LPBIG_OFFBIG_LIBS (LEGACY)
 7229 CS_XBS5_LPBIG_OFFBIG_LINTFLAGS (LEGACY)

7230

7231 If *len* is not 0, and if *name* has a configuration-defined value, *confstr()* shall copy that value into the *len*-byte buffer pointed to by *buf*. If the string to be returned is longer than *len* bytes, including the terminating null, then *confstr()* shall truncate the string to *len*–1 bytes and null-terminate the result. The application can detect that the string was truncated by comparing the value returned by *confstr()* with *len*.

7236 If *len* is 0 and *buf* is a null pointer, then *confstr()* shall still return the integer value as defined below, but shall not return a string. If *len* is 0 but *buf* is not a null pointer, the result is unspecified.

7238

7239 If the implementation supports the POSIX shell option, the string stored in *buf* after a call to:

```
7240 confstr(_CS_PATH, buf, sizeof(buf))
```

7241 can be used as a value of the *PATH* environment variable that accesses all of the standard
7242 utilities of IEEE Std 1003.1-2001, if the return value is less than or equal to *sizeof(buf)*.

7243 RETURN VALUE

7244 If *name* has a configuration-defined value, *confstr()* shall return the size of buffer that would be
7245 needed to hold the entire configuration-defined value including the terminating null. If this
7246 return value is greater than *len*, the string returned in *buf* is truncated.

7247 If *name* is invalid, *confstr()* shall return 0 and set *errno* to indicate the error.

7248 If *name* does not have a configuration-defined value, *confstr()* shall return 0 and leave *errno*
7249 unchanged.

7250 ERRORS

7251 The *confstr()* function shall fail if:

7252 [EINVAL] The value of the *name* argument is invalid.

7253 EXAMPLES

7254 None.

7255 APPLICATION USAGE

7256 An application can distinguish between an invalid *name* parameter value and one that
7257 corresponds to a configurable variable that has no configuration-defined value by checking if
7258 *errno* is modified. This mirrors the behavior of *sysconf()*.

7259 The original need for this function was to provide a way of finding the configuration-defined
7260 default value for the environment variable *PATH*. Since *PATH* can be modified by the user to
7261 include directories that could contain utilities replacing the standard utilities in the Shell and
7262 Utilities volume of IEEE Std 1003.1-2001, applications need a way to determine the system-
7263 supplied *PATH* environment variable value that contains the correct search path for the standard
7264 utilities.

7265 An application could use:

```
7266 confstr(name, (char *)NULL, (size_t)0)
```

7267 to find out how big a buffer is needed for the string value; use *malloc()* to allocate a buffer to
7268 hold the string; and call *confstr()* again to get the string. Alternately, it could allocate a fixed,
7269 static buffer that is big enough to hold most answers (perhaps 512 or 1024 bytes), but then use
7270 *malloc()* to allocate a larger buffer if it finds that this is too small.

7271 RATIONALE

7272 Application developers can normally determine any configuration variable by means of reading
7273 from the stream opened by a call to:

```
7274 popen("command -p getconf variable", "r");
```

7275 The *confstr()* function with a *name* argument of *_CS_PATH* returns a string that can be used as a
7276 *PATH* environment variable setting that will reference the standard shell and utilities as
7277 described in the Shell and Utilities volume of IEEE Std 1003.1-2001.

7278 The *confstr()* function copies the returned string into a buffer supplied by the application instead
7279 of returning a pointer to a string. This allows a cleaner function in some implementations (such
7280 as those with lightweight threads) and resolves questions about when the application must copy
7281 the string returned.

7282 **FUTURE DIRECTIONS**

7283 None.

7284 **SEE ALSO**7285 *pathconf()*, *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>, the Shell
7286 and Utilities volume of IEEE Std 1003.1-2001, *c99*7287 **CHANGE HISTORY**

7288 First released in Issue 4. Derived from the ISO POSIX-2 standard.

7289 **Issue 5**7290 A table indicating the permissible values of *name* is added to the DESCRIPTION. All those
7291 marked EX are new in this issue.7292 **Issue 6**7293 The Open Group Corrigendum U033/7 is applied. The return value for the case returning the
7294 size of the buffer now explicitly states that this includes the terminating null.7295 The following new requirements on POSIX implementations derive from alignment with the
7296 Single UNIX Specification:

- 7297
- The DESCRIPTION is updated with new arguments which can be used to determine
7298 configuration strings for C compiler flags, linker/loader flags, and libraries for each different
7299 supported programming environment. This is a change to support data size neutrality.

7300 The following changes were made to align with the IEEE P1003.1a draft standard:

- 7301
- The DESCRIPTION is updated to include text describing how `_CS_PATH` can be used to
7302 obtain a *PATH* to access the standard utilities.

7303 The macros associated with the *c89* programming models are marked LEGACY and new
7304 equivalent macros associated with *c99* are introduced.

7305 **NAME**

7306 conj, conjf, conjl — complex conjugate functions

7307 **SYNOPSIS**

7308 #include <complex.h>

7309 double complex conj(double complex z);

7310 float complex conjf(float complex z);

7311 long double complex conjl(long double complex z);

7312 **DESCRIPTION**

7313 cx The functionality described on this reference page is aligned with the ISO C standard. Any
7314 conflict between the requirements described here and the ISO C standard is unintentional. This
7315 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7316 These functions shall compute the complex conjugate of *z*, by reversing the sign of its imaginary
7317 part.

7318 **RETURN VALUE**

7319 These functions return the complex conjugate value.

7320 **ERRORS**

7321 No errors are defined.

7322 **EXAMPLES**

7323 None.

7324 **APPLICATION USAGE**

7325 None.

7326 **RATIONALE**

7327 None.

7328 **FUTURE DIRECTIONS**

7329 None.

7330 **SEE ALSO**

7331 *carg()*, *cimag()*, *cproj()*, *creal()*, the Base Definitions volume of IEEE Std 1003.1-2001,
7332 <complex.h>

7333 **CHANGE HISTORY**

7334 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7335 **NAME**

7336 connect — connect a socket

7337 **SYNOPSIS**

7338 #include <sys/socket.h>

7339 int connect(int *socket*, const struct sockaddr **address*,
7340 socklen_t *address_len*);7341 **DESCRIPTION**7342 The *connect()* function shall attempt to make a connection on a socket. The function takes the
7343 following arguments:

7344	<i>socket</i>	Specifies the file descriptor associated with the socket.
7345	<i>address</i>	Points to a sockaddr structure containing the peer address. The length and 7346 format of the address depend on the address family of the socket.
7347	<i>address_len</i>	Specifies the length of the sockaddr structure pointed to by the <i>address</i> 7348 argument.

7349 If the socket has not already been bound to a local address, *connect()* shall bind it to an address
7350 which, unless the socket's address family is AF_UNIX, is an unused local address.7351 If the initiating socket is not connection-mode, then *connect()* shall set the socket's peer address,
7352 and no connection is made. For SOCK_DGRAM sockets, the peer address identifies where all
7353 datagrams are sent on subsequent *send()* functions, and limits the remote sender for subsequent
7354 *recv()* functions. If *address* is a null address for the protocol, the socket's peer address shall be
7355 reset.7356 If the initiating socket is connection-mode, then *connect()* shall attempt to establish a connection
7357 to the address specified by the *address* argument. If the connection cannot be established
7358 immediately and O_NONBLOCK is not set for the file descriptor for the socket, *connect()* shall
7359 block for up to an unspecified timeout interval until the connection is established. If the timeout
7360 interval expires before the connection is established, *connect()* shall fail and the connection
7361 attempt shall be aborted. If *connect()* is interrupted by a signal that is caught while blocked
7362 waiting to establish a connection, *connect()* shall fail and set *errno* to [EINTR], but the connection
7363 request shall not be aborted, and the connection shall be established asynchronously.7364 If the connection cannot be established immediately and O_NONBLOCK is set for the file
7365 descriptor for the socket, *connect()* shall fail and set *errno* to [EINPROGRESS], but the connection
7366 request shall not be aborted, and the connection shall be established asynchronously.
7367 Subsequent calls to *connect()* for the same socket, before the connection is established, shall fail
7368 and set *errno* to [EALREADY].7369 When the connection has been established asynchronously, *select()* and *poll()* shall indicate that
7370 the file descriptor for the socket is ready for writing.7371 The socket in use may require the process to have appropriate privileges to use the *connect()*
7372 function.7373 **RETURN VALUE**7374 Upon successful completion, *connect()* shall return 0; otherwise, -1 shall be returned and *errno*
7375 set to indicate the error.7376 **ERRORS**7377 The *connect()* function shall fail if:

7378 [EADDRNOTAVAIL]

7379 The specified address is not available from the local machine.

7380	[EAFNOSUPPORT]	
7381		The specified address is not a valid address for the address family of the
7382		specified socket.
7383	[EALREADY]	A connection request is already in progress for the specified socket.
7384	[EBADF]	The <i>socket</i> argument is not a valid file descriptor.
7385	[ECONNREFUSED]	
7386		The target address was not listening for connections or refused the connection
7387		request.
7388	[EINPROGRESS]	O_NONBLOCK is set for the file descriptor for the socket and the connection
7389		cannot be immediately established; the connection shall be established
7390		asynchronously.
7391	[EINTR]	The attempt to establish a connection was interrupted by delivery of a signal
7392		that was caught; the connection shall be established asynchronously.
7393	[EISCONN]	The specified socket is connection-mode and is already connected.
7394	[ENETUNREACH]	
7395		No route to the network is present.
7396	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
7397	[EPROTOTYPE]	The specified address has a different type than the socket bound to the
7398		specified peer address.
7399	[ETIMEDOUT]	The attempt to connect timed out before a connection was made.
7400		If the address family of the socket is AF_UNIX, then <i>connect()</i> shall fail if:
7401	[EIO]	An I/O error occurred while reading from or writing to the file system.
7402	[ELOOP]	A loop exists in symbolic links encountered during resolution of the pathname
7403		in <i>address</i> .
7404	[ENAMETOOLONG]	
7405		A component of a pathname exceeded {NAME_MAX} characters, or an entire
7406		pathname exceeded {PATH_MAX} characters.
7407	[ENOENT]	A component of the pathname does not name an existing file or the pathname
7408		is an empty string.
7409	[ENOTDIR]	A component of the path prefix of the pathname in <i>address</i> is not a directory.
7410		The <i>connect()</i> function may fail if:
7411	[EACCES]	Search permission is denied for a component of the path prefix; or write
7412		access to the named socket is denied.
7413	[EADDRINUSE]	Attempt to establish a connection that uses addresses that are already in use.
7414	[ECONNRESET]	Remote host reset the connection request.
7415	[EHOSTUNREACH]	
7416		The destination host cannot be reached (probably because the host is down or
7417		a remote router cannot reach it).
7418	[EINVAL]	The <i>address_len</i> argument is not a valid length for the address family; or
7419		invalid address family in the sockaddr structure.

- 7420 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
7421 resolution of the pathname in *address*.
- 7422 [ENAMETOOLONG]
7423 Pathname resolution of a symbolic link produced an intermediate result
7424 whose length exceeds {PATH_MAX}.
- 7425 [ENETDOWN] The local network interface used to reach the destination is down.
- 7426 [ENOBUFS] No buffer space is available.
- 7427 [EOPNOTSUPP] The socket is listening and cannot be connected.
- 7428 **EXAMPLES**
7429 None.
- 7430 **APPLICATION USAGE**
7431 If *connect()* fails, the state of the socket is unspecified. Conforming applications should close the
7432 file descriptor and create a new socket before attempting to reconnect.
- 7433 **RATIONALE**
7434 None.
- 7435 **FUTURE DIRECTIONS**
7436 None.
- 7437 **SEE ALSO**
7438 *accept()*, *bind()*, *close()*, *getsockname()*, *poll()*, *select()*, *send()*, *shutdown()*, *socket()*, the Base
7439 Definitions volume of IEEE Std 1003.1-2001, <**sys/socket.h**>
- 7440 **CHANGE HISTORY**
7441 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.
7442 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
7443 [ELOOP] error condition is added.

7444 **NAME**

7445 copysign, copysignf, copysignl — number manipulation function

7446 **SYNOPSIS**

7447 #include <math.h>

7448 double copysign(double x, double y);

7449 float copysignf(float x, float y);

7450 long double copysignl(long double x, long double y);

7451 **DESCRIPTION**

7452 cx The functionality described on this reference page is aligned with the ISO C standard. Any
7453 conflict between the requirements described here and the ISO C standard is unintentional. This
7454 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7455 These functions shall produce a value with the magnitude of *x* and the sign of *y*. On
7456 implementations that represent a signed zero but do not treat negative zero consistently in
7457 arithmetic operations, these functions regard the sign of zero as positive.

7458 **RETURN VALUE**

7459 Upon successful completion, these functions shall return a value with the magnitude of *x* and
7460 the sign of *y*.

7461 **ERRORS**

7462 No errors are defined.

7463 **EXAMPLES**

7464 None.

7465 **APPLICATION USAGE**

7466 None.

7467 **RATIONALE**

7468 None.

7469 **FUTURE DIRECTIONS**

7470 None.

7471 **SEE ALSO**7472 *signbit()*, the Base Definitions volume of IEEE Std 1003.1-2001, <math.h>7473 **CHANGE HISTORY**

7474 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7475 **NAME**

7476 cos, cosf, cosl — cosine function

7477 **SYNOPSIS**

```
7478 #include <math.h>
7479 double cos(double x);
7480 float cosf(float x);
7481 long double cosl(long double x);
```

7482 **DESCRIPTION**

7483 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 7484 conflict between the requirements described here and the ISO C standard is unintentional. This
 7485 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7486 These functions shall compute the cosine of their argument *x*, measured in radians.

7487 An application wishing to check for error situations should set *errno* to zero and call
 7488 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 7489 *fetetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 7490 zero, an error has occurred.

7491 **RETURN VALUE**

7492 Upon successful completion, these functions shall return the cosine of *x*.

7493 **MX** If *x* is NaN, a NaN shall be returned.

7494 If *x* is ± 0 , the value 1.0 shall be returned.

7495 If *x* is $\pm \text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 7496 defined value shall be returned.

7497 **ERRORS**

7498 These functions shall fail if:

7499 **MX** **Domain Error** The *x* argument is $\pm \text{Inf}$.

7500 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 7501 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 7502 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 7503 shall be raised.

7504 **EXAMPLES**7505 **Taking the Cosine of a 45-Degree Angle**

```
7506 #include <math.h>
7507 ...
7508 double radians = 45 * M_PI / 180;
7509 double result;
7510 ...
7511 result = cos(radians);
```

7512 **APPLICATION USAGE**

7513 These functions may lose accuracy when their argument is near an odd multiple of $\pi/2$ or is far
 7514 from 0.

7515 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 7516 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

7517 **RATIONALE**

7518 None.

7519 **FUTURE DIRECTIONS**

7520 None.

7521 **SEE ALSO**

7522 *acos()*, *feclearexcept()*, *fetetestexcept()*, *isnan()*, *sin()*, *tan()*, the Base Definitions volume of
7523 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
7524 **<math.h>**

7525 **CHANGE HISTORY**

7526 First released in Issue 1. Derived from Issue 1 of the SVID.

7527 **Issue 5**

7528 The DESCRIPTION is updated to indicate how an application should check for an error. This
7529 text was previously published in the APPLICATION USAGE section.

7530 **Issue 6**7531 The *cosf()* and *cosl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

7532 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
7533 revised to align with the ISO/IEC 9899:1999 standard.

7534 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
7535 marked.

7536 **NAME**

7537 cosh, coshf, coshl — hyperbolic cosine functions

7538 **SYNOPSIS**

7539 #include <math.h>

7540 double cosh(double x);

7541 float coshf(float x);

7542 long double coshl(long double x);

7543 **DESCRIPTION**

7544 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 7545 conflict between the requirements described here and the ISO C standard is unintentional. This
 7546 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7547 These functions shall compute the hyperbolic cosine of their argument x .

7548 An application wishing to check for error situations should set *errno* to zero and call
 7549 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 7550 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 7551 zero, an error has occurred.

7552 **RETURN VALUE**7553 Upon successful completion, these functions shall return the hyperbolic cosine of x .

7554 If the correct value would cause overflow, a range error shall occur and *cosh*(), *coshf*(), and
 7555 *coshl*() shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
 7556 respectively.

7557 **MX** If x is NaN, a NaN shall be returned.7558 If x is ± 0 , the value 1.0 shall be returned.7559 If x is $\pm\text{Inf}$, $+\text{Inf}$ shall be returned.7560 **ERRORS**

7561 These functions shall fail if:

7562 **Range Error** The result would cause an overflow.

7563 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 7564 then *errno* shall be set to [ERANGE]. If the integer expression
 7565 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 7566 floating-point exception shall be raised.

7567 **EXAMPLES**

7568 None.

7569 **APPLICATION USAGE**

7570 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 7571 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

7572 For IEEE Std 754-1985 **double**, $710.5 < |x|$ implies that *cosh*(x) has overflowed.7573 **RATIONALE**

7574 None.

7575 **FUTURE DIRECTIONS**

7576 None.

7577 **SEE ALSO**

7578 *acosh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *sinh()*, *tanh()*, the Base Definitions volume of
7579 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
7580 <math.h>

7581 **CHANGE HISTORY**

7582 First released in Issue 1. Derived from Issue 1 of the SVID.

7583 **Issue 5**

7584 The DESCRIPTION is updated to indicate how an application should check for an error. This
7585 text was previously published in the APPLICATION USAGE section.

7586 **Issue 6**

7587 The *coshf()* and *coshl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

7588 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
7589 revised to align with the ISO/IEC 9899:1999 standard.

7590 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
7591 marked.

7592 **NAME**7593 **cosl** — cosine function7594 **SYNOPSIS**

7595 #include <math.h>

7596 long double cosl(long double x);

7597 **DESCRIPTION**7598 Refer to *cos()*.

7599 **NAME**

7600 cpow, cpowf, cpowl — complex power functions

7601 **SYNOPSIS**

7602 #include <complex.h>

7603 double complex cpow(double complex x, double complex y);

7604 float complex cpowf(float complex x, float complex y);

7605 long double complex cpowl(long double complex x,

7606 long double complex y);

7607 **DESCRIPTION**

7608 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
7609 conflict between the requirements described here and the ISO C standard is unintentional. This
7610 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7611 These functions shall compute the complex power function x^y , with a branch cut for the first
7612 parameter along the negative real axis.

7613 **RETURN VALUE**

7614 These functions shall return the complex power function value.

7615 **ERRORS**

7616 No errors are defined.

7617 **EXAMPLES**

7618 None.

7619 **APPLICATION USAGE**

7620 None.

7621 **RATIONALE**

7622 None.

7623 **FUTURE DIRECTIONS**

7624 None.

7625 **SEE ALSO**7626 *cabs()*, *csqrt()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>7627 **CHANGE HISTORY**

7628 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7629 **NAME**

7630 cproj, cprojf, cprojl — complex projection functions

7631 **SYNOPSIS**

7632 #include <complex.h>

7633 double complex cproj(double complex z);

7634 float complex cprojf(float complex z);

7635 long double complex cprojl(long double complex z);

7636 **DESCRIPTION**

7637 cx The functionality described on this reference page is aligned with the ISO C standard. Any
7638 conflict between the requirements described here and the ISO C standard is unintentional. This
7639 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7640 These functions shall compute a projection of z onto the Riemann sphere: z projects to z , except
7641 that all complex infinities (even those with one infinite part and one NaN part) project to
7642 positive infinity on the real axis. If z has an infinite part, then $cproj(z)$ shall be equivalent to:

7643 $\text{INFINITY} + \text{I} * \text{copysign}(0.0, \text{cimag}(z))$ 7644 **RETURN VALUE**

7645 These functions shall return the value of the projection onto the Riemann sphere.

7646 **ERRORS**

7647 No errors are defined.

7648 **EXAMPLES**

7649 None.

7650 **APPLICATION USAGE**

7651 None.

7652 **RATIONALE**

7653 Two topologies are commonly used in complex mathematics: the complex plane with its
7654 continuum of infinities, and the Riemann sphere with its single infinity. The complex plane is
7655 better suited for transcendental functions, the Riemann sphere for algebraic functions. The
7656 complex types with their multiplicity of infinities provide a useful (though imperfect) model for
7657 the complex plane. The $cproj()$ function helps model the Riemann sphere by mapping all
7658 infinities to one, and should be used just before any operation, especially comparisons, that
7659 might give spurious results for any of the other infinities. Note that a complex value with one
7660 infinite part and one NaN part is regarded as an infinity, not a NaN, because if one part is
7661 infinite, the complex value is infinite independent of the value of the other part. For the same
7662 reason, $cabs()$ returns an infinity if its argument has an infinite part and a NaN part.

7663 **FUTURE DIRECTIONS**

7664 None.

7665 **SEE ALSO**7666 $carg()$, $cimag()$, $conj()$, $creal()$, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>7667 **CHANGE HISTORY**

7668 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7669 **NAME**7670 `creal`, `crealf`, `creall` — complex real functions7671 **SYNOPSIS**7672 `#include <complex.h>`7673 `double creal(double complex z);`7674 `float crealf(float complex z);`7675 `long double creall(long double complex z);`7676 **DESCRIPTION**

7677 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
7678 conflict between the requirements described here and the ISO C standard is unintentional. This
7679 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7680 These functions shall compute the real part of *z*.7681 **RETURN VALUE**

7682 These functions shall return the real part value.

7683 **ERRORS**

7684 No errors are defined.

7685 **EXAMPLES**

7686 None.

7687 **APPLICATION USAGE**7688 For a variable *z* of type **complex**:7689 `z == creal(z) + cimag(z)*I`7690 **RATIONALE**

7691 None.

7692 **FUTURE DIRECTIONS**

7693 None.

7694 **SEE ALSO**

7695 `carg()`, `cimag()`, `conj()`, `cproj()`, the Base Definitions volume of IEEE Std 1003.1-2001,
7696 `<complex.h>`

7697 **CHANGE HISTORY**

7698 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7699 **NAME**

7700 creat — create a new file or rewrite an existing one

7701 **SYNOPSIS**

7702 OH #include <sys/stat.h>

7703 #include <fcntl.h>

7704 int creat(const char *path, mode_t mode);

7705 **DESCRIPTION**

7706 The function call:

7707 creat(path, mode)

7708 shall be equivalent to:

7709 open(path, O_WRONLY|O_CREAT|O_TRUNC, mode)

7710 **RETURN VALUE**7711 Refer to *open()*.7712 **ERRORS**7713 Refer to *open()*.7714 **EXAMPLES**7715 **Creating a File**

7716 The following example creates the file **/tmp/file** with read and write permissions for the file
 7717 owner and read permission for group and others. The resulting file descriptor is assigned to the
 7718 *fd* variable.

7719 #include <fcntl.h>

7720 ...

7721 int fd;

7722 mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;

7723 char *filename = "/tmp/file";

7724 ...

7725 fd = creat(filename, mode);

7726 ...

7727 **APPLICATION USAGE**

7728 None.

7729 **RATIONALE**

7730 The *creat()* function is redundant. Its services are also provided by the *open()* function. It has
 7731 been included primarily for historical purposes since many existing applications depend on it. It
 7732 is best considered a part of the C binding rather than a function that should be provided in other
 7733 languages.

7734 **FUTURE DIRECTIONS**

7735 None.

7736 **SEE ALSO**

7737 *open()*, the Base Definitions volume of IEEE Std 1003.1-2001, <fcntl.h>, <sys/stat.h>,
 7738 <sys/types.h>

7739 **CHANGE HISTORY**

7740 First released in Issue 1. Derived from Issue 1 of the SVID.

7741 **Issue 6**

7742 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

7743 The following new requirements on POSIX implementations derive from alignment with the
7744 Single UNIX Specification:

- 7745 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
7746 required for conforming implementations of previous POSIX specifications, it was not
7747 required for UNIX applications.

7748 **NAME**7749 crypt — string encoding function (**CRYPT**)7750 **SYNOPSIS**

7751 xSI #include <unistd.h>

7752 char *crypt(const char *key, const char *salt);

7753

7754 **DESCRIPTION**7755 The *crypt()* function is a string encoding function. The algorithm is implementation-defined.7756 The *key* argument points to a string to be encoded. The *salt* argument is a string chosen from the
7757 set:

7758 a b c d e f g h i j k l m n o p q r s t u v w x y z

7759 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

7760 0 1 2 3 4 5 6 7 8 9 . /

7761 The first two characters of this string may be used to perturb the encoding algorithm.

7762 The return value of *crypt()* points to static data that is overwritten by each call.7763 The *crypt()* function need not be reentrant. A function that is not required to be reentrant is not
7764 required to be thread-safe.7765 **RETURN VALUE**7766 Upon successful completion, *crypt()* shall return a pointer to the encoded string. The first two
7767 characters of the returned value shall be those of the *salt* argument. Otherwise, it shall return a
7768 null pointer and set *errno* to indicate the error.7769 **ERRORS**7770 The *crypt()* function shall fail if:

7771 [ENOSYS] The functionality is not supported on this implementation.

7772 **EXAMPLES**7773 **Encoding Passwords**7774 The following example finds a user database entry matching a particular user name and changes
7775 the current password to a new password. The *crypt()* function generates an encoded version of
7776 each password. The first call to *crypt()* produces an encoded version of the old password; that
7777 encoded password is then compared to the password stored in the user database. The second
7778 call to *crypt()* encodes the new password before it is stored.7779 The *putpwent()* function, used in the following example, is not part of IEEE Std 1003.1-2001.

7780 #include <unistd.h>

7781 #include <pwd.h>

7782 #include <string.h>

7783 #include <stdio.h>

7784 ...

7785 int valid_change;

7786 int pfd; /* Integer for file descriptor returned by open(). */

7787 FILE *fpfd; /* File pointer for use in putpwent(). */

7788 struct passwd *p;

7789 char user[100];

7790 char oldpasswd[100];

7791 char newpasswd[100];

```
7792     char savepasswd[100];
7793     ...
7794     valid_change = 0;
7795     while ((p = getpwent()) != NULL) {
7796         /* Change entry if found. */
7797         if (strcmp(p->pw_name, user) == 0) {
7798             if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
7799                 strcpy(savepasswd, crypt(newpasswd, user));
7800                 p->pw_passwd = savepasswd;
7801                 valid_change = 1;
7802             }
7803             else {
7804                 fprintf(stderr, "Old password is not valid\n");
7805             }
7806         }
7807         /* Put passwd entry into ptmp. */
7808         putpwent(p, fpfd);
7809     }
```

7810 APPLICATION USAGE

7811 The values returned by this function need not be portable among XSI-conformant systems.

7812 RATIONALE

7813 None.

7814 FUTURE DIRECTIONS

7815 None.

7816 SEE ALSO

7817 *encrypt()*, *setkey()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

7818 CHANGE HISTORY

7819 First released in Issue 1. Derived from Issue 1 of the SVID.

7820 Issue 5

7821 Normative text previously in the APPLICATION USAGE section is moved to the
7822 DESCRIPTION.

7823 **NAME**

7824 csin, csinf, csinl — complex sine functions

7825 **SYNOPSIS**

7826 #include <complex.h>

7827 double complex csin(double complex *z*);7828 float complex csinf(float complex *z*);7829 long double complex csinl(long double complex *z*);7830 **DESCRIPTION**

7831 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
7832 conflict between the requirements described here and the ISO C standard is unintentional. This
7833 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7834 These functions shall compute the complex sine of *z*.7835 **RETURN VALUE**

7836 These functions shall return the complex sine value.

7837 **ERRORS**

7838 No errors are defined.

7839 **EXAMPLES**

7840 None.

7841 **APPLICATION USAGE**

7842 None.

7843 **RATIONALE**

7844 None.

7845 **FUTURE DIRECTIONS**

7846 None.

7847 **SEE ALSO**7848 *casin()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>7849 **CHANGE HISTORY**

7850 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7851 **NAME**

7852 csinh, csinhf, csinhl — complex hyperbolic sine functions

7853 **SYNOPSIS**

7854 #include <complex.h>

7855 double complex csinh(double complex z);

7856 float complex csinhf(float complex z);

7857 long double complex csinhl(long double complex z);

7858 **DESCRIPTION**

7859 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
7860 conflict between the requirements described here and the ISO C standard is unintentional. This
7861 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7862 These functions shall compute the complex hyperbolic sine of *z*.7863 **RETURN VALUE**

7864 These functions shall return the complex hyperbolic sine value.

7865 **ERRORS**

7866 No errors are defined.

7867 **EXAMPLES**

7868 None.

7869 **APPLICATION USAGE**

7870 None.

7871 **RATIONALE**

7872 None.

7873 **FUTURE DIRECTIONS**

7874 None.

7875 **SEE ALSO**7876 *casinh()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>7877 **CHANGE HISTORY**

7878 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7879 **NAME**

7880 csinl — complex sine functions

7881 **SYNOPSIS**

7882 #include <complex.h>

7883 long double complex csinl(long double complex z);

7884 **DESCRIPTION**7885 Refer to *csin()*.

7886 **NAME**

7887 csqrt, csqrtf, csqrtl — complex square root functions

7888 **SYNOPSIS**

7889 #include <complex.h>

7890 double complex csqrt(double complex z);

7891 float complex csqrtf(float complex z);

7892 long double complex csqrtl(long double complex z);

7893 **DESCRIPTION**

7894 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
7895 conflict between the requirements described here and the ISO C standard is unintentional. This
7896 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7897 These functions shall compute the complex square root of *z*, with a branch cut along the
7898 negative real axis.

7899 **RETURN VALUE**

7900 These functions shall return the complex square root value, in the range of the right half-plane
7901 (including the imaginary axis).

7902 **ERRORS**

7903 No errors are defined.

7904 **EXAMPLES**

7905 None.

7906 **APPLICATION USAGE**

7907 None.

7908 **RATIONALE**

7909 None.

7910 **FUTURE DIRECTIONS**

7911 None.

7912 **SEE ALSO**7913 *cabs()*, *cpow()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>7914 **CHANGE HISTORY**

7915 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7916 **NAME**

7917 ctan, ctanf, ctanl — complex tangent functions

7918 **SYNOPSIS**

7919 #include <complex.h>

7920 double complex ctan(double complex *z*);7921 float complex ctanf(float complex *z*);7922 long double complex ctanl(long double complex *z*);7923 **DESCRIPTION**7924 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
7925 conflict between the requirements described here and the ISO C standard is unintentional. This
7926 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.7927 These functions shall compute the complex tangent of *z*.7928 **RETURN VALUE**

7929 These functions shall return the complex tangent value.

7930 **ERRORS**

7931 No errors are defined.

7932 **EXAMPLES**

7933 None.

7934 **APPLICATION USAGE**

7935 None.

7936 **RATIONALE**

7937 None.

7938 **FUTURE DIRECTIONS**

7939 None.

7940 **SEE ALSO**7941 *catan()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>7942 **CHANGE HISTORY**

7943 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7944 **NAME**

7945 ctanh, ctanhf, ctanhl — complex hyperbolic tangent functions

7946 **SYNOPSIS**

7947 #include <complex.h>

7948 double complex ctanh(double complex z);

7949 float complex ctanhf(float complex z);

7950 long double complex ctanhl(long double complex z);

7951 **DESCRIPTION**

7952 cx The functionality described on this reference page is aligned with the ISO C standard. Any
7953 conflict between the requirements described here and the ISO C standard is unintentional. This
7954 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

7955 These functions shall compute the complex hyperbolic tangent of z .7956 **RETURN VALUE**

7957 These functions shall return the complex hyperbolic tangent value.

7958 **ERRORS**

7959 No errors are defined.

7960 **EXAMPLES**

7961 None.

7962 **APPLICATION USAGE**

7963 None.

7964 **RATIONALE**

7965 None.

7966 **FUTURE DIRECTIONS**

7967 None.

7968 **SEE ALSO**7969 *catanh()*, the Base Definitions volume of IEEE Std 1003.1-2001, <complex.h>7970 **CHANGE HISTORY**

7971 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7972 **NAME**

7973 ctanl — complex tangent functions

7974 **SYNOPSIS**

7975 #include <complex.h>

7976 long double complex ctanl(long double complex z);

7977 **DESCRIPTION**7978 Refer to *ctan()*.

7979 **NAME**7980 `ctermid` — generate a pathname for the controlling terminal7981 **SYNOPSIS**7982 `cx` `#include <stdio.h>`7983 `char *ctermid(char *s);`

7984

7985 **DESCRIPTION**7986 The `ctermid()` function shall generate a string that, when used as a pathname, refers to the
7987 current controlling terminal for the current process. If `ctermid()` returns a pathname, access to the
7988 file is not guaranteed.7989 If the application uses any of the `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS`
7990 functions, it shall ensure that the `ctermid()` function is called with a non-NULL parameter.7991 **RETURN VALUE**7992 If `s` is a null pointer, the string shall be generated in an area that may be static (and therefore may
7993 be overwritten by each call), the address of which shall be returned. Otherwise, `s` is assumed to
7994 point to a character array of at least `L_ctermid` bytes; the string is placed in this array and the
7995 value of `s` shall be returned. The symbolic constant `L_ctermid` is defined in `<stdio.h>`, and shall
7996 have a value greater than 0.7997 The `ctermid()` function shall return an empty string if the pathname that would refer to the
7998 controlling terminal cannot be determined, or if the function is unsuccessful.7999 **ERRORS**

8000 No errors are defined.

8001 **EXAMPLES**8002 **Determining the Controlling Terminal for the Current Process**8003 The following example returns a pointer to a string that identifies the controlling terminal for the
8004 current process. The pathname for the terminal is stored in the array pointed to by the `ptr`
8005 argument, which has a size of `L_ctermid` bytes, as indicated by the `term` argument.8006 `#include <stdio.h>`8007 `...`8008 `char term[L_ctermid];`8009 `char *ptr;`8010 `ptr = ctermid(term);`8011 **APPLICATION USAGE**8012 The difference between `ctermid()` and `ttyname()` is that `ttyname()` must be handed a file
8013 descriptor and return a path of the terminal associated with that file descriptor, while `ctermid()`
8014 returns a string (such as `"/dev/tty"`) that refers to the current controlling terminal if used as a
8015 pathname.8016 **RATIONALE**8017 `L_ctermid` must be defined appropriately for a given implementation and must be greater than
8018 zero so that array declarations using it are accepted by the compiler. The value includes the
8019 terminating null byte.8020 Conforming applications that use threads cannot call `ctermid()` with NULL as the parameter if
8021 either `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS` is defined. If `s` is not
8022 NULL, the `ctermid()` function generates a string that, when used as a pathname, refers to the

8023 current controlling terminal for the current process. If *s* is NULL, the return value of *ctermid()* is
8024 undefined.

8025 There is no additional burden on the programmer—changing to use a hypothetical thread-safe
8026 version of *ctermid()* along with allocating a buffer is more of a burden than merely allocating a
8027 buffer. Application code should not assume that the returned string is short, as some
8028 implementations have more than two pathname components before reaching a logical device
8029 name.

8030 **FUTURE DIRECTIONS**

8031 None.

8032 **SEE ALSO**

8033 *ttyname()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**stdio.h**>

8034 **CHANGE HISTORY**

8035 First released in Issue 1. Derived from Issue 1 of the SVID.

8036 **Issue 5**

8037 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

8038 **Issue 6**

8039 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

8040 **NAME**

8041 ctime, ctime_r — convert a time value to a date and time string

8042 **SYNOPSIS**

8043 #include <time.h>

8044 char *ctime(const time_t *clock);

8045 TSF char *ctime_r(const time_t *clock, char *buf);

8046

8047 **DESCRIPTION**8048 CX For *ctime()*: The functionality described on this reference page is aligned with the ISO C
8049 standard. Any conflict between the requirements described here and the ISO C standard is
8050 unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.8051 The *ctime()* function shall convert the time pointed to by *clock*, representing time in seconds
8052 since the Epoch, to local time in the form of a string. It shall be equivalent to:

8053 asctime(localtime(clock))

8054 CX The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static
8055 objects: a broken-down time structure and an array of **char**. Execution of any of the functions
8056 may overwrite the information returned in either of these objects by any of the other functions.8057 The *ctime()* function need not be reentrant. A function that is not required to be reentrant is not
8058 required to be thread-safe.8059 TSF The *ctime_r()* function shall convert the calendar time pointed to by *clock* to local time in exactly
8060 the same form as *ctime()* and put the string into the array pointed to by *buf* (which shall be at
8061 least 26 bytes in size) and return *buf*.8062 Unlike *ctime()*, the thread-safe version *ctime_r()* is not required to set *tzname*.8063 **RETURN VALUE**8064 The *ctime()* function shall return the pointer returned by *asctime()* with that broken-down time
8065 as an argument.8066 TSF Upon successful completion, *ctime_r()* shall return a pointer to the string pointed to by *buf*.
8067 When an error is encountered, a null pointer shall be returned.8068 **ERRORS**

8069 No errors are defined.

8070 **EXAMPLES**

8071 None.

8072 **APPLICATION USAGE**8073 Values for the broken-down time structure can be obtained by calling *gmtime()* or *localtime()*.
8074 The *ctime()* function is included for compatibility with older implementations, and does not
8075 support localized date and time formats. Applications should use the *strptime()* function to
8076 achieve maximum portability.8077 The *ctime_r()* function is thread-safe and shall return values in a user-supplied buffer instead of
8078 possibly using a static data area that may be overwritten by each call.8079 **RATIONALE**

8080 None.

8081 **FUTURE DIRECTIONS**

8082 None.

8083 **SEE ALSO**8084 *asctime()*, *clock()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*,
8085 the Base Definitions volume of IEEE Std 1003.1-2001, <**time.h**>8086 **CHANGE HISTORY**

8087 First released in Issue 1. Derived from Issue 1 of the SVID.

8088 **Issue 5**8089 Normative text previously in the APPLICATION USAGE section is moved to the
8090 DESCRIPTION.8091 The *ctime_r()* function is included for alignment with the POSIX Threads Extension.8092 A note indicating that the *ctime()* function need not be reentrant is added to the DESCRIPTION.8093 **Issue 6**

8094 Extensions beyond the ISO C standard are marked.

8095 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

8096 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
8097 its avoidance of possibly using a static data area.

8098 **NAME**
8099 daylight — daylight savings time flag

8100 **SYNOPSIS**

```
8101 xsi #include <time.h>
```

```
8102 extern int daylight;
```

8103

8104 **DESCRIPTION**

8105 Refer to *tzset()*.

8106 NAME

8107 dbm_clearerr, dbm_close, dbm_delete, dbm_error, dbm_fetch, dbm_firstkey, dbm_nextkey,
8108 dbm_open, dbm_store — database functions

8109 SYNOPSIS

```
8110 xSI #include <ndbm.h>
8111
8112 int dbm_clearerr(DBM *db);
8113 void dbm_close(DBM *db);
8114 int dbm_delete(DBM *db, datum key);
8115 int dbm_error(DBM *db);
8116 datum dbm_fetch(DBM *db, datum key);
8117 datum dbm_firstkey(DBM *db);
8118 datum dbm_nextkey(DBM *db);
8119 DBM *dbm_open(const char *file, int open_flags, mode_t file_mode);
8120 int dbm_store(DBM *db, datum key, datum content, int store_mode);
```

8121 DESCRIPTION

8122 These functions create, access, and modify a database.

8123 A **datum** consists of at least two members, *dptr* and *dsize*. The *dptr* member points to an object
8124 that is *dsize* bytes in length. Arbitrary binary data, as well as character strings, may be stored in
8125 the object pointed to by *dptr*.

8126 The database is stored in two files. One file is a directory containing a bitmap of keys and has
8127 **.dir** as its suffix. The second file contains all data and has **.pag** as its suffix.

8128 The *dbm_open()* function shall open a database. The *file* argument to the function is the
8129 pathname of the database. The function opens two files named *file.dir* and *file.pag*. The
8130 *open_flags* argument has the same meaning as the *flags* argument of *open()* except that a database
8131 opened for write-only access opens the files for read and write access and the behavior of the
8132 O_APPEND flag is unspecified. The *file_mode* argument has the same meaning as the third
8133 argument of *open()*.

8134 The *dbm_close()* function shall close a database. The application shall ensure that argument *db* is
8135 a pointer to a **dbm** structure that has been returned from a call to *dbm_open()*.

8136 These database functions shall support an internal block size large enough to support
8137 key/content pairs of at least 1023 bytes.

8138 The *dbm_fetch()* function shall read a record from a database. The argument *db* is a pointer to a
8139 database structure that has been returned from a call to *dbm_open()*. The argument *key* is a
8140 **datum** that has been initialized by the application to the value of the key that matches the key of
8141 the record the program is fetching.

8142 The *dbm_store()* function shall write a record to a database. The argument *db* is a pointer to a
8143 database structure that has been returned from a call to *dbm_open()*. The argument *key* is a
8144 **datum** that has been initialized by the application to the value of the key that identifies (for
8145 subsequent reading, writing, or deleting) the record the application is writing. The argument
8146 *content* is a **datum** that has been initialized by the application to the value of the record the
8147 program is writing. The argument *store_mode* controls whether *dbm_store()* replaces any pre-
8148 existing record that has the same key that is specified by the *key* argument. The application shall
8149 set *store_mode* to either DBM_INSERT or DBM_REPLACE. If the database contains a record that
8150 matches the *key* argument and *store_mode* is DBM_REPLACE, the existing record shall be
8151 replaced with the new record. If the database contains a record that matches the *key* argument
8152 and *store_mode* is DBM_INSERT, the existing record shall be left unchanged and the new record

8153 ignored. If the database does not contain a record that matches the *key* argument and *store_mode*
8154 is either DBM_INSERT or DBM_REPLACE, the new record shall be inserted in the database.

8155 If the sum of a key/content pair exceeds the internal block size, the result is unspecified.
8156 Moreover, the application shall ensure that all key/content pairs that hash together fit on a
8157 single block. The *dbm_store()* function shall return an error in the event that a disk block fills
8158 with inseparable data.

8159 The *dbm_delete()* function shall delete a record and its key from the database. The argument *db* is
8160 a pointer to a database structure that has been returned from a call to *dbm_open()*. The argument
8161 *key* is a **datum** that has been initialized by the application to the value of the key that identifies
8162 the record the program is deleting.

8163 The *dbm_firstkey()* function shall return the first key in the database. The argument *db* is a
8164 pointer to a database structure that has been returned from a call to *dbm_open()*.

8165 The *dbm_nextkey()* function shall return the next key in the database. The argument *db* is a
8166 pointer to a database structure that has been returned from a call to *dbm_open()*. The application
8167 shall ensure that the *dbm_firstkey()* function is called before calling *dbm_nextkey()*. Subsequent
8168 calls to *dbm_nextkey()* return the next key until all of the keys in the database have been
8169 returned.

8170 The *dbm_error()* function shall return the error condition of the database. The argument *db* is a
8171 pointer to a database structure that has been returned from a call to *dbm_open()*.

8172 The *dbm_clearerr()* function shall clear the error condition of the database. The argument *db* is a
8173 pointer to a database structure that has been returned from a call to *dbm_open()*.

8174 The *dptr* pointers returned by these functions may point into static storage that may be changed
8175 by subsequent calls.

8176 These functions need not be reentrant. A function that is not required to be reentrant is not
8177 required to be thread-safe.

8178 **RETURN VALUE**

8179 The *dbm_store()* and *dbm_delete()* functions shall return 0 when they succeed and a negative
8180 value when they fail.

8181 The *dbm_store()* function shall return 1 if it is called with a *flags* value of DBM_INSERT and the
8182 function finds an existing record with the same key.

8183 The *dbm_error()* function shall return 0 if the error condition is not set and return a non-zero
8184 value if the error condition is set.

8185 The return value of *dbm_clearerr()* is unspecified.

8186 The *dbm_firstkey()* and *dbm_nextkey()* functions shall return a key **datum**. When the end of the
8187 database is reached, the *dptr* member of the key is a null pointer. If an error is detected, the *dptr*
8188 member of the key shall be a null pointer and the error condition of the database shall be set.

8189 The *dbm_fetch()* function shall return a content **datum**. If no record in the database matches the
8190 key or if an error condition has been detected in the database, the *dptr* member of the content
8191 shall be a null pointer.

8192 The *dbm_open()* function shall return a pointer to a database structure. If an error is detected
8193 during the operation, *dbm_open()* shall return a **(DBM *)0**.

8194 **ERRORS**

8195 No errors are defined.

8196 **EXAMPLES**

8197 None.

8198 **APPLICATION USAGE**

8199 The following code can be used to traverse the database:

8200

```
for(key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))
```

8201 The *dbm_** functions provided in this library should not be confused in any way with those of a
8202 general-purpose database management system. These functions do not provide for multiple
8203 search keys per entry, they do not protect against multi-user access (in other words they do not
8204 lock records or files), and they do not provide the many other useful database functions that are
8205 found in more robust database management systems. Creating and updating databases by use of
8206 these functions is relatively slow because of data copies that occur upon hash collisions. These
8207 functions are useful for applications requiring fast lookup of relatively static information that is
8208 to be indexed by a single key.

8209 Note that a strictly conforming application is extremely limited by these functions: since there is
8210 no way to determine that the keys in use do not all hash to the same value (although that would
8211 be rare), a strictly conforming application cannot be guaranteed that it can store more than one
8212 block's worth of data in the database. As long as a key collision does not occur, additional data
8213 may be stored, but because there is no way to determine whether an error is due to a key
8214 collision or some other error condition (*dbm_error()* being effectively a Boolean), once an error is
8215 detected, the application is effectively limited to guessing what the error might be if it wishes to
8216 continue using these functions.

8217 The *dbm_delete()* function need not physically reclaim file space, although it does make it
8218 available for reuse by the database.

8219 After calling *dbm_store()* or *dbm_delete()* during a pass through the keys by *dbm_firstkey()* and
8220 *dbm_nextkey()*, the application should reset the database by calling *dbm_firstkey()* before again
8221 calling *dbm_nextkey()*. The contents of these files are unspecified and may not be portable.

8222 **RATIONALE**

8223 None.

8224 **FUTURE DIRECTIONS**

8225 None.

8226 **SEE ALSO**8227 *open()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**ndbm.h**>8228 **CHANGE HISTORY**

8229 First released in Issue 4, Version 2.

8230 **Issue 5**

8231 Moved from X/OPEN UNIX extension to BASE.

8232 Normative text previously in the APPLICATION USAGE section is moved to the
8233 DESCRIPTION.

8234 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

8235 **Issue 6**

8236

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

8237 **NAME**

8238 difftime — compute the difference between two calendar time values

8239 **SYNOPSIS**

8240 #include <time.h>

8241 double difftime(time_t *time1*, time_t *time0*);

8242 **DESCRIPTION**

8243 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
8244 conflict between the requirements described here and the ISO C standard is unintentional. This
8245 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

8246 The *difftime()* function shall compute the difference between two calendar times (as returned by
8247 *time()*): *time1*– *time0*.

8248 **RETURN VALUE**

8249 The *difftime()* function shall return the difference expressed in seconds as a type **double**.

8250 **ERRORS**

8251 No errors are defined.

8252 **EXAMPLES**

8253 None.

8254 **APPLICATION USAGE**

8255 None.

8256 **RATIONALE**

8257 None.

8258 **FUTURE DIRECTIONS**

8259 None.

8260 **SEE ALSO**

8261 *asctime()*, *clock()*, *ctime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*,
8262 the Base Definitions volume of IEEE Std 1003.1-2001, <**time.h**>

8263 **CHANGE HISTORY**

8264 First released in Issue 4. Derived from the ISO C standard.

8265 **NAME**

8266 dirname — report the parent directory name of a file pathname

8267 **SYNOPSIS**

8268 XSI #include <libgen.h>

8269 char *dirname(char *path);

8270

8271 **DESCRIPTION**8272 The *dirname()* function shall take a pointer to a character string that contains a pathname, and
8273 return a pointer to a string that is a pathname of the parent directory of that file. Trailing '/'
8274 characters in the path are not counted as part of the path.8275 If *path* does not contain a '/', then *dirname()* shall return a pointer to the string ".". If *path* is a
8276 null pointer or points to an empty string, *dirname()* shall return a pointer to the string ".".8277 The *dirname()* function need not be reentrant. A function that is not required to be reentrant is
8278 not required to be thread-safe.8279 **RETURN VALUE**8280 The *dirname()* function shall return a pointer to a string that is the parent directory of *path*. If
8281 *path* is a null pointer or points to an empty string, a pointer to a string "." is returned.8282 The *dirname()* function may modify the string pointed to by *path*, and may return a pointer to
8283 static storage that may then be overwritten by subsequent calls to *dirname()*.8284 **ERRORS**

8285 No errors are defined.

8286 **EXAMPLES**8287 The following code fragment reads a pathname, changes the current working directory to the
8288 parent directory, and opens the file.8289 char path[PATH_MAX], *pathcopy;
8290 int fd;
8291 fgets(path, PATH_MAX, stdin);
8292 pathcopy = strdup(path);
8293 chdir(dirname(pathcopy));
8294 fd = open(basename(path), O_RDONLY);8295 **Sample Input and Output Strings for dirname()**8296 In the following table, the input string is the value pointed to by *path*, and the output string is
8297 the return value of the *dirname()* function.

Input String	Output String
"/usr/lib"	"/usr"
"/usr/"	"/"
"usr"	."
"/"	"/"
."	."
".."	."

8305 **Changing the Current Directory to the Parent Directory**

8306 The following program fragment reads a pathname, changes the current working directory to
8307 the parent directory, and opens the file.

```
8308 #include <unistd.h>
8309 #include <limits.h>
8310 #include <stdio.h>
8311 #include <fcntl.h>
8312 #include <string.h>
8313 #include <libgen.h>
8314 ...
8315 char path[PATH_MAX], *pathcopy;
8316 int fd;
8317 ...
8318 fgets(path, PATH_MAX, stdin);
8319 pathcopy = strdup(path);
8320 chdir(dirname(pathcopy));
8321 fd = open(basename(path), O_RDONLY);
```

8322 **APPLICATION USAGE**

8323 The *dirname()* and *basename()* functions together yield a complete pathname. The expression
8324 *dirname(path)* obtains the pathname of the directory where *basename(path)* is found.

8325 Since the meaning of the leading *"/"* is implementation-defined, *dirname("/foo)* may return
8326 either *"/"* or *'/'* (but nothing else).

8327 **RATIONALE**

8328 None.

8329 **FUTURE DIRECTIONS**

8330 None.

8331 **SEE ALSO**

8332 *basename()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<libgen.h>**

8333 **CHANGE HISTORY**

8334 First released in Issue 4, Version 2.

8335 **Issue 5**

8336 Moved from X/OPEN UNIX extension to BASE.

8337 Normative text previously in the APPLICATION USAGE section is moved to the
8338 DESCRIPTION.

8339 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

8340 **NAME**

8341 `div` — compute the quotient and remainder of an integer division

8342 **SYNOPSIS**

8343 `#include <stdlib.h>`

8344 `div_t div(int numer, int denom);`

8345 **DESCRIPTION**

8346 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
8347 conflict between the requirements described here and the ISO C standard is unintentional. This
8348 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

8349 The `div()` function shall compute the quotient and remainder of the division of the numerator
8350 *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the integer
8351 of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be
8352 represented, the behavior is undefined; otherwise, *quot*denom+rem* shall equal *numer*.

8353 **RETURN VALUE**

8354 The `div()` function shall return a structure of type `div_t`, comprising both the quotient and the
8355 remainder. The structure includes the following members, in any order:

8356 `int quot; /* quotient */`

8357 `int rem; /* remainder */`

8358 **ERRORS**

8359 No errors are defined.

8360 **EXAMPLES**

8361 None.

8362 **APPLICATION USAGE**

8363 None.

8364 **RATIONALE**

8365 None.

8366 **FUTURE DIRECTIONS**

8367 None.

8368 **SEE ALSO**

8369 `ldiv()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<stdlib.h>`

8370 **CHANGE HISTORY**

8371 First released in Issue 4. Derived from the ISO C standard.

8372 **NAME**8373 dldclose — close a *dlopen()* object8374 **SYNOPSIS**

8375 XSI #include <dldfcn.h>

8376 int dldclose(void *handle);

8377

8378 **DESCRIPTION**8379 The *dldclose()* function shall inform the system that the object referenced by a *handle* returned
8380 from a previous *dlopen()* invocation is no longer needed by the application.8381 The use of *dldclose()* reflects a statement of intent on the part of the process, but does not create
8382 any requirement upon the implementation, such as removal of the code or symbols referenced
8383 by *handle*. Once an object has been closed using *dldclose()* an application should assume that its
8384 symbols are no longer available to *dlsym()*. All objects loaded automatically as a result of
8385 invoking *dlopen()* on the referenced object shall also be closed if this is the last reference to it.8386 Although a *dldclose()* operation is not required to remove structures from an address space,
8387 neither is an implementation prohibited from doing so. The only restriction on such a removal is
8388 that no object shall be removed to which references have been relocated, until or unless all such
8389 references are removed. For instance, an object that had been loaded with a *dlopen()* operation
8390 specifying the `RTLD_GLOBAL` flag might provide a target for dynamic relocations performed in
8391 the processing of other objects—in such environments, an application may assume that no
8392 relocation, once made, shall be undone or remade unless the object requiring the relocation has
8393 itself been removed.8394 **RETURN VALUE**8395 If the referenced object was successfully closed, *dldclose()* shall return 0. If the object could not be
8396 closed, or if *handle* does not refer to an open object, *dldclose()* shall return a non-zero value. More
8397 detailed diagnostic information shall be available through *dlderror()*.8398 **ERRORS**

8399 No errors are defined.

8400 **EXAMPLES**8401 The following example illustrates use of *dlopen()* and *dldclose()*:8402 ...
8403 /* Open a dynamic library and then close it ... */
8404 #include <dldfcn.h>
8405 void *mylib;
8406 int eret;
8407 mylib = dlopen("mylib.so", RTLD_LOCAL | RTLD_LAZY);
8408 ...
8409 eret = dldclose(mylib);
8410 ...8411 **APPLICATION USAGE**8412 A conforming application should employ a *handle* returned from a *dlopen()* invocation only
8413 within a given scope bracketed by the *dlopen()* and *dldclose()* operations. Implementations are
8414 free to use reference counting or other techniques such that multiple calls to *dlopen()* referencing
8415 the same object may return the same object for *handle*. Implementations are also free to reuse a
8416 *handle*. For these reasons, the value of a *handle* must be treated as an opaque object by the
8417 application, used only in calls to *dlsym()* and *dldclose()*.

8418 **RATIONALE**

8419 None.

8420 **FUTURE DIRECTIONS**

8421 None.

8422 **SEE ALSO**8423 *derror()*, *dlopen()*, *dlsym()*, the Base Definitions volume of IEEE Std 1003.1-2001, <dlfcn.h>8424 **CHANGE HISTORY**

8425 First released in Issue 5.

8426 **Issue 6**8427 The DESCRIPTION is updated to say that the referenced object is closed “if this is the last
8428 reference to it”.

8429 **NAME**8430 `dlderror` — get diagnostic information8431 **SYNOPSIS**8432 XSI `#include <dldfcn.h>`8433 `char *dlderror(void);`

8434

8435 **DESCRIPTION**

8436 The `dlderror()` function shall return a null-terminated character string (with no trailing <newline>) that describes the last error that occurred during dynamic linking processing. If no dynamic linking errors have occurred since the last invocation of `dlderror()`, `dlderror()` shall return NULL. Thus, invoking `dlderror()` a second time, immediately following a prior invocation, shall result in NULL being returned.

8441 The `dlderror()` function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

8443 **RETURN VALUE**

8444 If successful, `dlderror()` shall return a null-terminated character string; otherwise, NULL shall be returned.

8446 **ERRORS**

8447 No errors are defined.

8448 **EXAMPLES**

8449 The following example prints out the last dynamic linking error:

```
8450 ...
8451 #include <dldfcn.h>
8452 char *errstr;
8453 errstr = dlderror();
8454 if (errstr != NULL)
8455     printf ("A dynamic linking error occurred: (%s)\n", errstr);
8456 ...
```

8457 **APPLICATION USAGE**

8458 The messages returned by `dlderror()` may reside in a static buffer that is overwritten on each call to `dlderror()`. Application code should not write to this buffer. Programs wishing to preserve an error message should make their own copies of that message. Depending on the application environment with respect to asynchronous execution events, such as signals or other asynchronous computation sharing the address space, conforming applications should use a critical section to retrieve the error pointer and buffer.

8464 **RATIONALE**

8465 None.

8466 **FUTURE DIRECTIONS**

8467 None.

8468 **SEE ALSO**8469 `dldclose()`, `dldopen()`, `dldsym()`, the Base Definitions volume of IEEE Std 1003.1-2001, <dldfcn.h>

8470 **CHANGE HISTORY**

8471 First released in Issue 5.

8472 **Issue 6**

8473 In the DESCRIPTION the note about reentrancy and thread-safety is added.

8474 NAME

8475 dlopen — gain access to an executable object file

8476 SYNOPSIS

8477 XSI `#include <dlfcn.h>`8478 `void *dlopen(const char *file, int mode);`

8479

8480 DESCRIPTION

8481 The *dlopen()* function shall make an executable object file specified by *file* available to the calling
 8482 program. The class of files eligible for this operation and the manner of their construction are
 8483 implementation-defined, though typically such files are executable objects such as shared
 8484 libraries, relocatable files, or programs. Note that some implementations permit the construction
 8485 of dependencies between such objects that are embedded within files. In such cases, a *dlopen()*
 8486 operation shall load such dependencies in addition to the object referenced by *file*.
 8487 Implementations may also impose specific constraints on the construction of programs that can
 8488 employ *dlopen()* and its related services.

8489 A successful *dlopen()* shall return a *handle* which the caller may use on subsequent calls to
 8490 *dlsym()* and *dlclose()*. The value of this *handle* should not be interpreted in any way by the caller.

8491 The *file* argument is used to construct a pathname to the object file. If *file* contains a slash
 8492 character, the *file* argument is used as the pathname for the file. Otherwise, *file* is used in an
 8493 implementation-defined manner to yield a pathname.

8494 If the value of *file* is 0, *dlopen()* shall provide a *handle* on a global symbol object. This object shall
 8495 provide access to the symbols from an ordered set of objects consisting of the original program
 8496 image file, together with any objects loaded at program start-up as specified by that process
 8497 image file (for example, shared libraries), and the set of objects loaded using a *dlopen()* operation
 8498 together with the RTLD_GLOBAL flag. As the latter set of objects can change during execution,
 8499 the set identified by *handle* can also change dynamically.

8500 Only a single copy of an object file is brought into the address space, even if *dlopen()* is invoked
 8501 multiple times in reference to the file, and even if different pathnames are used to reference the
 8502 file.

8503 The *mode* parameter describes how *dlopen()* shall operate upon *file* with respect to the processing
 8504 of relocations and the scope of visibility of the symbols provided within *file*. When an object is
 8505 brought into the address space of a process, it may contain references to symbols whose
 8506 addresses are not known until the object is loaded. These references shall be relocated before the
 8507 symbols can be accessed. The *mode* parameter governs when these relocations take place and
 8508 may have the following values:

8509 RTLD_LAZY Relocations shall be performed at an implementation-defined time,
 8510 ranging from the time of the *dlopen()* call until the first reference to a
 8511 given symbol occurs. Specifying RTLD_LAZY should improve
 8512 performance on implementations supporting dynamic symbol binding as
 8513 a process may not reference all of the functions in any given object. And,
 8514 for systems supporting dynamic symbol resolution for normal process
 8515 execution, this behavior mimics the normal handling of process
 8516 execution.

8517 RTLD_NOW All necessary relocations shall be performed when the object is first
 8518 loaded. This may waste some processing if relocations are performed for
 8519 functions that are never referenced. This behavior may be useful for
 8520 applications that need to know as soon as an object is loaded that all

8521 symbols referenced during execution are available.

8522 Any object loaded by *dlopen()* that requires relocations against global symbols can reference the
8523 symbols in the original process image file, any objects loaded at program start-up, from the
8524 object itself as well as any other object included in the same *dlopen()* invocation, and any objects
8525 that were loaded in any *dlopen()* invocation and which specified the RTLD_GLOBAL flag. To
8526 determine the scope of visibility for the symbols loaded with a *dlopen()* invocation, the *mode*
8527 parameter should be a bitwise-inclusive OR with one of the following values:

8528 RTLD_GLOBAL The object's symbols shall be made available for the relocation processing
8529 of any other object. In addition, symbol lookup using *dlopen(0, mode)* and
8530 an associated *dlsym()* allows objects loaded with this *mode* to be searched.

8531 RTLD_LOCAL The object's symbols shall not be made available for the relocation
8532 processing of any other object.

8533 If neither RTLD_GLOBAL nor RTLD_LOCAL are specified, then an implementation-defined
8534 default behavior shall be applied.

8535 If a *file* is specified in multiple *dlopen()* invocations, *mode* is interpreted at each invocation. Note,
8536 however, that once RTLD_NOW has been specified all relocations shall have been completed
8537 rendering further RTLD_NOW operations redundant and any further RTLD_LAZY operations
8538 irrelevant. Similarly, note that once RTLD_GLOBAL has been specified the object shall maintain
8539 the RTLD_GLOBAL status regardless of any previous or future specification of RTLD_LOCAL,
8540 as long as the object remains in the address space (see *dlclose()*).

8541 Symbols introduced into a program through calls to *dlopen()* may be used in relocation
8542 activities. Symbols so introduced may duplicate symbols already defined by the program or
8543 previous *dlopen()* operations. To resolve the ambiguities such a situation might present, the
8544 resolution of a symbol reference to symbol definition is based on a symbol resolution order. Two
8545 such resolution orders are defined: *load* or *dependency* ordering. Load order establishes an
8546 ordering among symbol definitions, such that the definition first loaded (including definitions
8547 from the image file and any dependent objects loaded with it) has priority over objects added
8548 later (via *dlopen()*). Load ordering is used in relocation processing. Dependency ordering uses a
8549 breadth-first order starting with a given object, then all of its dependencies, then any dependents
8550 of those, iterating until all dependencies are satisfied. With the exception of the global symbol
8551 object obtained via a *dlopen()* operation on a *file* of 0, dependency ordering is used by the
8552 *dlsym()* function. Load ordering is used in *dlsym()* operations upon the global symbol object.

8553 When an object is first made accessible via *dlopen()* it and its dependent objects are added in
8554 dependency order. Once all the objects are added, relocations are performed using load order.
8555 Note that if an object or its dependencies had been previously loaded, the load and dependency
8556 orders may yield different resolutions.

8557 The symbols introduced by *dlopen()* operations and available through *dlsym()* are at a minimum
8558 those which are exported as symbols of global scope by the object. Typically such symbols shall
8559 be those that were specified in (for example) C source code as having *extern* linkage. The precise
8560 manner in which an implementation constructs the set of exported symbols for a *dlopen()* object
8561 is specified by that implementation.

8562 **RETURN VALUE**

8563 If *file* cannot be found, cannot be opened for reading, is not of an appropriate object format for
8564 processing by *dlopen()*, or if an error occurs during the process of loading *file* or relocating its
8565 symbolic references, *dlopen()* shall return NULL. More detailed diagnostic information shall be
8566 available through *dlerror()*.

8567 **ERRORS**

8568 No errors are defined.

8569 **EXAMPLES**

8570 None.

8571 **APPLICATION USAGE**

8572 None.

8573 **RATIONALE**

8574 None.

8575 **FUTURE DIRECTIONS**

8576 None.

8577 **SEE ALSO**

8578 *dlclose()*, *dLError()*, *dlsym()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**dlfcn.h**>

8579 **CHANGE HISTORY**

8580 First released in Issue 5.

8581 **NAME**8582 dlsym — obtain the address of a symbol from a *dlopen()* object8583 **SYNOPSIS**

8584 XSI #include <dlfcn.h>

8585 void *dlsym(void *restrict *handle*, const char *restrict *name*);

8586

8587 **DESCRIPTION**

8588 The *dlsym()* function shall obtain the address of a symbol defined within an object made
 8589 accessible through a *dlopen()* call. The *handle* argument is the value returned from a call to
 8590 *dlopen()* (and which has not since been released via a call to *dlclose()*), and *name* is the symbol's
 8591 name as a character string.

8592 The *dlsym()* function shall search for the named symbol in all objects loaded automatically as a
 8593 result of loading the object referenced by *handle* (see *dlopen()*). Load ordering is used in *dlsym()*
 8594 operations upon the global symbol object. The symbol resolution algorithm used shall be
 8595 dependency order as described in *dlopen()*.

8596 The RTLD_DEFAULT and RTLD_NEXT flags are reserved for future use.

8597 **RETURN VALUE**

8598 If *handle* does not refer to a valid object opened by *dlopen()*, or if the named symbol cannot be
 8599 found within any of the objects associated with *handle*, *dlsym()* shall return NULL. More
 8600 detailed diagnostic information shall be available through *dlerror()*.

8601 **ERRORS**

8602 No errors are defined.

8603 **EXAMPLES**

8604 The following example shows how *dlopen()* and *dlsym()* can be used to access either function or
 8605 data objects. For simplicity, error checking has been omitted.

```
8606 void *handle;
8607 int *iptr, (*fptr)(int);

8608 /* open the needed object */
8609 handle = dlopen("/usr/home/me/libfoo.so", RTLD_LOCAL | RTLD_LAZY);

8610 /* find the address of function and data objects */
8611 *(void **)(&fptr) = dlsym(handle, "my_function");
8612 iptr = (int *)dlsym(handle, "my_object");

8613 /* invoke function, passing value of integer as a parameter */
8614 (*fptr)(*iptr);
```

8615 **APPLICATION USAGE**

8616 Special purpose values for *handle* are reserved for future use. These values and their meanings
 8617 are:

8618 **RTLD_DEFAULT** The symbol lookup happens in the normal global scope; that is, a search for a
 8619 symbol using this handle would find the same definition as a direct use of this
 8620 symbol in the program code.

8621 **RTLD_NEXT** Specifies the next object after this one that defines *name*. *This one* refers to the
 8622 object containing the invocation of *dlsym()*. The *next* object is the one found
 8623 upon the application of a load order symbol resolution algorithm (see
 8624 *dlopen()*). The next object is either one of global scope (because it was
 8625 introduced as part of the original process image or because it was added with

8626 a *dlopen()* operation including the RTLD_GLOBAL flag), or is an object that
8627 was included in the same *dlopen()* operation that loaded this one.

8628 The RTLD_NEXT flag is useful to navigate an intentionally created hierarchy
8629 of multiply-defined symbols created through *interposition*. For example, if a
8630 program wished to create an implementation of *malloc()* that embedded some
8631 statistics gathering about memory allocations, such an implementation could
8632 use the real *malloc()* definition to perform the memory allocation—and itself
8633 only embed the necessary logic to implement the statistics gathering function.

8634 RATIONALE

8635 The ISO C standard does not require that pointers to functions can be cast back and forth to
8636 pointers to data. Indeed, the ISO C standard does not require that an object of type **void *** can
8637 hold a pointer to a function. Implementations supporting the XSI extension, however, do require
8638 that an object of type **void *** can hold a pointer to a function. The result of converting a pointer to
8639 a function into a pointer to another data type (except **void ***) is still undefined, however. Note
8640 that compilers conforming to the ISO C standard are required to generate a warning if a
8641 conversion from a **void *** pointer to a function pointer is attempted as in:

```
8642 fptr = (int (*)(int))dlsym(handle, "my_function");
```

8643 Due to the problem noted here, a future version may either add a new function to return
8644 function pointers, or the current interface may be deprecated in favor of two new functions: one
8645 that returns data pointers and the other that returns function pointers.

8646 FUTURE DIRECTIONS

8647 None.

8648 SEE ALSO

8649 *dlclose()*, *dLError()*, *dlopen()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**dlfcn.h**>

8650 CHANGE HISTORY

8651 First released in Issue 5.

8652 Issue 6

8653 The **restrict** keyword is added to the *dlsym()* prototype for alignment with the
8654 ISO/IEC 9899:1999 standard.

8655 The RTLD_DEFAULT and RTLD_NEXT flags are reserved for future use.

8656 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/14 is applied, correcting an example, and
8657 adding text to the RATIONALE describing issues related to conversion of pointers to functions
8658 and back again.

8659 NAME

8660 drand48, erand48, jrand48, lcong48, lrand48, mrand48, nrand48, seed48, srand48 — generate
8661 uniformly distributed pseudo-random numbers

8662 SYNOPSIS

```
8663 xSI #include <stdlib.h>
8664
8664 double drand48(void);
8665 double erand48(unsigned short xsubi[3]);
8666 long jrand48(unsigned short xsubi[3]);
8667 void lcong48(unsigned short param[7]);
8668 long lrand48(void);
8669 long mrand48(void);
8670 long nrand48(unsigned short xsubi[3]);
8671 unsigned short *seed48(unsigned short seed16v[3]);
8672 void srand48(long seedval);
8673
```

8674 DESCRIPTION

8675 This family of functions shall generate pseudo-random numbers using a linear congruential
8676 algorithm and 48-bit integer arithmetic.

8677 The *drand48()* and *erand48()* functions shall return non-negative, double-precision, floating-
8678 point values, uniformly distributed over the interval [0.0,1.0).

8679 The *lrand48()* and *nrand48()* functions shall return non-negative, long integers, uniformly
8680 distributed over the interval $[0, 2^{31})$.

8681 The *mrnd48()* and *jrnd48()* functions shall return signed long integers uniformly distributed
8682 over the interval $[-2^{31}, 2^{31})$.

8683 The *srand48()*, *seed48()*, and *lcong48()* functions are initialization entry points, one of which
8684 should be invoked before either *drand48()*, *lrand48()*, or *mrnd48()* is called. (Although it is not
8685 recommended practice, constant default initializer values shall be supplied automatically if
8686 *drand48()*, *lrand48()*, or *mrnd48()* is called without a prior call to an initialization entry point.)
8687 The *erand48()*, *nrand48()*, and *jrnd48()* functions do not require an initialization entry point to
8688 be called first.

8689 All the routines work by generating a sequence of 48-bit integer values, X_i , according to the
8690 linear congruential formula:

$$8691 \quad X_{n+1} = (aX_n + c)_{\text{mod } m} \quad n \geq 0$$

8692 The parameter $m = 2^{48}$; hence 48-bit integer arithmetic is performed. Unless *lcong48()* is invoked,
8693 the multiplier value a and the addend value c are given by:

$$8694 \quad a = 5\text{DEECE66D}_{16} = 273673163155_8$$

$$8695 \quad c = \text{B}_{16} = 13_8$$

8696 The value returned by any of the *drand48()*, *erand48()*, *jrnd48()*, *lrand48()*, *mrnd48()*, or
8697 *nrand48()* functions is computed by first generating the next 48-bit X_i in the sequence. Then the
8698 appropriate number of bits, according to the type of data item to be returned, are copied from
8699 the high-order (leftmost) bits of X_i and transformed into the returned value.

8700 The *drand48()*, *lrand48()*, and *mrnd48()* functions store the last 48-bit X_i generated in an
8701 internal buffer; that is why the application shall ensure that these are initialized prior to being
8702 invoked. The *erand48()*, *nrand48()*, and *jrnd48()* functions require the calling program to
8703 provide storage for the successive X_i values in the array specified as an argument when the

8704 functions are invoked. That is why these routines do not have to be initialized; the calling
 8705 program merely has to place the desired initial value of X_i into the array and pass it as an
 8706 argument. By using different arguments, *erand48()*, *rand48()*, and *jrand48()* allow separate
 8707 modules of a large program to generate several *independent* streams of pseudo-random numbers;
 8708 that is, the sequence of numbers in each stream shall *not* depend upon how many times the
 8709 routines are called to generate numbers for the other streams.

8710 The initializer function *srand48()* sets the high-order 32 bits of X_i to the low-order 32 bits
 8711 contained in its argument. The low-order 16 bits of X_i are set to the arbitrary value $330E_{16}$.

8712 The initializer function *seed48()* sets the value of X_i to the 48-bit value specified in the argument
 8713 array. The low-order 16 bits of X_i are set to the low-order 16 bits of *seed16v[0]*. The mid-order 16
 8714 bits of X_i are set to the low-order 16 bits of *seed16v[1]*. The high-order 16 bits of X_i are set to the
 8715 low-order 16 bits of *seed16v[2]*. In addition, the previous value of X_i is copied into a 48-bit
 8716 internal buffer, used only by *seed48()*, and a pointer to this buffer is the value returned by
 8717 *seed48()*. This returned pointer, which can just be ignored if not needed, is useful if a program is
 8718 to be restarted from a given point at some future time—use the pointer to get at and store the
 8719 last X_i value, and then use this value to reinitialize via *seed48()* when the program is restarted.

8720 The initializer function *lcg48()* allows the user to specify the initial X_i , the multiplier value a ,
 8721 and the addend value c . Argument array elements *param[0-2]* specify X_i , *param[3-5]* specify the
 8722 multiplier a , and *param[6]* specifies the 16-bit addend c . After *lcg48()* is called, a subsequent
 8723 call to either *srand48()* or *seed48()* shall restore the standard multiplier and addend values, a and
 8724 c , specified above.

8725 The *drand48()*, *lrnd48()*, and *mrnd48()* functions need not be reentrant. A function that is not
 8726 required to be reentrant is not required to be thread-safe.

8727 RETURN VALUE

8728 As described in the DESCRIPTION above.

8729 ERRORS

8730 No errors are defined.

8731 EXAMPLES

8732 None.

8733 APPLICATION USAGE

8734 None.

8735 RATIONALE

8736 None.

8737 FUTURE DIRECTIONS

8738 None.

8739 SEE ALSO

8740 *rand()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>

8741 CHANGE HISTORY

8742 First released in Issue 1. Derived from Issue 1 of the SVID.

8743 Issue 5

8744 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

8745 Issue 6

8746 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

8747 **NAME**

8748 dup, dup2 — duplicate an open file descriptor

8749 **SYNOPSIS**

8750 #include <unistd.h>

8751 int dup(int *fildes*);8752 int dup2(int *fildes*, int *fildes2*);8753 **DESCRIPTION**8754 The *dup()* and *dup2()* functions provide an alternative interface to the service provided by *fcntl()* using the F_DUPFD command. The call:8756 fid = dup(*fildes*);

8757 shall be equivalent to:

8758 fid = fcntl(*fildes*, F_DUPFD, 0);

8759 The call:

8760 fid = dup2(*fildes*, *fildes2*);

8761 shall be equivalent to:

8762 close(*fildes2*);8763 fid = fcntl(*fildes*, F_DUPFD, *fildes2*);

8764 except for the following:

- 8765 • If *fildes2* is less than 0 or greater than or equal to {OPEN_MAX}, *dup2()* shall return -1 with
- 8766 *errno* set to [EBADF].
- 8767 • If *fildes* is a valid file descriptor and is equal to *fildes2*, *dup2()* shall return *fildes2* without
- 8768 closing it.
- 8769 • If *fildes* is not a valid file descriptor, *dup2()* shall return -1 and shall not close *fildes2*.
- 8770 • The value returned shall be equal to the value of *fildes2* upon successful completion, or -1
- 8771 upon failure.

8772 **RETURN VALUE**8773 Upon successful completion a non-negative integer, namely the file descriptor, shall be returned;
8774 otherwise, -1 shall be returned and *errno* set to indicate the error.8775 **ERRORS**8776 The *dup()* function shall fail if:8777 [EBADF] The *fildes* argument is not a valid open file descriptor.8778 [EMFILE] The number of file descriptors in use by this process would exceed
8779 {OPEN_MAX}.8780 The *dup2()* function shall fail if:8781 [EBADF] The *fildes* argument is not a valid open file descriptor or the argument *fildes2* is
8782 negative or greater than or equal to {OPEN_MAX}.8783 [EINTR] The *dup2()* function was interrupted by a signal.

8784 **EXAMPLES**8785 **Redirecting Standard Output to a File**

8786 The following example closes standard output for the current processes, re-assigns standard
8787 output to go to the file referenced by *pfid*, and closes the original file descriptor to clean up.

```
8788 #include <unistd.h>
8789 ...
8790 int pfd;
8791 ...
8792 close(1);
8793 dup(pfd);
8794 close(pfd);
8795 ...
```

8796 **Redirecting Error Messages**

8797 The following example redirects messages from *stderr* to *stdout*.

```
8798 #include <unistd.h>
8799 ...
8800 dup2(1, 2);
8801 ...
```

8802 **APPLICATION USAGE**

8803 None.

8804 **RATIONALE**

8805 The *dup()* and *dup2()* functions are redundant. Their services are also provided by the *fcntl()*
8806 function. They have been included in this volume of IEEE Std 1003.1-2001 primarily for historical
8807 reasons, since many existing applications use them.

8808 While the brief code segment shown is very similar in behavior to *dup2()*, a conforming
8809 implementation based on other functions defined in this volume of IEEE Std 1003.1-2001 is
8810 significantly more complex. Least obvious is the possible effect of a signal-catching function that
8811 could be invoked between steps and allocate or deallocate file descriptors. This could be avoided
8812 by blocking signals.

8813 The *dup2()* function is not marked obsolescent because it presents a type-safe version of
8814 functionality provided in a type-unsafe version by *fcntl()*. It is used in the POSIX Ada binding.

8815 The *dup2()* function is not intended for use in critical regions as a synchronization mechanism.

8816 In the description of [EBADF], the case of *fildes* being out of range is covered by the given case of
8817 *fildes* not being valid. The descriptions for *fildes* and *fildes2* are different because the only kind of
8818 invalidity that is relevant for *fildes2* is whether it is out of range; that is, it does not matter
8819 whether *fildes2* refers to an open file when the *dup2()* call is made.

8820 **FUTURE DIRECTIONS**

8821 None.

8822 **SEE ALSO**

8823 *close()*, *fcntl()*, *open()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

8824 **CHANGE HISTORY**

8825 First released in Issue 1. Derived from Issue 1 of the SVID.

8826 NAME

8827 ecvt, fcvt, gcvt — convert a floating-point number to a string (LEGACY)

8828 SYNOPSIS

8829 XSI `#include <stdlib.h>`8830 `char *ecvt(double value, int ndigit, int *restrict decpt,`
8831 `int *restrict sign);`8832 `char *fcvt(double value, int ndigit, int *restrict decpt,`
8833 `int *restrict sign);`8834 `char *gcvt(double value, int ndigit, char *buf);`

8835

8836 DESCRIPTION

8837 The *ecvt()*, *fcvt()*, and *gcvt()* functions shall convert floating-point numbers to null-terminated
8838 strings.8839 The *ecvt()* function shall convert *value* to a null-terminated string of *ndigit* digits (where *ndigit* is
8840 reduced to an unspecified limit determined by the precision of a **double**) and return a pointer to
8841 the string. The high-order digit shall be non-zero, unless the value is 0. The low-order digit shall
8842 be rounded in an implementation-defined manner. The position of the radix character relative to
8843 the beginning of the string shall be stored in the integer pointed to by *decpt* (negative means to
8844 the left of the returned digits). If *value* is zero, it is unspecified whether the integer pointed to by
8845 *decpt* would be 0 or 1. The radix character shall not be included in the returned string. If the sign
8846 of the result is negative, the integer pointed to by *sign* shall be non-zero; otherwise, it shall be 0.8847 If the converted value is out of range or is not representable, the contents of the returned string
8848 are unspecified.8849 The *fcvt()* function shall be equivalent to *ecvt()*, except that *ndigit* specifies the number of digits
8850 desired after the radix character. The total number of digits in the result string is restricted to an
8851 unspecified limit as determined by the precision of a **double**.8852 The *gcvt()* function shall convert *value* to a null-terminated string (similar to that of the %g
8853 conversion specification format of *printf()*) in the array pointed to by *buf* and shall return *buf*. It
8854 shall produce *ndigit* significant digits (limited to an unspecified value determined by the
8855 precision of a **double**) in the %E conversion specification format of *printf()* if possible, or the %e
8856 conversion specification format of *printf()* (scientific notation) otherwise. A minus sign shall be
8857 included in the returned string if *value* is less than 0. A radix character shall be included in the
8858 returned string if *value* is not a whole number. Trailing zeros shall be suppressed where *value* is
8859 not a whole number. The radix character is determined by the current locale. If *setlocale()* has not
8860 been called successfully, the default locale, POSIX, is used. The default locale specifies a period
8861 ('.') as the radix character. The *LC_NUMERIC* category determines the value of the radix
8862 character within the current locale.8863 These functions need not be reentrant. A function that is not required to be reentrant is not
8864 required to be thread-safe.

8865 RETURN VALUE

8866 The *ecvt()* and *fcvt()* functions shall return a pointer to a null-terminated string of digits.8867 The *gcvt()* function shall return *buf*.8868 The return values from *ecvt()* and *fcvt()* may point to static data which may be overwritten by
8869 subsequent calls to these functions.

8870 **ERRORS**

8871 No errors are defined.

8872 **EXAMPLES**

8873 None.

8874 **APPLICATION USAGE**8875 The *sprintf()* function is preferred over this function.8876 **RATIONALE**

8877 None.

8878 **FUTURE DIRECTIONS**

8879 These functions may be withdrawn in a future version.

8880 **SEE ALSO**8881 *printf()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>8882 **CHANGE HISTORY**

8883 First released in Issue 4, Version 2.

8884 **Issue 5**

8885 Moved from X/OPEN UNIX extension to BASE.

8886 Normative text previously in the APPLICATION USAGE section is moved to the
8887 DESCRIPTION.

8888 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

8889 **Issue 6**

8890 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

8891 This function is marked LEGACY.

8892 The **restrict** keyword is added to the *ecvt()* and *fcvt()* prototypes for alignment with the
8893 ISO/IEC 9899:1999 standard.8894 The DESCRIPTION is updated to explicitly use “conversion specification” to describe %g, %f,
8895 and %e.

8896 **NAME**8897 encrypt — encoding function (**CRYPT**)8898 **SYNOPSIS**8899 XSI `#include <unistd.h>`8900 `void encrypt(char block[64], int edflag);`

8901

8902 **DESCRIPTION**8903 The *encrypt()* function shall provide access to an implementation-defined encoding algorithm.
8904 The key generated by *setkey()* is used to encrypt the string *block* with *encrypt()*.8905 The *block* argument to *encrypt()* shall be an array of length 64 bytes containing only the bytes
8906 with values of 0 and 1. The array is modified in place to a similar array using the key set by
8907 *setkey()*. If *edflag* is 0, the argument is encoded. If *edflag* is 1, the argument may be decoded (see
8908 the APPLICATION USAGE section); if the argument is not decoded, *errno* shall be set to
8909 [ENOSYS].8910 The *encrypt()* function shall not change the setting of *errno* if successful. An application wishing
8911 to check for error situations should set *errno* to 0 before calling *encrypt()*. If *errno* is non-zero on
8912 return, an error has occurred.8913 The *encrypt()* function need not be reentrant. A function that is not required to be reentrant is
8914 not required to be thread-safe.8915 **RETURN VALUE**8916 The *encrypt()* function shall not return a value.8917 **ERRORS**8918 The *encrypt()* function shall fail if:

8919 [ENOSYS] The functionality is not supported on this implementation.

8920 **EXAMPLES**

8921 None.

8922 **APPLICATION USAGE**8923 Historical implementations of the *encrypt()* function used a rather primitive encoding algorithm.8924 In some environments, decoding might not be implemented. This is related to some Government
8925 restrictions on encryption and decryption routines. Historical practice has been to ship a
8926 different version of the encryption library without the decryption feature in the routines
8927 supplied. Thus the exported version of *encrypt()* does encoding but not decoding.8928 **RATIONALE**

8929 None.

8930 **FUTURE DIRECTIONS**

8931 None.

8932 **SEE ALSO**8933 *crypt()*, *setkey()*, the Base Definitions volume of IEEE Std 1003.1-2001, <unistd.h>8934 **CHANGE HISTORY**

8935 First released in Issue 1. Derived from Issue 1 of the SVID.

8936 **Issue 5**

8937 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

8938 **Issue 6**

8939 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

8940 **NAME**

8941 endgrent, getgrent, setgrent — group database entry functions

8942 **SYNOPSIS**

```
8943 xSI #include <grp.h>
8944 void endgrent(void);
8945 struct group *getgrent(void);
8946 void setgrent(void);
8947
```

8948 **DESCRIPTION**

8949 The *getgrent()* function shall return a pointer to a structure containing the broken-out fields of an
 8950 entry in the group database. When first called, *getgrent()* shall return a pointer to a **group**
 8951 structure containing the first entry in the group database. Thereafter, it shall return a pointer to a
 8952 **group** structure containing the next group structure in the group database, so successive calls
 8953 may be used to search the entire database.

8954 An implementation that provides extended security controls may impose further
 8955 implementation-defined restrictions on accessing the group database. In particular, the system
 8956 may deny the existence of some or all of the group database entries associated with groups other
 8957 than those groups associated with the caller and may omit users other than the caller from the
 8958 list of members of groups in database entries that are returned.

8959 The *setgrent()* function shall rewind the group database to allow repeated searches.

8960 The *endgrent()* function may be called to close the group database when processing is complete.

8961 These functions need not be reentrant. A function that is not required to be reentrant is not
 8962 required to be thread-safe.

8963 **RETURN VALUE**

8964 When first called, *getgrent()* shall return a pointer to the first group structure in the group
 8965 database. Upon subsequent calls it shall return the next group structure in the group database.
 8966 The *getgrent()* function shall return a null pointer on end-of-file or an error and *errno* may be set
 8967 to indicate the error.

8968 The return value may point to a static area which is overwritten by a subsequent call to
 8969 *getgrgid()*, *getgrnam()*, or *getgrent()*.

8970 **ERRORS**

8971 The *getgrent()* function may fail if:

- | | | |
|------|----------|--|
| 8972 | [EINTR] | A signal was caught during the operation. |
| 8973 | [EIO] | An I/O error has occurred. |
| 8974 | [EMFILE] | {OPEN_MAX} file descriptors are currently open in the calling process. |
| 8975 | [ENFILE] | The maximum allowable number of files is currently open in the system. |

8976 **EXAMPLES**

8977 None.

8978 **APPLICATION USAGE**

8979 These functions are provided due to their historical usage. Applications should avoid
8980 dependencies on fields in the group database, whether the database is a single file, or where in
8981 the file system name space the database resides. Applications should use *getgrnam()* and
8982 *getgrgid()* whenever possible because it avoids these dependencies.

8983 **RATIONALE**

8984 None.

8985 **FUTURE DIRECTIONS**

8986 None.

8987 **SEE ALSO**

8988 *getgrgid()*, *getgrnam()*, *getlogin()*, *getpwent()*, the Base Definitions volume of
8989 IEEE Std 1003.1-2001, <grp.h>

8990 **CHANGE HISTORY**

8991 First released in Issue 4, Version 2.

8992 **Issue 5**

8993 Moved from X/OPEN UNIX extension to BASE.

8994 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
8995 VALUE section.

8996 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

8997 **Issue 6**

8998 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

8999 **NAME**

9000 endhostent, gethostent, sethostent — network host database functions

9001 **SYNOPSIS**

9002 #include <netdb.h>

9003 void endhostent(void);

9004 struct hostent *gethostent(void);

9005 void sethostent(int *stayopen*);

9006 **DESCRIPTION**

9007 These functions shall retrieve information about hosts. This information is considered to be
9008 stored in a database that can be accessed sequentially or randomly. The implementation of this
9009 database is unspecified.

9010 **Note:** In many cases this database is implemented by the Domain Name System, as documented in
9011 RFC 1034, RFC 1035, and RFC 1886.

9012 The *sethostent()* function shall open a connection to the database and set the next entry for
9013 retrieval to the first entry in the database. If the *stayopen* argument is non-zero, the connection
9014 shall not be closed by a call to *gethostent()*, *gethostbyname()*, or *gethostbyaddr()*, and the
9015 implementation may maintain an open file descriptor.

9016 The *gethostent()* function shall read the next entry in the database, opening and closing a
9017 connection to the database as necessary.

9018 Entries shall be returned in **hostent** structures. Refer to *gethostbyaddr()* for a definition of the
9019 **hostent** structure.

9020 The *endhostent()* function shall close the connection to the database, releasing any open file
9021 descriptor.

9022 These functions need not be reentrant. A function that is not required to be reentrant is not
9023 required to be thread-safe.

9024 **RETURN VALUE**

9025 Upon successful completion, the *gethostent()* function shall return a pointer to a **hostent**
9026 structure if the requested entry was found, and a null pointer if the end of the database was
9027 reached or the requested entry was not found.

9028 **ERRORS**

9029 No errors are defined for *endhostent()*, *gethostent()*, and *sethostent()*.

9030 **EXAMPLES**

9031 None.

9032 **APPLICATION USAGE**

9033 The *gethostent()* function may return pointers to static data, which may be overwritten by
9034 subsequent calls to any of these functions.

9035 **RATIONALE**

9036 None.

9037 **FUTURE DIRECTIONS**

9038 None.

9039 **SEE ALSO**

9040 *endservent()*, *gethostbyaddr()*, the Base Definitions volume of IEEE Std 1003.1-2001, <netdb.h>

9041 **CHANGE HISTORY**

9042 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9043 **NAME**

9044 endnetent, getnetbyaddr, getnetbyname, getnetent, setnetent — network database functions

9045 **SYNOPSIS**

9046 #include <netdb.h>

9047 void endnetent(void);

9048 struct netent *getnetbyaddr(uint32_t net, int type);

9049 struct netent *getnetbyname(const char *name);

9050 struct netent *getnetent(void);

9051 void setnetent(int stayopen);

9052 **DESCRIPTION**9053 These functions shall retrieve information about networks. This information is considered to be
9054 stored in a database that can be accessed sequentially or randomly. The implementation of this
9055 database is unspecified.9056 The *setnetent()* function shall open and rewind the database. If the *stayopen* argument is non-
9057 zero, the connection to the *net* database shall not be closed after each call to *getnetent()* (either
9058 directly, or indirectly through one of the other *getnet**(*)* functions), and the implementation may
9059 maintain an open file descriptor to the database.9060 The *getnetent()* function shall read the next entry of the database, opening and closing a
9061 connection to the database as necessary.9062 The *getnetbyaddr()* function shall search the database from the beginning, and find the first entry
9063 for which the address family specified by *type* matches the *n_addrtype* member and the network
9064 number *net* matches the *n_net* member, opening and closing a connection to the database as
9065 necessary. The *net* argument shall be the network number in host byte order.9066 The *getnetbyname()* function shall search the database from the beginning and find the first entry
9067 for which the network name specified by *name* matches the *n_name* member, opening and
9068 closing a connection to the database as necessary.9069 The *getnetbyaddr()*, *getnetbyname()*, and *getnetent()* functions shall each return a pointer to a
9070 **netent** structure, the members of which shall contain the fields of an entry in the network
9071 database.9072 The *endnetent()* function shall close the database, releasing any open file descriptor.9073 These functions need not be reentrant. A function that is not required to be reentrant is not
9074 required to be thread-safe.9075 **RETURN VALUE**9076 Upon successful completion, *getnetbyaddr()*, *getnetbyname()*, and *getnetent()* shall return a
9077 pointer to a **netent** structure if the requested entry was found, and a null pointer if the end of the
9078 database was reached or the requested entry was not found. Otherwise, a null pointer shall be
9079 returned.9080 **ERRORS**

9081 No errors are defined.

9082 **EXAMPLES**

9083 None.

9084 **APPLICATION USAGE**9085 The *getnetbyaddr()*, *getnetbyname()*, and *getnetent()* functions may return pointers to static data,
9086 which may be overwritten by subsequent calls to any of these functions.9087 **RATIONALE**

9088 None.

9089 **FUTURE DIRECTIONS**

9090 None.

9091 **SEE ALSO**9092 The Base Definitions volume of IEEE Std 1003.1-2001, <**netdb.h**>9093 **CHANGE HISTORY**

9094 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9095 **NAME**

9096 endprotoent, getprotobyname, getprotobynumber, getprotoent, setprotoent — network protocol
9097 database functions

9098 **SYNOPSIS**

```
9099 #include <netdb.h>

9100 void endprotoent(void);
9101 struct protoent *getprotobyname(const char *name);
9102 struct protoent *getprotobynumber(int proto);
9103 struct protoent *getprotoent(void);
9104 void setprotoent(int stayopen);
```

9105 **DESCRIPTION**

9106 These functions shall retrieve information about protocols. This information is considered to be
9107 stored in a database that can be accessed sequentially or randomly. The implementation of this
9108 database is unspecified.

9109 The *setprotoent()* function shall open a connection to the database, and set the next entry to the
9110 first entry. If the *stayopen* argument is non-zero, the connection to the network protocol database
9111 shall not be closed after each call to *getprotoent()* (either directly, or indirectly through one of the
9112 other *getproto**() functions), and the implementation may maintain an open file descriptor for
9113 the database.

9114 The *getprotobyname()* function shall search the database from the beginning and find the first
9115 entry for which the protocol name specified by *name* matches the *p_name* member, opening and
9116 closing a connection to the database as necessary.

9117 The *getprotobynumber()* function shall search the database from the beginning and find the first
9118 entry for which the protocol number specified by *proto* matches the *p_proto* member, opening
9119 and closing a connection to the database as necessary.

9120 The *getprotoent()* function shall read the next entry of the database, opening and closing a
9121 connection to the database as necessary.

9122 The *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* functions shall each return a pointer
9123 to a **protoent** structure, the members of which shall contain the fields of an entry in the network
9124 protocol database.

9125 The *endprotoent()* function shall close the connection to the database, releasing any open file
9126 descriptor.

9127 These functions need not be reentrant. A function that is not required to be reentrant is not
9128 required to be thread-safe.

9129 **RETURN VALUE**

9130 Upon successful completion, *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* return a
9131 pointer to a **protoent** structure if the requested entry was found, and a null pointer if the end of
9132 the database was reached or the requested entry was not found. Otherwise, a null pointer is
9133 returned.

9134 **ERRORS**

9135 No errors are defined.

9136 **EXAMPLES**

9137 None.

9138 **APPLICATION USAGE**9139 The *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* functions may return pointers to
9140 static data, which may be overwritten by subsequent calls to any of these functions.9141 **RATIONALE**

9142 None.

9143 **FUTURE DIRECTIONS**

9144 None.

9145 **SEE ALSO**9146 The Base Definitions volume of IEEE Std 1003.1-2001, <**netdb.h**>9147 **CHANGE HISTORY**

9148 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9149 **NAME**

9150 endpwent, getpwent, setpwent — user database functions

9151 **SYNOPSIS**

```
9152 xSI      #include <pwd.h>
9153          void endpwent(void);
9154          struct passwd *getpwent(void);
9155          void setpwent(void);
9156
```

9157 **DESCRIPTION**

9158 These functions shall retrieve information about users.

9159 The *getpwent()* function shall return a pointer to a structure containing the broken-out fields of
 9160 an entry in the user database. Each entry in the user database contains a **passwd** structure. When
 9161 first called, *getpwent()* shall return a pointer to a **passwd** structure containing the first entry in
 9162 the user database. Thereafter, it shall return a pointer to a **passwd** structure containing the next
 9163 entry in the user database. Successive calls can be used to search the entire user database.

9164 If an end-of-file or an error is encountered on reading, *getpwent()* shall return a null pointer.

9165 An implementation that provides extended security controls may impose further
 9166 implementation-defined restrictions on accessing the user database. In particular, the system
 9167 may deny the existence of some or all of the user database entries associated with users other
 9168 than the caller.

9169 The *setpwent()* function effectively rewinds the user database to allow repeated searches.9170 The *endpwent()* function may be called to close the user database when processing is complete.

9171 These functions need not be reentrant. A function that is not required to be reentrant is not
 9172 required to be thread-safe.

9173 **RETURN VALUE**9174 The *getpwent()* function shall return a null pointer on end-of-file or error.9175 **ERRORS**9176 The *getpwent()*, *setpwent()*, and *endpwent()* functions may fail if:

9177 [EIO] An I/O error has occurred.

9178 In addition, *getpwent()* and *setpwent()* may fail if:

9179 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

9180 [ENFILE] The maximum allowable number of files is currently open in the system.

9181 The return value may point to a static area which is overwritten by a subsequent call to
 9182 *getpwuid()*, *getpwnam()*, or *getpwent()*.

9183 **EXAMPLES**9184 **Searching the User Database**

9185 The following example uses the *getpwent()* function to get successive entries in the user
 9186 database, returning a pointer to a **passwd** structure that contains information about each user.
 9187 The call to *endpwent()* closes the user database and cleans up.

```
9188 #include <pwd.h>
9189 ...
9190 struct passwd *p;
9191 ...
9192 while ((p = getpwent ()) != NULL) {
9193     ...
9194 }
9195 endpwent ();
9196 ...
```

9197 **APPLICATION USAGE**

9198 These functions are provided due to their historical usage. Applications should avoid
 9199 dependencies on fields in the password database, whether the database is a single file, or where
 9200 in the file system name space the database resides. Applications should use *getpwuid()*
 9201 whenever possible because it avoids these dependencies.

9202 **RATIONALE**

9203 None.

9204 **FUTURE DIRECTIONS**

9205 None.

9206 **SEE ALSO**

9207 *endgrent()*, *getlogin()*, *getpwnam()*, *getpwuid()*, the Base Definitions volume of
 9208 IEEE Std 1003.1-2001, <pwd.h>

9209 **CHANGE HISTORY**

9210 First released in Issue 4, Version 2.

9211 **Issue 5**

9212 Moved from X/OPEN UNIX extension to BASE.

9213 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
 9214 VALUE section.

9215 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

9216 **Issue 6**

9217 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

9218 **NAME**

9219 endservent, getservbyname, getservbyport, getservent, setservent — network services database
9220 functions

9221 **SYNOPSIS**

```
9222 #include <netdb.h>

9223 void endservent(void);
9224 struct servent *getservbyname(const char *name, const char *proto);
9225 struct servent *getservbyport(int port, const char *proto);
9226 struct servent *getservent(void);
9227 void setservent(int stayopen);
```

9228 **DESCRIPTION**

9229 These functions shall retrieve information about network services. This information is
9230 considered to be stored in a database that can be accessed sequentially or randomly. The
9231 implementation of this database is unspecified.

9232 The *setservent()* function shall open a connection to the database, and set the next entry to the
9233 first entry. If the *stayopen* argument is non-zero, the *net* database shall not be closed after each
9234 call to the *getservent()* function (either directly, or indirectly through one of the other *getserv*()*
9235 functions), and the implementation may maintain an open file descriptor for the database.

9236 The *getservent()* function shall read the next entry of the database, opening and closing a
9237 connection to the database as necessary.

9238 The *getservbyname()* function shall search the database from the beginning and find the first
9239 entry for which the service name specified by *name* matches the *s_name* member and the protocol
9240 name specified by *proto* matches the *s_proto* member, opening and closing a connection to the
9241 database as necessary. If *proto* is a null pointer, any value of the *s_proto* member shall be
9242 matched.

9243 The *getservbyport()* function shall search the database from the beginning and find the first entry
9244 for which the port specified by *port* matches the *s_port* member and the protocol name specified
9245 by *proto* matches the *s_proto* member, opening and closing a connection to the database as
9246 necessary. If *proto* is a null pointer, any value of the *s_proto* member shall be matched. The *port*
9247 argument shall be in network byte order.

9248 The *getservbyname()*, *getservbyport()*, and *getservent()* functions shall each return a pointer to a
9249 **servent** structure, the members of which shall contain the fields of an entry in the network
9250 services database.

9251 The *endservent()* function shall close the database, releasing any open file descriptor.

9252 These functions need not be reentrant. A function that is not required to be reentrant is not
9253 required to be thread-safe.

9254 **RETURN VALUE**

9255 Upon successful completion, *getservbyname()*, *getservbyport()*, and *getservent()* return a pointer to
9256 a **servent** structure if the requested entry was found, and a null pointer if the end of the database
9257 was reached or the requested entry was not found. Otherwise, a null pointer is returned.

9258 **ERRORS**

9259 No errors are defined.

9260 **EXAMPLES**

9261 None.

9262 **APPLICATION USAGE**9263 The *port* argument of *getservbyport()* need not be compatible with the port values of all address
9264 families.9265 The *getservbyname()*, *getservbyport()*, and *getservent()* functions may return pointers to static
9266 data, which may be overwritten by subsequent calls to any of these functions.9267 **RATIONALE**

9268 None.

9269 **FUTURE DIRECTIONS**

9270 None.

9271 **SEE ALSO**9272 *endhostent()*, *endprotoent()*, *htonl()*, *inet_addr()*, the Base Definitions volume of
9273 IEEE Std 1003.1-2001, <**netdb.h**>9274 **CHANGE HISTORY**

9275 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9276 **NAME**

9277 endutxent, getutxent, getutxid, getutxline, pututxline, setutxent — user accounting database
 9278 functions

9279 **SYNOPSIS**

```
9280 xSI #include <utmpx.h>
9281 void endutxent(void);
9282 struct utmpx *getutxent(void);
9283 struct utmpx *getutxid(const struct utmpx *id);
9284 struct utmpx *getutxline(const struct utmpx *line);
9285 struct utmpx *pututxline(const struct utmpx *utmpx);
9286 void setutxent(void);
9287
```

9288 **DESCRIPTION**

9289 These functions shall provide access to the user accounting database.

9290 The *getutxent()* function shall read the next entry from the user accounting database. If the
 9291 database is not already open, it shall open it. If it reaches the end of the database, it shall fail.

9292 The *getutxid()* function shall search forward from the current point in the database. If the
 9293 *ut_type* value of the **utmpx** structure pointed to by *id* is `BOOT_TIME`, `OLD_TIME`, or
 9294 `NEW_TIME`, then it shall stop when it finds an entry with a matching *ut_type* value. If the
 9295 *ut_type* value is `INIT_PROCESS`, `LOGIN_PROCESS`, `USER_PROCESS`, or `DEAD_PROCESS`,
 9296 then it shall stop when it finds an entry whose type is one of these four and whose *ut_id* member
 9297 matches the *ut_id* member of the **utmpx** structure pointed to by *id*. If the end of the database is
 9298 reached without a match, *getutxid()* shall fail.

9299 The *getutxline()* function shall search forward from the current point in the database until it
 9300 finds an entry of the type `LOGIN_PROCESS` or `USER_PROCESS` which also has a *ut_line* value
 9301 matching that in the **utmpx** structure pointed to by *line*. If the end of the database is reached
 9302 without a match, *getutxline()* shall fail.

9303 The *getutxid()* or *getutxline()* function may cache data. For this reason, to use *getutxline()* to
 9304 search for multiple occurrences, the application shall zero out the static data after each success,
 9305 or *getutxline()* may return a pointer to the same **utmpx** structure.

9306 There is one exception to the rule about clearing the structure before further reads are done. The
 9307 implicit read done by *pututxline()* (if it finds that it is not already at the correct place in the user
 9308 accounting database) shall not modify the static structure returned by *getutxent()*, *getutxid()*, or
 9309 *getutxline()*, if the application has modified this structure and passed the pointer back to
 9310 *pututxline()*.

9311 For all entries that match a request, the *ut_type* member indicates the type of the entry. Other
 9312 members of the entry shall contain meaningful data based on the value of the *ut_type* member as
 9313 follows:

9314
9315
9316
9317
9318
9319
9320
9321
9322
9323
9324

ut_type Member	Other Members with Meaningful Data
EMPTY	No others
BOOT_TIME	<i>ut_tv</i>
OLD_TIME	<i>ut_tv</i>
NEW_TIME	<i>ut_tv</i>
USER_PROCESS	<i>ut_id</i> , <i>ut_user</i> (login name of the user), <i>ut_line</i> , <i>ut_pid</i> , <i>ut_tv</i>
INIT_PROCESS	<i>ut_id</i> , <i>ut_pid</i> , <i>ut_tv</i>
LOGIN_PROCESS	<i>ut_id</i> , <i>ut_user</i> (implementation-defined name of the login process), <i>ut_pid</i> , <i>ut_tv</i>
DEAD_PROCESS	<i>ut_id</i> , <i>ut_pid</i> , <i>ut_tv</i>

9325
9326
9327
9328

An implementation that provides extended security controls may impose implementation-defined restrictions on accessing the user accounting database. In particular, the system may deny the existence of some or all of the user accounting database entries associated with users other than the caller.

9329
9330
9331
9332

If the process has appropriate privileges, the *pututxline()* function shall write out the structure into the user accounting database. It shall use *getutxid()* to search for a record that satisfies the request. If this search succeeds, then the entry shall be replaced. Otherwise, a new entry shall be made at the end of the user accounting database.

9333

The *endutxent()* function shall close the user accounting database.

9334
9335

The *setutxent()* function shall reset the input to the beginning of the database. This should be done before each search for a new entry if it is desired that the entire database be examined.

9336
9337

These functions need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

9338 RETURN VALUE

9339
9340
9341

Upon successful completion, *getutxent()*, *getutxid()*, and *getutxline()* shall return a pointer to a **utmpx** structure containing a copy of the requested entry in the user accounting database. Otherwise, a null pointer shall be returned.

9342
9343

The return value may point to a static area which is overwritten by a subsequent call to *getutxid()* or *getutxline()*.

9344
9345
9346

Upon successful completion, *pututxline()* shall return a pointer to a **utmpx** structure containing a copy of the entry added to the user accounting database. Otherwise, a null pointer shall be returned.

9347

The *endutxent()* and *setutxent()* functions shall not return a value.

9348 ERRORS

9349
9350

No errors are defined for the *endutxent()*, *getutxent()*, *getutxid()*, *getutxline()*, and *setutxent()* functions.

9351

The *pututxline()* function may fail if:

9352

[EPERM] The process does not have appropriate privileges.

9353 **EXAMPLES**

9354 None.

9355 **APPLICATION USAGE**9356 The sizes of the arrays in the structure can be found using the *sizeof* operator.9357 **RATIONALE**

9358 None.

9359 **FUTURE DIRECTIONS**

9360 None.

9361 **SEE ALSO**

9362 The Base Definitions volume of IEEE Std 1003.1-2001, <utmpx.h>

9363 **CHANGE HISTORY**

9364 First released in Issue 4, Version 2.

9365 **Issue 5**

9366 Moved from X/OPEN UNIX extension to BASE.

9367 Normative text previously in the APPLICATION USAGE section is moved to the
9368 DESCRIPTION.

9369 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

9370 **Issue 6**

9371 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

9372 **NAME**

9373 **environ** — array of character pointers to the environment strings

9374 **SYNOPSIS**

9375 extern char **environ;

9376 **DESCRIPTION**

9377 Refer to the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables
9378 and *exec*.

9379 **NAME**9380 `erand48` — generate uniformly distributed pseudo-random numbers9381 **SYNOPSIS**9382 `XSI` `#include <stdlib.h>`9383 `double erand48(unsigned short xsubi[3]);`

9384

9385 **DESCRIPTION**9386 Refer to *drand48()*.

9387 **NAME**

9388 erf, erff, erfl — error functions

9389 **SYNOPSIS**

9390 #include <math.h>

9391 double erf(double x);

9392 float erff(float x);

9393 long double erfl(long double x);

9394 **DESCRIPTION**

9395 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 9396 conflict between the requirements described here and the ISO C standard is unintentional. This
 9397 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

9398 These functions shall compute the error function of their argument *x*, defined as:

$$9399 \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

9400 An application wishing to check for error situations should set *errno* to zero and call
 9401 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 9402 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 9403 zero, an error has occurred.

9404 **RETURN VALUE**

9405 Upon successful completion, these functions shall return the value of the error function.

9406 **MX** If *x* is NaN, a NaN shall be returned.9407 If *x* is ±0, ±0 shall be returned.9408 If *x* is ±Inf, ±1 shall be returned.9409 If *x* is subnormal, a range error may occur, and $2 * x / \text{sqrt}(\pi)$ should be returned.9410 **ERRORS**

9411 These functions may fail if:

9412 **MX** **Range Error** The result underflows.

9413 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 9414 then *errno* shall be set to [ERANGE]. If the integer expression
 9415 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 9416 floating-point exception shall be raised.

9417 **EXAMPLES**

9418 None.

9419 **APPLICATION USAGE**9420 Underflow occurs when $|x| < \text{DBL_MIN} * (\text{sqrt}(\pi)/2)$.

9421 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 9422 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

9423 **RATIONALE**

9424 None.

9425 **FUTURE DIRECTIONS**

9426 None.

9427 **SEE ALSO**9428 *erfc()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,
9429 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>9430 **CHANGE HISTORY**

9431 First released in Issue 1. Derived from Issue 1 of the SVID.

9432 **Issue 5**9433 The DESCRIPTION is updated to indicate how an application should check for an error. This
9434 text was previously published in the APPLICATION USAGE section.9435 **Issue 6**9436 The *erf()* function is no longer marked as an extension.9437 The *erfc()* function is split out onto its own reference page. |9438 The *erff()* and *erfl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.9439 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
9440 revised to align with the ISO/IEC 9899:1999 standard.9441 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
9442 marked.

9443 **NAME**

9444 erfc, erfcf, erfcl — complementary error functions

9445 **SYNOPSIS**

9446 #include <math.h>

9447 double erfc(double x);

9448 float erfcf(float x);

9449 long double erfcl(long double x);

9450 **DESCRIPTION**

9451 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 9452 conflict between the requirements described here and the ISO C standard is unintentional. This
 9453 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

9454 These functions shall compute the complementary error function $1.0 - \operatorname{erf}(x)$.

9455 An application wishing to check for error situations should set *errno* to zero and call
 9456 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 9457 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 9458 zero, an error has occurred.

9459 **RETURN VALUE**9460 Upon successful completion, these functions shall return the value of the complementary error
9461 function.

9462 If the correct value would cause underflow and is not representable, a range error may occur
 9463 **MX** and either 0.0 (if representable), or an implementation-defined value shall be returned.

9464 **MX** If *x* is NaN, a NaN shall be returned.9465 If *x* is ± 0 , +1 shall be returned.9466 If *x* is $-\operatorname{Inf}$, +2 shall be returned.9467 If *x* is $+\operatorname{Inf}$, +0 shall be returned.

9468 If the correct value would cause underflow and is representable, a range error may occur and the
 9469 correct value shall be returned.

9470 **ERRORS**

9471 These functions may fail if:

9472 Range Error The result underflows.

9473 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 9474 then *errno* shall be set to [ERANGE]. If the integer expression
 9475 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 9476 floating-point exception shall be raised.

9477 **EXAMPLES**

9478 None.

9479 **APPLICATION USAGE**9480 The *erfc()* function is provided because of the extreme loss of relative accuracy if *erf(x)* is called
9481 for large *x* and the result subtracted from 1.0.9482 Note for IEEE Std 754-1985 **double**, $26.55 < x$ implies *erfc(x)* has underflowed.

9483 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 9484 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

9485 **RATIONALE**

9486 None.

9487 **FUTURE DIRECTIONS**

9488 None.

9489 **SEE ALSO**9490 *erf()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,
9491 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>9492 **CHANGE HISTORY**

9493 First released in Issue 1. Derived from Issue 1 of the SVID.

9494 **Issue 5**9495 The DESCRIPTION is updated to indicate how an application should check for an error. This
9496 text was previously published in the APPLICATION USAGE section.9497 **Issue 6**9498 The *erfc()* function is no longer marked as an extension.9499 These functions are split out from the *erf()* reference page.9500 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
9501 revised to align with the ISO/IEC 9899:1999 standard.9502 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
9503 marked.

9504 **NAME**

9505 erff, erfl — error functions

9506 **SYNOPSIS**

9507 #include <math.h>

9508 float erff(float x);

9509 long double erfl(long double x);

9510 **DESCRIPTION**9511 Refer to *erf()*.

9512 **NAME**

9513 errno — error return value

9514 **SYNOPSIS**

9515 #include <errno.h>

9516 **DESCRIPTION**9517 The lvalue *errno* is used by many functions to return error values.

9518 Many functions provide an error number in *errno*, which has type **int** and is defined in
9519 <**errno.h**>. The value of *errno* shall be defined only after a call to a function for which it is
9520 explicitly stated to be set and until it is changed by the next function call or if the application
9521 assigns it a value. The value of *errno* should only be examined when it is indicated to be valid by
9522 a function's return value. Applications shall obtain the definition of *errno* by the inclusion of
9523 <**errno.h**>. No function in this volume of IEEE Std 1003.1-2001 shall set *errno* to 0.

9524 It is unspecified whether *errno* is a macro or an identifier declared with external linkage. If a
9525 macro definition is suppressed in order to access an actual object, or a program defines an
9526 identifier with the name *errno*, the behavior is undefined.

9527 The symbolic values stored in *errno* are documented in the ERRORS sections on all relevant
9528 pages.

9529 **RETURN VALUE**

9530 None.

9531 **ERRORS**

9532 None.

9533 **EXAMPLES**

9534 None.

9535 **APPLICATION USAGE**

9536 Previously both POSIX and X/Open documents were more restrictive than the ISO C standard
9537 in that they required *errno* to be defined as an external variable, whereas the ISO C standard
9538 required only that *errno* be defined as a modifiable lvalue with type **int**.

9539 An application that needs to examine the value of *errno* to determine the error should set it to 0
9540 before a function call, then inspect it before a subsequent function call.

9541 **RATIONALE**

9542 None.

9543 **FUTURE DIRECTIONS**

9544 None.

9545 **SEE ALSO**9546 Section 2.3, the Base Definitions volume of IEEE Std 1003.1-2001, <**errno.h**>9547 **CHANGE HISTORY**

9548 First released in Issue 1. Derived from Issue 1 of the SVID.

9549 **Issue 5**

9550 The following sentence is deleted from the DESCRIPTION: "The value of *errno* is 0 at program
9551 start-up, but is never set to 0 by any XSI function". The DESCRIPTION also no longer states that
9552 conforming implementations may support the declaration:

9553 extern int errno;

9554 **Issue 6**

9555 Obsolescent text regarding defining *errno* as:

9556 `extern int errno`

9557 is removed.

9558 Text regarding no function setting *errno* to zero to indicate an error is changed to no function
9559 shall set *errno* to zero. This is for alignment with the ISO/IEC 9899:1999 standard.

9560 NAME

9561 environ, execl, execv, execl, execve, execlp, execvp — execute a file

9562 SYNOPSIS

```
9563     #include <unistd.h>

9564     extern char **environ;
9565     int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
9566     int execv(const char *path, char *const argv[]);
9567     int execl(const char *path, const char *arg0, ... /*,
9568             (char *)0, char *const envp[] */);
9569     int execve(const char *path, char *const argv[], char *const envp[]);
9570     int execlp(const char *file, const char *arg0, ... /*, (char *)0 */);
9571     int execvp(const char *file, char *const argv[]);
```

9572 DESCRIPTION

9573 The *exec* family of functions shall replace the current process image with a new process image.
 9574 The new image shall be constructed from a regular, executable file called the *new process image*
 9575 *file*. There shall be no return from a successful *exec*, because the calling process image is overlaid
 9576 by the new process image.

9577 When a C-language program is executed as a result of this call, it shall be entered as a C-
 9578 language function call as follows:

```
9579     int main (int argc, char *argv[]);
```

9580 where *argc* is the argument count and *argv* is an array of character pointers to the arguments
 9581 themselves. In addition, the following variable:

```
9582     extern char **environ;
```

9583 is initialized as a pointer to an array of character pointers to the environment strings. The *argv*
 9584 and *environ* arrays are each terminated by a null pointer. The null pointer terminating the *argv*
 9585 array is not counted in *argc*.

9586 THR Conforming multi-threaded applications shall not use the *environ* variable to access or modify
 9587 any environment variable while any other thread is concurrently modifying any environment
 9588 variable. A call to any function dependent on any environment variable shall be considered a use
 9589 of the *environ* variable to access that environment variable.

9590 The arguments specified by a program with one of the *exec* functions shall be passed on to the
 9591 new process image in the corresponding *main()* arguments.

9592 The argument *path* points to a pathname that identifies the new process image file.

9593 The argument *file* is used to construct a pathname that identifies the new process image file. If
 9594 the *file* argument contains a slash character, the *file* argument shall be used as the pathname for
 9595 this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed
 9596 as the environment variable *PATH* (see the Base Definitions volume of IEEE Std 1003.1-2001,
 9597 Chapter 8, Environment Variables). If this environment variable is not present, the results of the
 9598 search are implementation-defined.

9599 There are two distinct ways in which the contents of the process image file may cause the
 9600 execution to fail, distinguished by the setting of *errno* to either [ENOEXEC] or [EINVAL] (see the
 9601 ERRORS section). In the cases where the other members of the *exec* family of functions would
 9602 fail and set *errno* to [ENOEXEC], the *execlp()* and *execvp()* functions shall execute a command
 9603 interpreter and the environment of the executed command shall be as if the process invoked the
 9604 *sh* utility using *execl()* as follows:

9605 `execl(<shell path>, arg0, file, arg1, ..., (char *)0);`

9606 where *<shell path>* is an unspecified pathname for the *sh* utility, *file* is the process image file, and
 9607 for *execvp()*, where *arg0*, *arg1*, and so on correspond to the values passed to *execvp()* in *argv[0]*,
 9608 *argv[1]*, and so on.

9609 The arguments represented by *arg0*,... are pointers to null-terminated character strings. These
 9610 strings shall constitute the argument list available to the new process image. The list is
 9611 terminated by a null pointer. The argument *arg0* should point to a filename that is associated
 9612 with the process being started by one of the *exec* functions.

9613 The argument *argv* is an array of character pointers to null-terminated strings. The application
 9614 shall ensure that the last member of this array is a null pointer. These strings shall constitute the
 9615 argument list available to the new process image. The value in *argv[0]* should point to a filename
 9616 that is associated with the process being started by one of the *exec* functions.

9617 The argument *envp* is an array of character pointers to null-terminated strings. These strings
 9618 shall constitute the environment for the new process image. The *envp* array is terminated by a
 9619 null pointer.

9620 For those forms not containing an *envp* pointer (*execl()*, *execv()*, *execlp()*, and *execvp()*), the
 9621 environment for the new process image shall be taken from the external variable *environ* in the
 9622 calling process.

9623 The number of bytes available for the new process' combined argument and environment lists is
 9624 {ARG_MAX}. It is implementation-defined whether null terminators, pointers, and/or any
 9625 alignment bytes are included in this total.

9626 File descriptors open in the calling process image shall remain open in the new process image,
 9627 except for those whose close-on-exec flag FD_CLOEXEC is set. For those file descriptors that
 9628 remain open, all attributes of the open file description remain unchanged. For any file descriptor
 9629 that is closed for this reason, file locks are removed as a result of the close as described in *close()*.
 9630 Locks that are not removed by closing of file descriptors remain unchanged.

9631 If file descriptors 0, 1, and 2 would otherwise be closed after a successful call to one of the *exec* |
 9632 family of functions, and the new process image file has the set-user-ID or set-group-ID file mode |
 9633 XSI bits set, and the ST_NOSUID bit is not set for the file system containing the new process image |
 9634 file, implementations may open an unspecified file for each of these file descriptors in the new |
 9635 process image. |

9636 Directory streams open in the calling process image shall be closed in the new process image.

9637 The state of the floating-point environment in the new process image shall be set to the default.

9638 XSI The state of conversion descriptors and message catalog descriptors in the new process image is
 9639 undefined. For the new process image, the equivalent of:

9640 `setlocale(LC_ALL, "C")`

9641 shall be executed at start-up.

9642 Signals set to the default action (SIG_DFL) in the calling process image shall be set to the default
 9643 action in the new process image. Except for SIGCHLD, signals set to be ignored (SIG_IGN) by
 9644 the calling process image shall be set to be ignored by the new process image. Signals set to be
 9645 caught by the calling process image shall be set to the default action in the new process image
 9646 (see *<signal.h>*). If the SIGCHLD signal is set to be ignored by the calling process image, it is
 9647 unspecified whether the SIGCHLD signal is set to be ignored or to the default action in the new
 9648 XSI process image. After a successful call to any of the *exec* functions, alternate signal stacks are not
 9649 preserved and the SA_ONSTACK flag shall be cleared for all signals.

9650		After a successful call to any of the <i>exec</i> functions, any functions previously registered by <i>atexit()</i>
9651		are no longer registered.
9652	XSI	If the <i>ST_NOSUID</i> bit is set for the file system containing the new process image file, then the effective user ID, effective group ID, saved set-user-ID, and saved set-group-ID are unchanged
9653		in the new process image. Otherwise, if the set-user-ID mode bit of the new process image file is
9654		set, the effective user ID of the new process image shall be set to the user ID of the new process
9655		image file. Similarly, if the set-group-ID mode bit of the new process image file is set, the
9656		effective group ID of the new process image shall be set to the group ID of the new process
9657		image file. The real user ID, real group ID, and supplementary group IDs of the new process
9658		image shall remain the same as those of the calling process image. The effective user ID and
9659		effective group ID of the new process image shall be saved (as the saved set-user-ID and the
9660		saved set-group-ID) for use by <i>setuid()</i> .
9661		
9662	XSI	Any shared memory segments attached to the calling process image shall not be attached to the
9663		new process image.
9664	SEM	Any named semaphores open in the calling process shall be closed as if by appropriate calls to
9665		<i>sem_close()</i> .
9666	TYM	Any blocks of typed memory that were mapped in the calling process are unmapped, as if
9667		<i>munmap()</i> was implicitly called to unmap them.
9668	ML	Memory locks established by the calling process via calls to <i>mlockall()</i> or <i>mlock()</i> shall be
9669		removed. If locked pages in the address space of the calling process are also mapped into the
9670		address spaces of other processes and are locked by those processes, the locks established by the
9671		other processes shall be unaffected by the call by this process to the <i>exec</i> function. If the <i>exec</i>
9672		function fails, the effect on memory locks is unspecified.
9673	MF SHM	Memory mappings created in the process are unmapped before the address space is rebuilt for
9674		the new process image.
9675	PS	For the <i>SCHED_FIFO</i> and <i>SCHED_RR</i> scheduling policies, the policy and priority settings shall
9676		not be changed by a call to an <i>exec</i> function. For other scheduling policies, the policy and priority
9677		settings on <i>exec</i> are implementation-defined.
9678	TMR	Per-process timers created by the calling process shall be deleted before replacing the current
9679		process image with the new process image.
9680	MSG	All open message queue descriptors in the calling process shall be closed, as described in
9681		<i>mq_close()</i> .
9682	AIO	Any outstanding asynchronous I/O operations may be canceled. Those asynchronous I/O
9683		operations that are not canceled shall complete as if the <i>exec</i> function had not yet occurred, but
9684		any associated signal notifications shall be suppressed. It is unspecified whether the <i>exec</i>
9685		function itself blocks awaiting such I/O completion. In no event, however, shall the new process
9686		image created by the <i>exec</i> function be affected by the presence of outstanding asynchronous I/O
9687		operations at the time the <i>exec</i> function is called. Whether any I/O is canceled, and which I/O
9688		may be canceled upon <i>exec</i> , is implementation-defined.
9689	CPT	The new process image shall inherit the CPU-time clock of the calling process image. This
9690		inheritance means that the process CPU-time clock of the process being <i>exec</i> -ed shall not be
9691		reinitialized or altered as a result of the <i>exec</i> function other than to reflect the time spent by the
9692		process executing the <i>exec</i> function itself.
9693	TCT	The initial value of the CPU-time clock of the initial thread of the new process image shall be set
9694		to zero.

- 9695 TRC If the calling process is being traced, the new process image shall continue to be traced into the
 9696 same trace stream as the original process image, but the new process image shall not inherit the
 9697 mapping of trace event names to trace event type identifiers that was defined by calls to the
 9698 *posix_trace_eventid_open()* or the *posix_trace_trid_eventid_open()* functions in the calling process
 9699 image.
- 9700 If the calling process is a trace controller process, any trace streams that were created by the
 9701 calling process shall be shut down as described in the *posix_trace_shutdown()* function.
- 9702 The new process shall inherit at least the following attributes from the calling process image:
- 9703 XSI • Nice value (see *nice()*)
 - 9704 XSI • *semadj* values (see *semop()*)
 - 9705 • Process ID
 - 9706 • Parent process ID
 - 9707 • Process group ID
 - 9708 • Session membership
 - 9709 • Real user ID
 - 9710 • Real group ID
 - 9711 • Supplementary group IDs
 - 9712 • Time left until an alarm clock signal (see *alarm()*)
 - 9713 • Current working directory
 - 9714 • Root directory
 - 9715 • File mode creation mask (see *umask()*)
 - 9716 XSI • File size limit (see *ulimit()*)
 - 9717 • Process signal mask (see *sigprocmask()*)
 - 9718 • Pending signal (see *sigpending()*)
 - 9719 • *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime* (see *times()*)
 - 9720 XSI • Resource limits
 - 9721 XSI • Controlling terminal
 - 9722 XSI • Interval timers
- 9723 All other process attributes defined in this volume of IEEE Std 1003.1-2001 shall be the same in
 9724 the new and old process images. The inheritance of process attributes not defined by this
 9725 volume of IEEE Std 1003.1-2001 is implementation-defined.
- 9726 A call to any *exec* function from a process with more than one thread shall result in all threads
 9727 being terminated and the new executable image being loaded and executed. No destructor
 9728 functions shall be called.
- 9729 Upon successful completion, the *exec* functions shall mark for update the *st_atime* field of the file.
 9730 If an *exec* function failed but was able to locate the process image file, whether the *st_atime* field
 9731 is marked for update is unspecified. Should the *exec* function succeed, the process image file
 9732 shall be considered to have been opened with *open()*. The corresponding *close()* shall be
 9733 considered to occur at a time after this open, but before process termination or successful
 9734 completion of a subsequent call to one of the *exec* functions, *posix_spawn()*, or *posix_spawnp()*.

9735 The *argv[]* and *envp[]* arrays of pointers and the strings to which those arrays point shall not be
 9736 modified by a call to one of the *exec* functions, except as a consequence of replacing the process
 9737 image.

9738 XSI The saved resource limits in the new process image are set to be a copy of the process'
 9739 corresponding hard and soft limits.

9740 RETURN VALUE

9741 If one of the *exec* functions returns to the calling process image, an error has occurred; the return
 9742 value shall be -1 , and *errno* shall be set to indicate the error.

9743 ERRORS

9744 The *exec* functions shall fail if:

9745 [E2BIG] The number of bytes used by the new process image's argument list and
 9746 environment list is greater than the system-imposed limit of {ARG_MAX}
 9747 bytes.

9748 [EACCES] Search permission is denied for a directory listed in the new process image
 9749 file's path prefix, or the new process image file denies execution permission,
 9750 or the new process image file is not a regular file and the implementation does
 9751 not support execution of files of its type.

9752 [EINVAL] The new process image file has the appropriate permission and has a
 9753 recognized executable binary format, but the system does not support
 9754 execution of a file with this format.

9755 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path* or *file*
 9756 argument.

9757 [ENAMETOOLONG] The length of the *path* or *file* arguments exceeds {PATH_MAX} or a pathname
 9758 component is longer than {NAME_MAX}.

9760 [ENOENT] A component of *path* or *file* does not name an existing file or *path* or *file* is an
 9761 empty string.

9762 [ENOTDIR] A component of the new process image file's path prefix is not a directory.

9763 The *exec* functions, except for *execlp()* and *execvp()*, shall fail if:

9764 [ENOEXEC] The new process image file has the appropriate access permission but has an
 9765 unrecognized format.

9766 The *exec* functions may fail if:

9767 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 9768 resolution of the *path* or *file* argument.

9769 [ENAMETOOLONG] As a result of encountering a symbolic link in resolution of the *path* argument,
 9770 the length of the substituted pathname string exceeded {PATH_MAX}.

9772 [ENOMEM] The new process image requires more memory than is allowed by the
 9773 hardware or system-imposed memory management constraints.

9774 [ETXTBSY] The new process image file is a pure procedure (shared text) file that is
 9775 currently open for writing by some process.

9776 **EXAMPLES**9777 **Using execl()**

9778 The following example executes the *ls* command, specifying the pathname of the executable
9779 (**/bin/ls**) and using arguments supplied directly to the command to produce single-column
9780 output.

```
9781 #include <unistd.h>
9782 int ret;
9783 ...
9784 ret = execl ("/bin/ls", "ls", "-l", (char *)0);
```

9785 **Using execl_e()**

9786 The following example is similar to **Using execl()**. In addition, it specifies the environment for
9787 the new process image using the *env* argument.

```
9788 #include <unistd.h>
9789 int ret;
9790 char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
9791 ...
9792 ret = execl_e ("/bin/ls", "ls", "-l", (char *)0, env);
```

9793 **Using execl_p()**

9794 The following example searches for the location of the *ls* command among the directories
9795 specified by the *PATH* environment variable.

```
9796 #include <unistd.h>
9797 int ret;
9798 ...
9799 ret = execl_p ("ls", "ls", "-l", (char *)0);
```

9800 **Using execv()**

9801 The following example passes arguments to the *ls* command in the *cmd* array.

```
9802 #include <unistd.h>
9803 int ret;
9804 char *cmd[] = { "ls", "-l", (char *)0 };
9805 ...
9806 ret = execv ("/bin/ls", cmd);
```

9807 **Using `execve()`**

9808 The following example passes arguments to the `ls` command in the `cmd` array, and specifies the
 9809 environment for the new process image using the `env` argument.

```
9810 #include <unistd.h>
9811 int ret;
9812 char *cmd[] = { "ls", "-l", (char *)0 };
9813 char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
9814 ...
9815 ret = execve ("/bin/ls", cmd, env);
```

9816 **Using `execvp()`**

9817 The following example searches for the location of the `ls` command among the directories
 9818 specified by the `PATH` environment variable, and passes arguments to the `ls` command in the
 9819 `cmd` array.

```
9820 #include <unistd.h>
9821 int ret;
9822 char *cmd[] = { "ls", "-l", (char *)0 };
9823 ...
9824 ret = execvp ("ls", cmd);
```

9825 **APPLICATION USAGE**

9826 As the state of conversion descriptors and message catalog descriptors in the new process image
 9827 is undefined, conforming applications should not rely on their use and should close them prior
 9828 to calling one of the `exec` functions.

9829 Applications that require other than the default POSIX locale should call `setlocale()` with the
 9830 appropriate parameters to establish the locale of the new process.

9831 The `environ` array should not be accessed directly by the application. |

9832 Applications should not depend on file descriptors 0, 1, and 2 being closed after an `exec`. A |
 9833 future version may allow these file descriptors to be automatically opened for any process. |

9834 **RATIONALE**

9835 Early proposals required that the value of `argc` passed to `main()` be “one or greater”. This was
 9836 driven by the same requirement in drafts of the ISO C standard. In fact, historical
 9837 implementations have passed a value of zero when no arguments are supplied to the caller of
 9838 the `exec` functions. This requirement was removed from the ISO C standard and subsequently
 9839 removed from this volume of IEEE Std 1003.1-2001 as well. The wording, in particular the use of
 9840 the word *should*, requires a Strictly Conforming POSIX Application to pass at least one argument
 9841 to the `exec` function, thus guaranteeing that `argc` be one or greater when invoked by such an
 9842 application. In fact, this is good practice, since many existing applications reference `argv[0]`
 9843 without first checking the value of `argc`.

9844 The requirement on a Strictly Conforming POSIX Application also states that the value passed
 9845 as the first argument be a filename associated with the process being started. Although some
 9846 existing applications pass a pathname rather than a filename in some circumstances, a filename
 9847 is more generally useful, since the common usage of `argv[0]` is in printing diagnostics. In some
 9848 cases the filename passed is not the actual filename of the file; for example, many
 9849 implementations of the `login` utility use a convention of prefixing a hyphen (‘-’) to the actual
 9850 filename, which indicates to the command interpreter being invoked that it is a “login shell”.

9851 Historically there have been two ways that implementations can *exec* shell scripts.

9852 One common historical implementation is that the *execl()*, *execv()*, *execle()*, and *execve()*
9853 functions return an [ENOEXEC] error for any file not recognizable as executable, including a
9854 shell script. When the *execlp()* and *execvp()* functions encounter such a file, they assume the file
9855 to be a shell script and invoke a known command interpreter to interpret such files. This is now
9856 required by IEEE Std 1003.1-2001. These implementations of *execvp()* and *execlp()* only give the
9857 [ENOEXEC] error in the rare case of a problem with the command interpreter's executable file.
9858 Because of these implementations, the [ENOEXEC] error is not mentioned for *execlp()* or
9859 *execvp()*, although implementations can still give it.

9860 Another way that some historical implementations handle shell scripts is by recognizing the first
9861 two bytes of the file as the character string "#!" and using the remainder of the first line of the
9862 file as the name of the command interpreter to execute.

9863 One potential source of confusion noted by the standard developers is over how the contents of
9864 a process image file affect the behavior of the *exec* family of functions. The following is a
9865 description of the actions taken:

- 9866 1. If the process image file is a valid executable (in a format that is executable and valid and
9867 having appropriate permission) for this system, then the system executes the file.
- 9868 2. If the process image file has appropriate permission and is in a format that is executable
9869 but not valid for this system (such as a recognized binary for another architecture), then
9870 this is an error and *errno* is set to [EINVAL] (see later RATIONALE on [EINVAL]).
- 9871 3. If the process image file has appropriate permission but is not otherwise recognized:
 - 9872 a. If this is a call to *execlp()* or *execvp()*, then they invoke a command interpreter
9873 assuming that the process image file is a shell script.
 - 9874 b. If this is not a call to *execlp()* or *execvp()*, then an error occurs and *errno* is set to
9875 [ENOEXEC].

9876 Applications that do not require to access their arguments may use the form:

```
9877 main(void)
```

9878 as specified in the ISO C standard. However, the implementation will always provide the two
9879 arguments *argc* and *argv*, even if they are not used.

9880 Some implementations provide a third argument to *main()* called *envp*. This is defined as a
9881 pointer to the environment. The ISO C standard specifies invoking *main()* with two arguments,
9882 so implementations must support applications written this way. Since this volume of
9883 IEEE Std 1003.1-2001 defines the global variable *environ*, which is also provided by historical
9884 implementations and can be used anywhere that *envp* could be used, there is no functional need
9885 for the *envp* argument. Applications should use the *getenv()* function rather than accessing the
9886 environment directly via either *envp* or *environ*. Implementations are required to support the
9887 two-argument calling sequence, but this does not prohibit an implementation from supporting
9888 *envp* as an optional third argument.

9889 This volume of IEEE Std 1003.1-2001 specifies that signals set to SIG_IGN remain set to
9890 SIG_IGN, and that the process signal mask be unchanged across an *exec*. This is consistent with
9891 historical implementations, and it permits some useful functionality, such as the *nohup*
9892 command. However, it should be noted that many existing applications wrongly assume that
9893 they start with certain signals set to the default action and/or unblocked. In particular,
9894 applications written with a simpler signal model that does not include blocking of signals, such
9895 as the one in the ISO C standard, may not behave properly if invoked with some signals blocked.
9896 Therefore, it is best not to block or ignore signals across *execs* without explicit reason to do so,

9897 and especially not to block signals across *execs* of arbitrary (not closely co-operating) programs.

9898 The *exec* functions always save the value of the effective user ID and effective group ID of the
9899 process at the completion of the *exec*, whether or not the set-user-ID or the set-group-ID bit of
9900 the process image file is set.

9901 The statement about *argv[]* and *envp[]* being constants is included to make explicit to future
9902 writers of language bindings that these objects are completely constant. Due to a limitation of
9903 the ISO C standard, it is not possible to state that idea in standard C. Specifying two levels of
9904 *const-qualification* for the *argv[]* and *envp[]* parameters for the *exec* functions may seem to be the
9905 natural choice, given that these functions do not modify either the array of pointers or the
9906 characters to which the function points, but this would disallow existing correct code. Instead,
9907 only the array of pointers is noted as constant. The table of assignment compatibility for *dst=src*
9908 derived from the ISO C standard summarizes the compatibility:

9909	<i>dst:</i>	char *[]	const char *[]	char *const[]	const char *const[]
9910	<i>src:</i>				
9911	char *[]	VALID	—	VALID	—
9912	const char *[]	—	VALID	—	VALID
9913	char * const []	—	—	VALID	—
9914	const char *const[]	—	—	—	VALID

9915 Since all existing code has a source type matching the first row, the column that gives the most
9916 valid combinations is the third column. The only other possibility is the fourth column, but
9917 using it would require a cast on the *argv* or *envp* arguments. It is unfortunate that the fourth
9918 column cannot be used, because the declaration a non-expert would naturally use would be that
9919 in the second row.

9920 The ISO C standard and this volume of IEEE Std 1003.1-2001 do not conflict on the use of
9921 *environ*, but some historical implementations of *environ* may cause a conflict. As long as *environ*
9922 is treated in the same way as an entry point (for example, *fork()*), it conforms to both standards.
9923 A library can contain *fork()*, but if there is a user-provided *fork()*, that *fork()* is given precedence
9924 and no problem ensues. The situation is similar for *environ*: the definition in this volume of
9925 IEEE Std 1003.1-2001 is to be used if there is no user-provided *environ* to take precedence. At
9926 least three implementations are known to exist that solve this problem.

9927 [E2BIG] The limit {ARG_MAX} applies not just to the size of the argument list, but to
9928 the sum of that and the size of the environment list.

9929 [EFAULT] Some historical systems return [EFAULT] rather than [ENOEXEC] when the
9930 new process image file is corrupted. They are non-conforming.

9931 [EINVAL] This error condition was added to IEEE Std 1003.1-2001 to allow an
9932 implementation to detect executable files generated for different architectures,
9933 and indicate this situation to the application. Historical implementations of
9934 shells, *execvp()*, and *execlp()* that encounter an [ENOEXEC] error will execute
9935 a shell on the assumption that the file is a shell script. This will not produce
9936 the desired effect when the file is a valid executable for a different
9937 architecture. An implementation may now choose to avoid this problem by
9938 returning [EINVAL] when a valid executable for a different architecture is
9939 encountered. Some historical implementations return [EINVAL] to indicate
9940 that the *path* argument contains a character with the high order bit set. The
9941 standard developers chose to deviate from historical practice for the following
9942 reasons:

- 9943 1. The new utilization of [EINVAL] will provide some measure of utility to
9944 the user community.
- 9945 2. Historical use of [EINVAL] is not acceptable in an internationalized
9946 operating environment.

9947 [ENAMETOOLONG]

9948 Since the file pathname may be constructed by taking elements in the *PATH*
9949 variable and putting them together with the filename, the
9950 [ENAMETOOLONG] error condition could also be reached this way.

9951 [ETXTBSY]

9952 System V returns this error when the executable file is currently open for
9953 writing by some process. This volume of IEEE Std 1003.1-2001 neither requires
nor prohibits this behavior.

9954 Other systems (such as System V) may return [EINTR] from *exec*. This is not addressed by this
9955 volume of IEEE Std 1003.1-2001, but implementations may have a window between the call to
9956 *exec* and the time that a signal could cause one of the *exec* calls to return with [EINTR].

9957 An explicit statement regarding the floating-point environment (as defined in the *<fenv.h>*
9958 header) was added to make it clear that the floating-point environment is set to its default when
9959 a call to one of the *exec* functions succeeds. The requirements for inheritance or setting to the
9960 default for other process and thread start-up functions is covered by more generic statements in
9961 their descriptions and can be summarized as follows:

9962 *posix_spawn()* Set to default.

9963 *fork()* Inherit.

9964 *pthread_create()* Inherit.

9965 FUTURE DIRECTIONS

9966 None.

9967 SEE ALSO

9968 *alarm()*, *atexit()*, *chmod()*, *close()*, *exit()*, *fcntl()*, *fork()*, *fstatvfs()*, *getenv()*, *getitimer()*, *getrlimit()*,
9969 *mmap()*, *nice()*, *posix_spawn()*, *posix_trace_eventid_open()*, *posix_trace_shutdown()*,
9970 *posix_trace_trid_eventid_open()*, *putenv()*, *semop()*, *setlocale()*, *shmat()*, *sigaction()*, *sigaltstack()*,
9971 *sigpending()*, *sigprocmask()*, *system()*, *times()*, *ulimit()*, *umask()*, the Base Definitions volume of
9972 IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, *<unistd.h>*

9973 CHANGE HISTORY

9974 First released in Issue 1. Derived from Issue 1 of the SVID.

9975 Issue 5

9976 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
9977 Threads Extension.

9978 Large File Summit extensions are added.

9979 Issue 6

9980 The following new requirements on POSIX implementations derive from alignment with the
9981 Single UNIX Specification:

- 9982 • In the DESCRIPTION, behavior is defined for when the process image file is not a valid
9983 executable.
- 9984 • In this issue, `_POSIX_SAVED_IDS` is mandated, thus the effective user ID and effective group
9985 ID of the new process image shall be saved (as the saved set-user-ID and the saved set-
9986 group-ID) for use by the *setuid()* function.

- 9987 • The [ELOOP] mandatory error condition is added.
- 9988 • A second [ENAMETOOLONG] is added as an optional error condition.
- 9989 • The [ETXTBSY] optional error condition is added.
- 9990 The following changes were made to align with the IEEE P1003.1a draft standard:
- 9991 • The [EINVAL] mandatory error condition is added.
- 9992 • The [ELOOP] optional error condition is added.
- 9993 The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.
- 9994 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for
9995 typed memory.
- 9996 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.
- 9997 The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.
- 9998 IEEE PASC Interpretation 1003.1 #132 is applied.
- 9999 The DESCRIPTION is updated to make it explicit that the floating-point environment in the new
10000 process image is set to the default.
- 10001 The DESCRIPTION and RATIONALE are updated to include clarifications of how the contents
10002 of a process image file affect the behavior of the *exec* functions.
- 10003 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/15 is applied, adding a new paragraph to
10004 the DESCRIPTION and text to the end of the APPLICATION USAGE section. This change
10005 addresses a security concern, where implementations may want to reopen file descriptors 0, 1,
10006 and 2 for programs with the set-user-id or set-group-id file mode bits calling the *exec* family of
10007 functions.

10008 NAME

10009 exit, _Exit, _exit — terminate a process

10010 SYNOPSIS

10011 #include <stdlib.h>

10012 void exit(int status);

10013 void _Exit(int status);

10014 #include <unistd.h>

10015 void _exit(int status);

10016 DESCRIPTION

10017 CX For *exit()* and *_Exit()*: The functionality described on this reference page is aligned with the
 10018 ISO C standard. Any conflict between the requirements described here and the ISO C standard is
 10019 unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

10020 CX The value of *status* may be 0, EXIT_SUCCESS, EXIT_FAILURE, or any other value, though only
 10021 the least significant 8 bits (that is, *status* & 0377) shall be available to a waiting parent process.

10022 The *exit()* function shall first call all functions registered by *atexit()*, in the reverse order of their
 10023 registration, except that a function is called after any previously registered functions that had
 10024 already been called at the time it was registered. Each function is called as many times as it was
 10025 registered. If, during the call to any such function, a call to the *longjmp()* function is made that
 10026 would terminate the call to the registered function, the behavior is undefined.

10027 If a function registered by a call to *atexit()* fails to return, the remaining registered functions shall
 10028 not be called and the rest of the *exit()* processing shall not be completed. If *exit()* is called more
 10029 than once, the behavior is undefined.

10030 The *exit()* function shall then flush all open streams with unwritten buffered data, close all open
 10031 streams, and remove all files created by *tmpfile()*. Finally, control shall be terminated with the
 10032 consequences described below.

10033 CX The *_Exit()* and *_exit()* functions shall be functionally equivalent.

10034 CX The *_Exit()* and *_exit()* functions shall not call functions registered with *atexit()* nor any
 10035 registered signal handlers. Whether open streams are flushed or closed, or temporary files are
 10036 removed is implementation-defined. Finally, the calling process is terminated with the
 10037 consequences described below.

10038 CX These functions shall terminate the calling process with the following consequences:

10039 **Note:** These consequences are all extensions to the ISO C standard and are not further CX shaded.
 10040 However, XSI extensions are shaded.

10041 XSI • All of the file descriptors, directory streams, conversion descriptors, and message catalog
 10042 descriptors open in the calling process shall be closed.

10043 XSI • If the parent process of the calling process is executing a *wait()* or *waitpid()*, and has neither
 10044 set its SA_NOCLDWAIT flag nor set SIGCHLD to SIG_IGN, it shall be notified of the calling
 10045 process' termination and the low-order eight bits (that is, bits 0377) of *status* shall be made
 10046 available to it. If the parent is not waiting, the child's status shall be made available to it
 10047 when the parent subsequently executes *wait()* or *waitpid()*.

10048 XSI The semantics of the *waitid()* function shall be equivalent to *wait()*.

10049 XSI • If the parent process of the calling process is not executing a *wait()* or *waitpid()*, and has
 10050 neither set its SA_NOCLDWAIT flag nor set SIGCHLD to SIG_IGN, the calling process shall
 10051 be transformed into a *zombie process*. A *zombie process* is an inactive process and it shall be

- 10052 deleted at some later time when its parent process executes `wait()` or `waitpid()`.
- 10053 XSI The semantics of the `waitid()` function shall be equivalent to `wait()`.
- 10054 • Termination of a process does not directly terminate its children. The sending of a SIGHUP
10055 signal as described below indirectly terminates children in some circumstances.
- 10056 • Either:
- 10057 If the implementation supports the SIGCHLD signal, a SIGCHLD shall be sent to the parent
10058 process.
- 10059 Or:
- 10060 XSI If the parent process has set its SA_NOCLDWAIT flag, or set SIGCHLD to SIG_IGN, the
10061 status shall be discarded, and the lifetime of the calling process shall end immediately. If
10062 SA_NOCLDWAIT is set, it is implementation-defined whether a SIGCHLD signal is sent to
10063 the parent process.
- 10064 • The parent process ID of all of the calling process' existing child processes and zombie
10065 processes shall be set to the process ID of an implementation-defined system process. That is,
10066 these processes shall be inherited by a special system process.
- 10067 XSI • Each attached shared-memory segment is detached and the value of `shm_nattch` (see
10068 `shmget()`) in the data structure associated with its shared memory ID shall be decremented by
10069 1.
- 10070 XSI • For each semaphore for which the calling process has set a `semadj` value (see `semop()`), that
10071 value shall be added to the `semval` of the specified semaphore.
- 10072 • If the process is a controlling process, the SIGHUP signal shall be sent to each process in the
10073 foreground process group of the controlling terminal belonging to the calling process.
- 10074 • If the process is a controlling process, the controlling terminal associated with the session
10075 shall be disassociated from the session, allowing it to be acquired by a new controlling
10076 process.
- 10077 • If the exit of the process causes a process group to become orphaned, and if any member of
10078 the newly-orphaned process group is stopped, then a SIGHUP signal followed by a
10079 SIGCONT signal shall be sent to each process in the newly-orphaned process group.
- 10080 SEM • All open named semaphores in the calling process shall be closed as if by appropriate calls to
10081 `sem_close()`.
- 10082 ML • Any memory locks established by the process via calls to `mlockall()` or `mlock()` shall be
10083 removed. If locked pages in the address space of the calling process are also mapped into the
10084 address spaces of other processes and are locked by those processes, the locks established by
10085 the other processes shall be unaffected by the call by this process to `_Exit()` or `_exit()`.
- 10086 MF|SHM • Memory mappings that were created in the process shall be unmapped before the process is
10087 destroyed.
- 10088 TYM • Any blocks of typed memory that were mapped in the calling process shall be unmapped, as
10089 if `munmap()` was implicitly called to unmap them.
- 10090 MSG • All open message queue descriptors in the calling process shall be closed as if by appropriate
10091 calls to `mq_close()`.
- 10092 AIO • Any outstanding cancelable asynchronous I/O operations may be canceled. Those
10093 asynchronous I/O operations that are not canceled shall complete as if the `_Exit()` or `_exit()`
10094 operation had not yet occurred, but any associated signal notifications shall be suppressed.

10095 The `_Exit()` or `_exit()` operation may block awaiting such I/O completion. Whether any I/O
 10096 is canceled, and which I/O may be canceled upon `_Exit()` or `_exit()`, is implementation-
 10097 defined.

10098 • Threads terminated by a call to `_Exit()` or `_exit()` shall not invoke their cancellation cleanup
 10099 handlers or per-thread data destructors.

10100 TRC • If the calling process is a trace controller process, any trace streams that were created by the
 10101 calling process shall be shut down as described by the `posix_trace_shutdown()` function, and
 10102 any process' mapping of trace event names to trace event type identifiers built for these trace
 10103 streams may be deallocated.

10104 RETURN VALUE

10105 These functions do not return.

10106 ERRORS

10107 No errors are defined.

10108 EXAMPLES

10109 None.

10110 APPLICATION USAGE

10111 Normally applications should use `exit()` rather than `_Exit()` or `_exit()`.

10112 RATIONALE

10113 Process Termination

10114 Early proposals drew a distinction between normal and abnormal process termination.
 10115 Abnormal termination was caused only by certain signals and resulted in implementation-
 10116 defined “actions”, as discussed below. Subsequent proposals distinguished three types of
 10117 termination: *normal termination* (as in the current specification), *simple abnormal termination*, and
 10118 *abnormal termination with actions*. Again the distinction between the two types of abnormal
 10119 termination was that they were caused by different signals and that implementation-defined
 10120 actions would result in the latter case. Given that these actions were completely
 10121 implementation-defined, the early proposals were only saying when the actions could occur and
 10122 how their occurrence could be detected, but not what they were. This was of little or no use to
 10123 conforming applications, and thus the distinction is not made in this volume of
 10124 IEEE Std 1003.1-2001.

10125 The implementation-defined actions usually include, in most historical implementations, the
 10126 creation of a file named **core** in the current working directory of the process. This file contains an
 10127 image of the memory of the process, together with descriptive information about the process,
 10128 perhaps sufficient to reconstruct the state of the process at the receipt of the signal.

10129 There is a potential security problem in creating a **core** file if the process was set-user-ID and the
 10130 current user is not the owner of the program, if the process was set-group-ID and none of the
 10131 user's groups match the group of the program, or if the user does not have permission to write in
 10132 the current directory. In this situation, an implementation either should not create a **core** file or
 10133 should make it unreadable by the user.

10134 Despite the silence of this volume of IEEE Std 1003.1-2001 on this feature, applications are
 10135 advised not to create files named **core** because of potential conflicts in many implementations.
 10136 Some implementations use a name other than **core** for the file; for example, by appending the
 10137 process ID to the filename.

10138 **Terminating a Process**

10139 It is important that the consequences of process termination as described occur regardless of
10140 whether the process called `_exit()` (perhaps indirectly through `exit()`) or instead was terminated
10141 due to a signal or for some other reason. Note that in the specific case of `exit()` this means that
10142 the *status* argument to `exit()` is treated in the same way as the *status* argument to `_exit()`.

10143 A language other than C may have other termination primitives than the C-language `exit()`
10144 function, and programs written in such a language should use its native termination primitives,
10145 but those should have as part of their function the behavior of `_exit()` as described.
10146 Implementations in languages other than C are outside the scope of this version of this volume
10147 of IEEE Std 1003.1-2001, however.

10148 As required by the ISO C standard, using **return** from `main()` has the same behavior (other than
10149 with respect to language scope issues) as calling `exit()` with the returned value. Reaching the end
10150 of the `main()` function has the same behavior as calling `exit(0)`.

10151 A value of zero (or `EXIT_SUCCESS`, which is required to be zero) for the argument *status*
10152 conventionally indicates successful termination. This corresponds to the specification for `exit()`
10153 in the ISO C standard. The convention is followed by utilities such as *make* and various shells,
10154 which interpret a zero status from a child process as success. For this reason, applications should
10155 not call `exit(0)` or `_exit(0)` when they terminate unsuccessfully; for example, in signal-catching
10156 functions.

10157 Historically, the implementation-defined process that inherits children whose parents have
10158 terminated without waiting on them is called *init* and has a process ID of 1.

10159 The sending of a `SIGHUP` to the foreground process group when a controlling process
10160 terminates corresponds to somewhat different historical implementations. In System V, the
10161 kernel sends a `SIGHUP` on termination of (essentially) a controlling process. In 4.2 BSD, the
10162 kernel does not send `SIGHUP` in a case like this, but the termination of a controlling process is
10163 usually noticed by a system daemon, which arranges to send a `SIGHUP` to the foreground
10164 process group with the `vhangup()` function. However, in 4.2 BSD, due to the behavior of the
10165 shells that support job control, the controlling process is usually a shell with no other processes
10166 in its process group. Thus, a change to make `_exit()` behave this way in such systems should not
10167 cause problems with existing applications.

10168 The termination of a process may cause a process group to become orphaned in either of two
10169 ways. The connection of a process group to its parent(s) outside of the group depends on both
10170 the parents and their children. Thus, a process group may be orphaned by the termination of the
10171 last connecting parent process outside of the group or by the termination of the last direct
10172 descendant of the parent process(es). In either case, if the termination of a process causes a
10173 process group to become orphaned, processes within the group are disconnected from their job
10174 control shell, which no longer has any information on the existence of the process group.
10175 Stopped processes within the group would languish forever. In order to avoid this problem,
10176 newly orphaned process groups that contain stopped processes are sent a `SIGHUP` signal and a
10177 `SIGCONT` signal to indicate that they have been disconnected from their session. The `SIGHUP`
10178 signal causes the process group members to terminate unless they are catching or ignoring
10179 `SIGHUP`. Under most circumstances, all of the members of the process group are stopped if any
10180 of them are stopped.

10181 The action of sending a `SIGHUP` and a `SIGCONT` signal to members of a newly orphaned
10182 process group is similar to the action of 4.2 BSD, which sends `SIGHUP` and `SIGCONT` to each
10183 stopped child of an exiting process. If such children exit in response to the `SIGHUP`, any
10184 additional descendants receive similar treatment at that time. In this volume of
10185 IEEE Std 1003.1-2001, the signals are sent to the entire process group at the same time. Also, in

10186 this volume of IEEE Std 1003.1-2001, but not in 4.2 BSD, stopped processes may be orphaned, but
 10187 may be members of a process group that is not orphaned; therefore, the action taken at `_exit()`
 10188 must consider processes other than child processes.

10189 It is possible for a process group to be orphaned by a call to `setpgid()` or `setsid()`, as well as by
 10190 process termination. This volume of IEEE Std 1003.1-2001 does not require sending `SIGHUP` and
 10191 `SIGCONT` in those cases, because, unlike process termination, those cases are not caused
 10192 accidentally by applications that are unaware of job control. An implementation can choose to
 10193 send `SIGHUP` and `SIGCONT` in those cases as an extension; such an extension must be
 10194 documented as required in `<signal.h>`.

10195 The ISO/IEC 9899:1999 standard adds the `_Exit()` function that results in immediate program
 10196 termination without triggering signals or `atexit()`-registered functions. In IEEE Std 1003.1-2001,
 10197 this is equivalent to the `_exit()` function.

10198 FUTURE DIRECTIONS

10199 None.

10200 SEE ALSO

10201 `atexit()`, `close()`, `fclose()`, `longjmp()`, `posix_trace_shutdown()`, `posix_trace_trid_eventid_open()`,
 10202 `semop()`, `shmget()`, `sigaction()`, `wait()`, `waitid()`, `waitpid()`, the Base Definitions volume of
 10203 IEEE Std 1003.1-2001, `<stdlib.h>`, `<unistd.h>`

10204 CHANGE HISTORY

10205 First released in Issue 1. Derived from Issue 1 of the SVID.

10206 Issue 5

10207 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 10208 Threads Extension.

10209 Interactions with the `SA_NOCLDWAIT` flag and `SIGCHLD` signal are further clarified.

10210 The values of `status` from `exit()` are better described.

10211 Issue 6

10212 Extensions beyond the ISO C standard are marked.

10213 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for
 10214 typed memory.

10215 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 10216 • The `_Exit()` function is included.
- 10217 • The DESCRIPTION is updated.

10218 The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

10219 References to the `wait3()` function are removed.

10220 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/16 is applied, correcting grammar in the
 10221 DESCRIPTION.

10222 **NAME**

10223 exp, expf, expl — exponential function

10224 **SYNOPSIS**

10225 #include <math.h>

10226 double exp(double x);

10227 float expf(float x);

10228 long double expl(long double x);

10229 **DESCRIPTION**

10230 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 10231 conflict between the requirements described here and the ISO C standard is unintentional. This
 10232 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

10233 These functions shall compute the base-*e* exponential of *x*.

10234 An application wishing to check for error situations should set *errno* to zero and call
 10235 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 10236 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 10237 zero, an error has occurred.

10238 **RETURN VALUE**10239 Upon successful completion, these functions shall return the exponential value of *x*.

10240 If the correct value would cause overflow, a range error shall occur and *exp()*, *expf()*, and *expl()*
 10241 shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

10242 If the correct value would cause underflow, and is not representable, a range error may occur,
 10243 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.

10244 **MX** If *x* is NaN, a NaN shall be returned.10245 If *x* is ±0, 1 shall be returned.10246 If *x* is -Inf, +0 shall be returned.10247 If *x* is +Inf, *x* shall be returned.

10248 If the correct value would cause underflow, and is representable, a range error may occur and
 10249 the correct value shall be returned.

10250 **ERRORS**

10251 These functions shall fail if:

10252 **Range Error** The result overflows.

10253 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 10254 then *errno* shall be set to [ERANGE]. If the integer expression
 10255 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 10256 floating-point exception shall be raised.

10257 These functions may fail if:

10258 **Range Error** The result underflows.

10259 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 10260 then *errno* shall be set to [ERANGE]. If the integer expression
 10261 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 10262 floating-point exception shall be raised.

10263 **EXAMPLES**

10264 None.

10265 **APPLICATION USAGE**

10266 Note that for IEEE Std 754-1985 **double**, $709.8 < x$ implies $\exp(x)$ has overflowed. The value
10267 $x < -708.4$ implies $\exp(x)$ has underflowed.

10268 On error, the expressions `(math_errhandling & MATH_ERRNO)` and `(math_errhandling &`
10269 `MATH_ERREXCEPT)` are independent of each other, but at least one of them must be non-zero.

10270 **RATIONALE**

10271 None.

10272 **FUTURE DIRECTIONS**

10273 None.

10274 **SEE ALSO**

10275 `feclearexcept()`, `fetestexcept()`, `isnan()`, `log()`, the Base Definitions volume of IEEE Std 1003.1-2001,
10276 Section 4.18, Treatment of Error Conditions for Mathematical Functions, `<math.h>`

10277 **CHANGE HISTORY**

10278 First released in Issue 1. Derived from Issue 1 of the SVID.

10279 **Issue 5**

10280 The DESCRIPTION is updated to indicate how an application should check for an error. This
10281 text was previously published in the APPLICATION USAGE section.

10282 **Issue 6**10283 The `expf()` and `expl()` functions are added for alignment with the ISO/IEC 9899:1999 standard.

10284 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
10285 revised to align with the ISO/IEC 9899:1999 standard.

10286 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
10287 marked.

10288 **NAME**

10289 exp2, exp2f, exp2l — exponential base 2 functions

10290 **SYNOPSIS**

10291 #include <math.h>

10292 double exp2(double x);

10293 float exp2f(float x);

10294 long double exp2l(long double x);

10295 **DESCRIPTION**

10296 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 10297 conflict between the requirements described here and the ISO C standard is unintentional. This
 10298 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

10299 These functions shall compute the base-2 exponential of *x*.

10300 An application wishing to check for error situations should set *errno* to zero and call
 10301 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 10302 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 10303 zero, an error has occurred.

10304 **RETURN VALUE**10305 Upon successful completion, these functions shall return 2^x .

10306 If the correct value would cause overflow, a range error shall occur and *exp2()*, *exp2f()*, and
 10307 *exp2l()* shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
 10308 respectively.

10309 If the correct value would cause underflow, and is not representable, a range error may occur,
 10310 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.

10311 **MX** If *x* is NaN, a NaN shall be returned.10312 If *x* is ± 0 , 1 shall be returned.10313 If *x* is $-\text{Inf}$, +0 shall be returned.10314 If *x* is $+\text{Inf}$, *x* shall be returned.

10315 If the correct value would cause underflow, and is representable, a range error may occur and
 10316 the correct value shall be returned.

10317 **ERRORS**

10318 These functions shall fail if:

10319 **Range Error** The result overflows.

10320 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 10321 then *errno* shall be set to [ERANGE]. If the integer expression
 10322 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 10323 floating-point exception shall be raised.

10324 These functions may fail if:

10325 **Range Error** The result underflows.

10326 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 10327 then *errno* shall be set to [ERANGE]. If the integer expression
 10328 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 10329 floating-point exception shall be raised.

10330 **EXAMPLES**

10331 None.

10332 **APPLICATION USAGE**

10333 For IEEE Std 754-1985 **double**, $1024 \leq x$ implies $\text{exp2}(x)$ has overflowed. The value $x < -1022$
10334 implies $\text{exp}(x)$ has underflowed.

10335 On error, the expressions `(math_errhandling & MATH_ERRNO)` and `(math_errhandling &`
10336 `MATH_ERREXCEPT)` are independent of each other, but at least one of them must be non-zero.

10337 **RATIONALE**

10338 None.

10339 **FUTURE DIRECTIONS**

10340 None.

10341 **SEE ALSO**

10342 `exp()`, `feclearexcept()`, `fetestexcept()`, `isnan()`, `log()`, the Base Definitions volume of
10343 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
10344 `<math.h>`

10345 **CHANGE HISTORY**

10346 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

10347 **NAME**

10348 expm1, expm1f, expm1l — compute exponential functions

10349 **SYNOPSIS**

10350 #include <math.h>

10351 double expm1(double x);

10352 float expm1f(float x);

10353 long double expm1l(long double x);

10354 **DESCRIPTION**

10355 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 10356 conflict between the requirements described here and the ISO C standard is unintentional. This
 10357 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

10358 These functions shall compute $e^x-1.0$.

10359 An application wishing to check for error situations should set *errno* to zero and call
 10360 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 10361 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 10362 zero, an error has occurred.

10363 **RETURN VALUE**10364 Upon successful completion, these functions return $e^x-1.0$.

10365 If the correct value would cause overflow, a range error shall occur and *expm1()*, *expm1f()*, and
 10366 *expm1l()* shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
 10367 respectively.

10368 **MX** If *x* is NaN, a NaN shall be returned.10369 If *x* is ± 0 , ± 0 shall be returned.10370 If *x* is $-\text{Inf}$, -1 shall be returned.10371 If *x* is $+\text{Inf}$, *x* shall be returned.10372 If *x* is subnormal, a range error may occur and *x* should be returned.10373 **ERRORS**

10374 These functions shall fail if:

10375 **Range Error** The result overflows.

10376 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 10377 then *errno* shall be set to [ERANGE]. If the integer expression
 10378 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 10379 floating-point exception shall be raised.

10380 These functions may fail if:

10381 **MX** **Range Error** The value of *x* is subnormal.

10382 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 10383 then *errno* shall be set to [ERANGE]. If the integer expression
 10384 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 10385 floating-point exception shall be raised.

10386 **EXAMPLES**

10387 None.

10388 **APPLICATION USAGE**10389 The value of $\text{expm1}(x)$ may be more accurate than $\exp(x)-1.0$ for small values of x .10390 The $\text{expm1}()$ and $\text{log1p}()$ functions are useful for financial calculations of $((1+x)^n-1)/x$, namely:10391 $\text{expm1}(n * \text{log1p}(x)) / x$ 10392 when x is very small (for example, when calculating small daily interest rates). These functions
10393 also simplify writing accurate inverse hyperbolic functions.10394 For IEEE Std 754-1985 **double**, $709.8 < x$ implies $\text{expm1}(x)$ has overflowed.10395 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`
10396 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.10397 **RATIONALE**

10398 None.

10399 **FUTURE DIRECTIONS**

10400 None.

10401 **SEE ALSO**10402 $\text{exp}()$, $\text{feclearexcept}()$, $\text{fetestexcept}()$, $\text{ilogb}()$, $\text{log1p}()$, the Base Definitions volume of
10403 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,
10404 `<math.h>`10405 **CHANGE HISTORY**

10406 First released in Issue 4, Version 2.

10407 **Issue 5**

10408 Moved from X/OPEN UNIX extension to BASE.

10409 **Issue 6**10410 The $\text{expm1f}()$ and $\text{expm1l}()$ functions are added for alignment with the ISO/IEC 9899:1999
10411 standard.10412 The $\text{expm1}()$ function is no longer marked as an extension.10413 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
10414 revised to align with the ISO/IEC 9899:1999 standard.10415 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
10416 marked.

10417 **NAME**

10418 fabs, fabsf, fabsl — absolute value function

10419 **SYNOPSIS**

10420 #include <math.h>

10421 double fabs(double x);

10422 float fabsf(float x);

10423 long double fabsl(long double x);

10424 **DESCRIPTION**

10425 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
10426 conflict between the requirements described here and the ISO C standard is unintentional. This
10427 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

10428 These functions shall compute the absolute value of their argument x , $|x|$.

10429 **RETURN VALUE**

10430 Upon successful completion, these functions shall return the absolute value of x .

10431 **MX** If x is NaN, a NaN shall be returned.

10432 If x is ± 0 , $+0$ shall be returned.

10433 If x is $\pm\text{Inf}$, $+\text{Inf}$ shall be returned.

10434 **ERRORS**

10435 No errors are defined.

10436 **EXAMPLES**

10437 None.

10438 **APPLICATION USAGE**

10439 None.

10440 **RATIONALE**

10441 None.

10442 **FUTURE DIRECTIONS**

10443 None.

10444 **SEE ALSO**

10445 *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001, <math.h>

10446 **CHANGE HISTORY**

10447 First released in Issue 1. Derived from Issue 1 of the SVID.

10448 **Issue 5**

10449 The DESCRIPTION is updated to indicate how an application should check for an error. This
10450 text was previously published in the APPLICATION USAGE section.

10451 **Issue 6**

10452 The *fabsf()* and *fabsl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

10453 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
10454 revised to align with the ISO/IEC 9899:1999 standard.

10455 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
10456 marked.

10457 **NAME**

10458 fattach — attach a STREAMS-based file descriptor to a file in the file system name space
 10459 (STREAMS)

10460 **SYNOPSIS**

```
10461 XSR #include <stropts.h>
```

```
10462 int fattach(int fildev, const char *path);
```

10463

10464 **DESCRIPTION**

10465 The *fattach()* function shall attach a STREAMS-based file descriptor to a file, effectively
 10466 associating a pathname with *fildev*. The application shall ensure that the *fildev* argument is a
 10467 valid open file descriptor associated with a STREAMS file. The *path* argument points to a
 10468 pathname of an existing file. The application shall have the appropriate privileges or be the
 10469 owner of the file named by *path* and have write permission. A successful call to *fattach()* shall
 10470 cause all pathnames that name the file named by *path* to name the STREAMS file associated with
 10471 *fildev*, until the STREAMS file is detached from the file. A STREAMS file can be attached to more
 10472 than one file and can have several pathnames associated with it.

10473 The attributes of the named STREAMS file shall be initialized as follows: the permissions, user
 10474 ID, group ID, and times are set to those of the file named by *path*, the number of links is set to 1,
 10475 and the size and device identifier are set to those of the STREAMS file associated with *fildev*. If
 10476 any attributes of the named STREAMS file are subsequently changed (for example, by *chmod()*),
 10477 neither the attributes of the underlying file nor the attributes of the STREAMS file to which *fildev*
 10478 refers shall be affected.

10479 File descriptors referring to the underlying file, opened prior to an *fattach()* call, shall continue to
 10480 refer to the underlying file.

10481 **RETURN VALUE**

10482 Upon successful completion, *fattach()* shall return 0. Otherwise, -1 shall be returned and *errno*
 10483 set to indicate the error.

10484 **ERRORS**

10485 The *fattach()* function shall fail if:

10486 [EACCES] Search permission is denied for a component of the path prefix, or the process
 10487 is the owner of *path* but does not have write permissions on the file named by
 10488 *path*.

10489 [EBADF] The *fildev* argument is not a valid open file descriptor.

10490 [EBUSY] The file named by *path* is currently a mount point or has a STREAMS file
 10491 attached to it.

10492 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 10493 argument.

10494 [ENAMETOOLONG]

10495 The size of *path* exceeds {PATH_MAX} or a component of *path* is longer than
 10496 {NAME_MAX}.

10497 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

10498 [ENOTDIR] A component of the path prefix is not a directory.

10499 [EPERM] The effective user ID of the process is not the owner of the file named by *path*
 10500 and the process does not have appropriate privilege.

- 10501 The *fattach()* function may fail if:
- 10502 [EINVAL] The *fildev* argument does not refer to a STREAMS file.
- 10503 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
10504 resolution of the *path* argument.
- 10505 [ENAMETOOLONG]
10506 Pathname resolution of a symbolic link produced an intermediate result
10507 whose length exceeds {PATH_MAX}.
- 10508 [EXDEV] A link to a file on another file system was attempted.

10509 **EXAMPLES**10510 **Attaching a File Descriptor to a File**

10511 In the following example, *fd* refers to an open STREAMS file. The call to *fattach()* associates this
10512 STREAM with the file */tmp/named-STREAM*, such that any future calls to open */tmp/named-*
10513 *STREAM*, prior to breaking the attachment via a call to *fdetach()*, will instead create a new file
10514 handle referring to the STREAMS file associated with *fd*.

```
10515 #include <stropts.h>
10516 ...
10517     int fd;
10518     char *filename = "/tmp/named-STREAM";
10519     int ret;
10520
10521     ret = fattach(fd, filename);
```

10521 **APPLICATION USAGE**

10522 The *fattach()* function behaves similarly to the traditional *mount()* function in the way a file is
10523 temporarily replaced by the root directory of the mounted file system. In the case of *fattach()*, the
10524 replaced file need not be a directory and the replacing file is a STREAMS file.

10525 **RATIONALE**

10526 The file attributes of a file which has been the subject of an *fattach()* call are specifically set
10527 because of an artefact of the original implementation. The internal mechanism was the same as
10528 for the *mount()* function. Since *mount()* is typically only applied to directories, the effects when
10529 applied to a regular file are a little surprising, especially as regards the link count which rigidly
10530 remains one, even if there were several links originally and despite the fact that all original links
10531 refer to the STREAM as long as the *fattach()* remains in effect.

10532 **FUTURE DIRECTIONS**

10533 None.

10534 **SEE ALSO**

10535 *fdetach()*, *isastream()*, the Base Definitions volume of IEEE Std 1003.1-2001, <*stropts.h*>

10536 **CHANGE HISTORY**

10537 First released in Issue 4, Version 2.

10538 **Issue 5**

10539 Moved from X/OPEN UNIX extension to BASE.

10540 The [EXDEV] error is added to the list of optional errors in the ERRORS section.

10541 **Issue 6**

10542 This function is marked as part of the XSI STREAMS Option Group.

10543 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

10544 The wording of the mandatory [ELOOP] error condition is updated, and a second optional

10545 [ELOOP] error condition is added.

10546 **NAME**

10547 fchdir — change working directory

10548 **SYNOPSIS**

10549 XSI #include <unistd.h>

10550 int fchdir(int *fildev*);

10551

10552 **DESCRIPTION**10553 The *fchdir()* function shall be equivalent to *chdir()* except that the directory that is to be the new
10554 current working directory is specified by the file descriptor *fildev*.10555 A conforming application can obtain a file descriptor for a file of type directory using *open()*,
10556 provided that the file status flags and access modes do not contain *O_WRONLY* or *O_RDWR*.10557 **RETURN VALUE**10558 Upon successful completion, *fchdir()* shall return 0. Otherwise, it shall return -1 and set *errno* to
10559 indicate the error. On failure the current working directory shall remain unchanged.10560 **ERRORS**10561 The *fchdir()* function shall fail if:10562 [EACCES] Search permission is denied for the directory referenced by *fildev*.10563 [EBADF] The *fildev* argument is not an open file descriptor.10564 [ENOTDIR] The open file descriptor *fildev* does not refer to a directory.10565 The *fchdir()* may fail if:10566 [EINTR] A signal was caught during the execution of *fchdir()*.

10567 [EIO] An I/O error occurred while reading from or writing to the file system.

10568 **EXAMPLES**

10569 None.

10570 **APPLICATION USAGE**

10571 None.

10572 **RATIONALE**

10573 None.

10574 **FUTURE DIRECTIONS**

10575 None.

10576 **SEE ALSO**10577 *chdir()*, the Base Definitions volume of IEEE Std 1003.1-2001, <unistd.h>10578 **CHANGE HISTORY**

10579 First released in Issue 4, Version 2.

10580 **Issue 5**

10581 Moved from X/OPEN UNIX extension to BASE.

10582 **NAME**

10583 fchmod — change mode of a file

10584 **SYNOPSIS**

10585 #include <sys/stat.h>

10586 int fchmod(int *fildev*, mode_t *mode*);10587 **DESCRIPTION**10588 The *fchmod()* function shall be equivalent to *chmod()* except that the file whose permissions are
10589 changed is specified by the file descriptor *fildev*.10590 SHM If *fildev* references a shared memory object, the *fchmod()* function need only affect the S_IRUSR,
10591 S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits.10592 TYM If *fildev* references a typed memory object, the behavior of *fchmod()* is unspecified.10593 If *fildev* refers to a socket, the behavior of *fchmod()* is unspecified.10594 XSR If *fildev* refers to a STREAM (which is *fattach()*-ed into the file system name space) the call
10595 returns successfully, doing nothing.10596 **RETURN VALUE**10597 Upon successful completion, *fchmod()* shall return 0. Otherwise, it shall return -1 and set *errno* to
10598 indicate the error.10599 **ERRORS**10600 The *fchmod()* function shall fail if:10601 [EBADF] The *fildev* argument is not an open file descriptor.10602 [EPERM] The effective user ID does not match the owner of the file and the process
10603 does not have appropriate privilege.10604 [EROFS] The file referred to by *fildev* resides on a read-only file system.10605 The *fchmod()* function may fail if:10606 XSI [EINTR] The *fchmod()* function was interrupted by a signal.10607 XSI [EINVAL] The value of the *mode* argument is invalid.10608 [EINVAL] The *fildev* argument refers to a pipe and the implementation disallows
10609 execution of *fchmod()* on a pipe.10610 **EXAMPLES**10611 **Changing the Current Permissions for a File**10612 The following example shows how to change the permissions for a file named */home/cnd/mod1*
10613 so that the owner and group have read/write/execute permissions, but the world only has
10614 read/write permissions.

10615 #include <sys/stat.h>

10616 #include <fcntl.h>

10617 mode_t mode;

10618 int fildev;

10619 ...

10620 fildev = open("/home/cnd/mod1", O_RDWR);

10621 fchmod(fildev, S_IRWXU | S_IRWXG | S_IROTH | S_IWOTH);

10622 **APPLICATION USAGE**

10623 None.

10624 **RATIONALE**

10625 None.

10626 **FUTURE DIRECTIONS**

10627 None.

10628 **SEE ALSO**

10629 *chmod()*, *chown()*, *creat()*, *fcntl()*, *fstatvfs()*, *mknod()*, *open()*, *read()*, *stat()*, *write()*, the Base
10630 Definitions volume of IEEE Std 1003.1-2001, <sys/stat.h>

10631 **CHANGE HISTORY**

10632 First released in Issue 4, Version 2.

10633 **Issue 5**

10634 Moved from X/OPEN UNIX extension to BASE and aligned with *fchmod()* in the POSIX
10635 Realtime Extension. Specifically, the second paragraph of the DESCRIPTION is added and a
10636 second instance of [EINVAL] is defined in the list of optional errors.

10637 **Issue 6**

10638 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by stating that *fchmod()*
10639 behavior is unspecified for typed memory objects.

10640 **NAME**

10641 fchown — change owner and group of a file

10642 **SYNOPSIS**

10643 #include <unistd.h>

10644 int fchown(int *fildev*, uid_t *owner*, gid_t *group*);10645 **DESCRIPTION**10646 The *fchown()* function shall be equivalent to *chown()* except that the file whose owner and group
10647 are changed is specified by the file descriptor *fildev*.10648 **RETURN VALUE**10649 Upon successful completion, *fchown()* shall return 0. Otherwise, it shall return -1 and set *errno* to
10650 indicate the error.10651 **ERRORS**10652 The *fchown()* function shall fail if:10653 [EBADF] The *fildev* argument is not an open file descriptor.10654 [EPERM] The effective user ID does not match the owner of the file or the process does
10655 not have appropriate privilege and `_POSIX_CHOWN_RESTRICTED` indicates
10656 that such privilege is required.10657 [EROFS] The file referred to by *fildev* resides on a read-only file system.10658 The *fchown()* function may fail if:10659 [EINVAL] The owner or group ID is not a value supported by the implementation. The
10660 XSR *fildev* argument refers to a pipe or socket or an *fcntl()*-ed `STREAM` and the
10661 implementation disallows execution of *fchown()* on a pipe.

10662 [EIO] A physical I/O error has occurred.

10663 [EINTR] The *fchown()* function was interrupted by a signal which was caught.10664 **EXAMPLES**10665 **Changing the Current Owner of a File**10666 The following example shows how to change the owner of a file named `/home/cnd/mod1` to
10667 “jones” and the group to “cnd”.10668 The numeric value for the user ID is obtained by extracting the user ID from the user database
10669 entry associated with “jones”. Similarly, the numeric value for the group ID is obtained by
10670 extracting the group ID from the group database entry associated with “cnd”. This example
10671 assumes the calling program has appropriate privileges.10672 #include <sys/types.h>
10673 #include <unistd.h>
10674 #include <fcntl.h>
10675 #include <pwd.h>
10676 #include <grp.h>

10677 struct passwd *pwd;
10678 struct group *grp;
10679 int fildev;
10680 ...
10681 fildev = open("/home/cnd/mod1", O_RDWR);
10682 pwd = getpwnam("jones");

```
10683     grp = getgrnam("cnd");
10684     fchown(fildes, pwd->pw_uid, grp->gr_gid);
```

10685 APPLICATION USAGE

10686 None.

10687 RATIONALE

10688 None.

10689 FUTURE DIRECTIONS

10690 None.

10691 SEE ALSO

10692 *chown()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

10693 CHANGE HISTORY

10694 First released in Issue 4, Version 2.

10695 Issue 5

10696 Moved from X/OPEN UNIX extension to BASE.

10697 Issue 6

10698 The following changes were made to align with the IEEE P1003.1a draft standard:

10699 • Clarification is added that a call to *fchown()* may not be allowed on a pipe.

10700 The *fchown()* function is defined as mandatory.

10701 **NAME**

10702 fclose — close a stream

10703 **SYNOPSIS**

10704 #include <stdio.h>

10705 int fclose(FILE *stream);

10706 **DESCRIPTION**

10707 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 10708 conflict between the requirements described here and the ISO C standard is unintentional. This
 10709 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

10710 The *fclose()* function shall cause the stream pointed to by *stream* to be flushed and the associated
 10711 file to be closed. Any unwritten buffered data for the stream shall be written to the file; any
 10712 unread buffered data shall be discarded. Whether or not the call succeeds, the stream shall be
 10713 disassociated from the file and any buffer set by the *setbuf()* or *setvbuf()* function shall be
 10714 disassociated from the stream. If the associated buffer was automatically allocated, it shall be
 10715 deallocated.

10716 CX The *fclose()* function shall mark for update the *st_ctime* and *st_mtime* fields of the underlying file,
 10717 if the stream was writable, and if buffered data remains that has not yet been written to the file.
 10718 The *fclose()* function shall perform the equivalent of a *close()* on the file descriptor that is
 10719 associated with the stream pointed to by *stream*.

10720 After the call to *fclose()*, any use of *stream* results in undefined behavior.

10721 **RETURN VALUE**

10722 CX Upon successful completion, *fclose()* shall return 0; otherwise, it shall return EOF and set *errno* to
 10723 indicate the error.

10724 **ERRORS**

10725 The *fclose()* function shall fail if:

10726 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
 10727 process would be delayed in the write operation.

10728 CX [EBADF] The file descriptor underlying stream is not valid.

10729 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

10730 XSI [EFBIG] An attempt was made to write a file that exceeds the process' file size limit.

10731 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
 10732 offset maximum associated with the corresponding stream.

10733 CX [EINTR] The *fclose()* function was interrupted by a signal.

10734 CX [EIO] The process is a member of a background process group attempting to write
 10735 to its controlling terminal, TOSTOP is set, the process is neither ignoring nor
 10736 blocking SIGTTOU, and the process group of the process is orphaned. This
 10737 error may also be returned under implementation-defined conditions.

10738 CX [ENOSPC] There was no free space remaining on the device containing the file.

10739 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
 10740 any process. A SIGPIPE signal shall also be sent to the thread.

10741 The *fclose()* function may fail if:

10742 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
 10743 capabilities of the device.

10744 **EXAMPLES**

10745 None.

10746 **APPLICATION USAGE**

10747 None.

10748 **RATIONALE**

10749 None.

10750 **FUTURE DIRECTIONS**

10751 None.

10752 **SEE ALSO**

10753 *close()*, *fopen()*, *getrlimit()*, *ulimit()*, the Base Definitions volume of IEEE Std 1003.1-2001,
10754 <**stdio.h**>

10755 **CHANGE HISTORY**

10756 First released in Issue 1. Derived from Issue 1 of the SVID.

10757 **Issue 5**

10758 Large File Summit extensions are added.

10759 **Issue 6**

10760 Extensions beyond the ISO C standard are marked.

10761 The following new requirements on POSIX implementations derive from alignment with the
10762 Single UNIX Specification:

- 10763 • The [EFBIG] error is added as part of the large file support extensions.
- 10764 • The [ENXIO] optional error condition is added.

10765 The DESCRIPTION is updated to note that the stream and any buffer are disassociated whether
10766 or not the call succeeds. This is for alignment with the ISO/IEC 9899:1999 standard.

10767 **NAME**

10768 fcntl — file control

10769 **SYNOPSIS**

10770 OH #include <unistd.h>

10771 #include <fcntl.h>

10772 int fcntl(int *fildes*, int *cmd*, ...);10773 **DESCRIPTION**10774 The *fcntl()* function shall perform the operations described below on open files. The *fildes*
10775 argument is a file descriptor.10776 The available values for *cmd* are defined in <fcntl.h> and are as follows:

10777 **F_DUPFD** Return a new file descriptor which shall be the lowest numbered available
10778 (that is, not already open) file descriptor greater than or equal to the third
10779 argument, *arg*, taken as an integer of type **int**. The new file descriptor shall
10780 refer to the same open file description as the original file descriptor, and shall
10781 share any locks. The FD_CLOEXEC flag associated with the new file
10782 descriptor shall be cleared to keep the file open across calls to one of the *exec*
10783 functions.

10784 **F_GETFD** Get the file descriptor flags defined in <fcntl.h> that are associated with the
10785 file descriptor *fildes*. File descriptor flags are associated with a single file
10786 descriptor and do not affect other file descriptors that refer to the same file.

10787 **F_SETFD** Set the file descriptor flags defined in <fcntl.h>, that are associated with *fildes*,
10788 to the third argument, *arg*, taken as type **int**. If the FD_CLOEXEC flag in the
10789 third argument is 0, the file shall remain open across the *exec* functions;
10790 otherwise, the file shall be closed upon successful execution of one of the *exec*
10791 functions.

10792 **F_GETFL** Get the file status flags and file access modes, defined in <fcntl.h>, for the file
10793 description associated with *fildes*. The file access modes can be extracted from
10794 the return value using the mask O_ACCMODE, which is defined in <fcntl.h>.
10795 File status flags and file access modes are associated with the file description
10796 and do not affect other file descriptors that refer to the same file with different
10797 open file descriptions.

10798 **F_SETFL** Set the file status flags, defined in <fcntl.h>, for the file description associated
10799 with *fildes* from the corresponding bits in the third argument, *arg*, taken as
10800 type **int**. Bits corresponding to the file access mode and the file creation flags,
10801 as defined in <fcntl.h>, that are set in *arg* shall be ignored. If any bits in *arg*
10802 other than those mentioned here are changed by the application, the result is
10803 unspecified.

10804 **F_GETOWN** If *fildes* refers to a socket, get the process or process group ID specified to
10805 receive SIGURG signals when out-of-band data is available. Positive values
10806 indicate a process ID; negative values, other than -1, indicate a process group
10807 ID. If *fildes* does not refer to a socket, the results are unspecified.

10808 **F_SETOWN** If *fildes* refers to a socket, set the process or process group ID specified to
10809 receive SIGURG signals when out-of-band data is available, using the value of
10810 the third argument, *arg*, taken as type **int**. Positive values indicate a process
10811 ID; negative values, other than -1, indicate a process group ID. If *fildes* does
10812 not refer to a socket, the results are unspecified.

10813 The following values for *cmd* are available for advisory record locking. Record locking shall be
 10814 supported for regular files, and may be supported for other files.

10815 **F_GETLK** Get the first lock which blocks the lock description pointed to by the third
 10816 argument, *arg*, taken as a pointer to type **struct flock**, defined in `<fcntl.h>`.
 10817 The information retrieved shall overwrite the information passed to *fcntl()* in
 10818 the structure **flock**. If no lock is found that would prevent this lock from
 10819 being created, then the structure shall be left unchanged except for the lock
 10820 type which shall be set to **F_UNLCK**.

10821 **F_SETLK** Set or clear a file segment lock according to the lock description pointed to by
 10822 the third argument, *arg*, taken as a pointer to type **struct flock**, defined in
 10823 `<fcntl.h>`. **F_SETLK** can establish shared (or read) locks (**F_RDLCK**) or
 10824 exclusive (or write) locks (**F_WRLCK**), as well as to remove either type of lock
 10825 (**F_UNLCK**). **F_RDLCK**, **F_WRLCK**, and **F_UNLCK** are defined in `<fcntl.h>`.
 10826 If a shared or exclusive lock cannot be set, *fcntl()* shall return immediately
 10827 with a return value of `-1`.

10828 **F_SETLKW** This command shall be equivalent to **F_SETLK** except that if a shared or
 10829 exclusive lock is blocked by other locks, the thread shall wait until the request
 10830 can be satisfied. If a signal that is to be caught is received while *fcntl()* is
 10831 waiting for a region, *fcntl()* shall be interrupted. Upon return from the signal
 10832 handler, *fcntl()* shall return `-1` with *errno* set to `[EINTR]`, and the lock
 10833 operation shall not be done.

10834 Additional implementation-defined values for *cmd* may be defined in `<fcntl.h>`. Their names
 10835 shall start with **F_**.

10836 When a shared lock is set on a segment of a file, other processes shall be able to set shared locks
 10837 on that segment or a portion of it. A shared lock prevents any other process from setting an
 10838 exclusive lock on any portion of the protected area. A request for a shared lock shall fail if the
 10839 file descriptor was not opened with read access.

10840 An exclusive lock shall prevent any other process from setting a shared lock or an exclusive lock
 10841 on any portion of the protected area. A request for an exclusive lock shall fail if the file
 10842 descriptor was not opened with write access.

10843 The structure **flock** describes the type (*l_type*), starting offset (*l_whence*), relative offset (*l_start*),
 10844 size (*l_len*), and process ID (*l_pid*) of the segment of the file to be affected.

10845 The value of *l_whence* is **SEEK_SET**, **SEEK_CUR**, or **SEEK_END**, to indicate that the relative
 10846 offset *l_start* bytes shall be measured from the start of the file, current position, or end of the file,
 10847 respectively. The value of *l_len* is the number of consecutive bytes to be locked. The value of
 10848 *l_len* may be negative (where the definition of **off_t** permits negative values of *l_len*). The *l_pid*
 10849 field is only used with **F_GETLK** to return the process ID of the process holding a blocking lock.
 10850 After a successful **F_GETLK** request, when a blocking lock is found, the values returned in the
 10851 **flock** structure shall be as follows:

10852 *l_type* Type of blocking lock found.

10853 *l_whence* **SEEK_SET**.

10854 *l_start* Start of the blocking lock.

10855 *l_len* Length of the blocking lock.

10856 *l_pid* Process ID of the process that holds the blocking lock.

- 10857 If the command is F_SETLKW and the process must wait for another process to release a lock,
 10858 then the range of bytes to be locked shall be determined before the *fcntl()* function blocks. If the
 10859 file size or file descriptor seek offset change while *fcntl()* is blocked, this shall not affect the
 10860 range of bytes locked.
- 10861 If *l_len* is positive, the area affected shall start at *l_start* and end at *l_start+l_len-1*. If *l_len* is
 10862 negative, the area affected shall start at *l_start+l_len* and end at *l_start-1*. Locks may start and
 10863 extend beyond the current end of a file, but shall not extend before the beginning of the file. A
 10864 lock shall be set to extend to the largest possible value of the file offset for that file by setting
 10865 *l_len* to 0. If such a lock also has *l_start* set to 0 and *l_whence* is set to SEEK_SET, the whole file
 10866 shall be locked.
- 10867 There shall be at most one type of lock set for each byte in the file. Before a successful return
 10868 from an F_SETLK or an F_SETLKW request when the calling process has previously existing
 10869 locks on bytes in the region specified by the request, the previous lock type for each byte in the
 10870 specified region shall be replaced by the new lock type. As specified above under the
 10871 descriptions of shared locks and exclusive locks, an F_SETLK or an F_SETLKW request
 10872 (respectively) shall fail or block when another process has existing locks on bytes in the specified
 10873 region and the type of any of those locks conflicts with the type specified in the request.
- 10874 All locks associated with a file for a given process shall be removed when a file descriptor for
 10875 that file is closed by that process or the process holding that file descriptor terminates. Locks are
 10876 not inherited by a child process.
- 10877 A potential for deadlock occurs if a process controlling a locked region is put to sleep by
 10878 attempting to lock another process' locked region. If the system detects that sleeping until a
 10879 locked region is unlocked would cause a deadlock, *fcntl()* shall fail with an [EDEADLK] error.
- 10880 An unlock (F_UNLCK) request in which *l_len* is non-zero and the offset of the last byte of the
 10881 requested segment is the maximum value for an object of type **off_t**, when the process has an
 10882 existing lock in which *l_len* is 0 and which includes the last byte of the requested segment, shall
 10883 be treated as a request to unlock from the start of the requested segment with an *l_len* equal to 0.
 10884 Otherwise, an unlock (F_UNLCK) request shall attempt to unlock only the requested segment.
- 10885 SHM When the file descriptor *fdes* refers to a shared memory object, the behavior of *fcntl()* shall be
 10886 the same as for a regular file except the effect of the following values for the argument *cmd* shall
 10887 be unspecified: F_SETFL, F_GETLK, F_SETLK, and F_SETLKW.
- 10888 TYM If *fdes* refers to a typed memory object, the result of the *fcntl()* function is unspecified.
- 10889 **RETURN VALUE**
- 10890 Upon successful completion, the value returned shall depend on *cmd* as follows:
- | | | |
|-------|----------|--|
| 10891 | F_DUPFD | A new file descriptor. |
| 10892 | F_GETFD | Value of flags defined in <fcntl.h>. The return value shall not be negative. |
| 10893 | F_SETFD | Value other than -1. |
| 10894 | F_GETFL | Value of file status flags and access modes. The return value is not negative. |
| 10895 | F_SETFL | Value other than -1. |
| 10896 | F_GETLK | Value other than -1. |
| 10897 | F_SETLK | Value other than -1. |
| 10898 | F_SETLKW | Value other than -1. |
| 10899 | F_GETOWN | Value of the socket owner process or process group; this will not be -1. |

- 10900 F_SETOWN Value other than -1.
- 10901 Otherwise, -1 shall be returned and *errno* set to indicate the error.
- 10902 **ERRORS**
- 10903 The *fcntl()* function shall fail if:
- 10904 [EACCES] or [EAGAIN]
- 10905 The *cmd* argument is F_SETLK; the type of lock (*l_type*) is a shared (F_RDLCK)
- 10906 or exclusive (F_WRLCK) lock and the segment of a file to be locked is already
- 10907 exclusive-locked by another process, or the type is an exclusive lock and some
- 10908 portion of the segment of a file to be locked is already shared-locked or
- 10909 exclusive-locked by another process.
- 10910 [EBADF] The *fildev* argument is not a valid open file descriptor, or the argument *cmd* is
- 10911 F_SETLK or F_SETLKW, the type of lock, *l_type*, is a shared lock (F_RDLCK),
- 10912 and *fildev* is not a valid file descriptor open for reading, or the type of lock,
- 10913 *l_type*, is an exclusive lock (F_WRLCK), and *fildev* is not a valid file descriptor
- 10914 open for writing.
- 10915 [EINTR] The *cmd* argument is F_SETLKW and the function was interrupted by a signal.
- 10916 [EINVAL] The *cmd* argument is invalid, or the *cmd* argument is F_DUPFD and *arg* is
- 10917 negative or greater than or equal to {OPEN_MAX}, or the *cmd* argument is
- 10918 F_GETLK, F_SETLK, or F_SETLKW and the data pointed to by *arg* is not valid,
- 10919 or *fildev* refers to a file that does not support locking.
- 10920 [EMFILE] The argument *cmd* is F_DUPFD and {OPEN_MAX} file descriptors are
- 10921 currently open in the calling process, or no file descriptors greater than or
- 10922 equal to *arg* are available.
- 10923 [ENOLCK] The argument *cmd* is F_SETLK or F_SETLKW and satisfying the lock or unlock
- 10924 request would result in the number of locked regions in the system exceeding
- 10925 a system-imposed limit.
- 10926 [EOVERFLOW] One of the values to be returned cannot be represented correctly.
- 10927 [EOVERFLOW] The *cmd* argument is F_GETLK, F_SETLK, or F_SETLKW and the smallest or,
- 10928 if *l_len* is non-zero, the largest offset of any byte in the requested segment
- 10929 cannot be represented correctly in an object of type **off_t**.
- 10930 The *fcntl()* function may fail if:
- 10931 [EDEADLK] The *cmd* argument is F_SETLKW, the lock is blocked by a lock from another
- 10932 process, and putting the calling process to sleep to wait for that lock to
- 10933 become free would cause a deadlock.
- 10934 **EXAMPLES**
- 10935 None.
- 10936 **APPLICATION USAGE**
- 10937 None.
- 10938 **RATIONALE**
- 10939 The ellipsis in the SYNOPSIS is the syntax specified by the ISO C standard for a variable number
- 10940 of arguments. It is used because System V uses pointers for the implementation of file locking
- 10941 functions.
- 10942 The *arg* values to F_GETFD, F_SETFD, F_GETFL, and F_SETFL all represent flag values to allow
- 10943 for future growth. Applications using these functions should do a read-modify-write operation

10944 on them, rather than assuming that only the values defined by this volume of
10945 IEEE Std 1003.1-2001 are valid. It is a common error to forget this, particularly in the case of
10946 F_SETFD.

10947 This volume of IEEE Std 1003.1-2001 permits concurrent read and write access to file data using
10948 the *fcntl()* function; this is a change from the 1984 /usr/group standard and early proposals.
10949 Without concurrency controls, this feature may not be fully utilized without occasional loss of
10950 data.

10951 Data losses occur in several ways. One case occurs when several processes try to update the
10952 same record, without sequencing controls; several updates may occur in parallel and the last
10953 writer “wins”. Another case is a bit-tree or other internal list-based database that is undergoing
10954 reorganization. Without exclusive use to the tree segment by the updating process, other reading
10955 processes chance getting lost in the database when the index blocks are split, condensed,
10956 inserted, or deleted. While *fcntl()* is useful for many applications, it is not intended to be overly
10957 general and does not handle the bit-tree example well.

10958 This facility is only required for regular files because it is not appropriate for many devices such
10959 as terminals and network connections.

10960 Since *fcntl()* works with “any file descriptor associated with that file, however it is obtained”,
10961 the file descriptor may have been inherited through a *fork()* or *exec* operation and thus may
10962 affect a file that another process also has open.

10963 The use of the open file description to identify what to lock requires extra calls and presents
10964 problems if several processes are sharing an open file description, but there are too many
10965 implementations of the existing mechanism for this volume of IEEE Std 1003.1-2001 to use
10966 different specifications.

10967 Another consequence of this model is that closing any file descriptor for a given file (whether or
10968 not it is the same open file description that created the lock) causes the locks on that file to be
10969 relinquished for that process. Equivalently, any close for any file/process pair relinquishes the
10970 locks owned on that file for that process. But note that while an open file description may be
10971 shared through *fork()*, locks are not inherited through *fork()*. Yet locks may be inherited through
10972 one of the *exec* functions.

10973 The identification of a machine in a network environment is outside the scope of this volume of
10974 IEEE Std 1003.1-2001. Thus, an *L_sysid* member, such as found in System V, is not included in the
10975 locking structure.

10976 Changing of lock types can result in a previously locked region being split into smaller regions.

10977 Mandatory locking was a major feature of the 1984 /usr/group standard.

10978 For advisory file record locking to be effective, all processes that have access to a file must
10979 cooperate and use the advisory mechanism before doing I/O on the file. Enforcement-mode
10980 record locking is important when it cannot be assumed that all processes are cooperating. For
10981 example, if one user uses an editor to update a file at the same time that a second user executes
10982 another process that updates the same file and if only one of the two processes is using advisory
10983 locking, the processes are not cooperating. Enforcement-mode record locking would protect
10984 against accidental collisions.

10985 Secondly, advisory record locking requires a process using locking to bracket each I/O operation
10986 with lock (or test) and unlock operations. With enforcement-mode file and record locking, a
10987 process can lock the file once and unlock when all I/O operations have been completed.
10988 Enforcement-mode record locking provides a base that can be enhanced; for example, with
10989 sharable locks. That is, the mechanism could be enhanced to allow a process to lock a file so
10990 other processes could read it, but none of them could write it.

- 10991 Mandatory locks were omitted for several reasons:
- 10992 1. Mandatory lock setting was done by multiplexing the set-group-ID bit in most
10993 implementations; this was confusing, at best.
 - 10994 2. The relationship to file truncation as supported in 4.2 BSD was not well specified.
 - 10995 3. Any publicly readable file could be locked by anyone. Many historical implementations
10996 keep the password database in a publicly readable file. A malicious user could thus
10997 prohibit logins. Another possibility would be to hold open a long-distance telephone line.
 - 10998 4. Some demand-paged historical implementations offer memory mapped files, and
10999 enforcement cannot be done on that type of file.
- 11000 Since sleeping on a region is interrupted with any signal, *alarm()* may be used to provide a
11001 timeout facility in applications requiring it. This is useful in deadlock detection. Since
11002 implementation of full deadlock detection is not always feasible, the [EDEADLK] error was
11003 made optional.
- 11004 **FUTURE DIRECTIONS**
- 11005 None.
- 11006 **SEE ALSO**
- 11007 *alarm()*, *close()*, *exec*, *open()*, *sigaction()*, the Base Definitions volume of IEEE Std 1003.1-2001,
11008 `<fcntl.h>`, `<signal.h>`, `<unistd.h>`
- 11009 **CHANGE HISTORY**
- 11010 First released in Issue 1. Derived from Issue 1 of the SVID.
- 11011 **Issue 5**
- 11012 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
11013 Threads Extension.
- 11014 Large File Summit extensions are added.
- 11015 **Issue 6**
- 11016 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.
- 11017 The following new requirements on POSIX implementations derive from alignment with the
11018 Single UNIX Specification:
- 11019 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
11020 required for conforming implementations of previous POSIX specifications, it was not
11021 required for UNIX applications.
 - 11022 • In the DESCRIPTION, sentences describing behavior when *L_len* is negative are now
11023 mandated, and the description of unlock (F_UNLOCK) when *L_len* is non-negative is
11024 mandated.
 - 11025 • In the ERRORS section, the [EINVAL] error condition has the case mandated when the *cmd* is
11026 invalid, and two [E_OVERFLOW] error conditions are added.
- 11027 The F_GETOWN and F_SETOWN values are added for sockets.
- 11028 The following changes were made to align with the IEEE P1003.1a draft standard:
- 11029 • Clarification is added that the extent of the bytes locked is determined prior to the blocking
11030 action.
- 11031 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
11032 *fcntl()* results are unspecified for typed memory objects.

11033

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

11034 **NAME**11035 fcvt — convert a floating-point number to a string (**LEGACY**)11036 **SYNOPSIS**

11037 xSI #include <stdlib.h>

11038 char *fcvt(double *value*, int *ndigit*, int *restrict *decpt*,
11039 int *restrict *sign*);

11040

11041 **DESCRIPTION**11042 Refer to *ecvt()*.

11043 **NAME**11044 fdatasync — synchronize the data of a file (**REALTIME**)11045 **SYNOPSIS**

11046 SIO #include <unistd.h>

11047 int fdatasync(int *fil*des);

11048

11049 **DESCRIPTION**11050 The *fdatasync()* function shall force all currently queued I/O operations associated with the file
11051 indicated by file descriptor *fil*des to the synchronized I/O completion state.11052 The functionality shall be equivalent to *fsync()* with the symbol `_POSIX_SYNCHRONIZED_IO`
11053 defined, with the exception that all I/O operations shall be completed as defined for
11054 synchronized I/O data integrity completion.11055 **RETURN VALUE**11056 If successful, the *fdatasync()* function shall return the value 0; otherwise, the function shall return
11057 the value `-1` and set *errno* to indicate the error. If the *fdatasync()* function fails, outstanding I/O
11058 operations are not guaranteed to have been completed.11059 **ERRORS**11060 The *fdatasync()* function shall fail if:11061 [EBADF] The *fil*des argument is not a valid file descriptor open for writing.

11062 [EINVAL] This implementation does not support synchronized I/O for this file.

11063 In the event that any of the queued I/O operations fail, *fdatasync()* shall return the error
11064 conditions defined for *read()* and *write()*.11065 **EXAMPLES**

11066 None.

11067 **APPLICATION USAGE**

11068 None.

11069 **RATIONALE**

11070 None.

11071 **FUTURE DIRECTIONS**

11072 None.

11073 **SEE ALSO**11074 *aio_fsync()*, *fcntl()*, *fsync()*, *open()*, *read()*, *write()*, the Base Definitions volume of
11075 IEEE Std 1003.1-2001, <unistd.h>11076 **CHANGE HISTORY**

11077 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

11078 **Issue 6**11079 The [ENOSYS] error condition has been removed as stubs need not be provided if an
11080 implementation does not support the Synchronized Input and Output option.11081 The *fdatasync()* function is marked as part of the Synchronized Input and Output option.

11082 **NAME**11083 fdetach — detach a name from a STREAMS-based file descriptor (**STREAMS**)11084 **SYNOPSIS**

11085 XSR #include <stropts.h>

11086 int fdetach(const char *path);

11087

11088 **DESCRIPTION**

11089 The *fdetach()* function shall detach a STREAMS-based file from the file to which it was attached
 11090 by a previous call to *fattach()*. The *path* argument points to the pathname of the attached
 11091 STREAMS file. The process shall have appropriate privileges or be the owner of the file. A
 11092 successful call to *fdetach()* shall cause all pathnames that named the attached STREAMS file to
 11093 again name the file to which the STREAMS file was attached. All subsequent operations on *path*
 11094 shall operate on the underlying file and not on the STREAMS file.

11095 All open file descriptions established while the STREAMS file was attached to the file referenced
 11096 by *path* shall still refer to the STREAMS file after the *fdetach()* has taken effect.

11097 If there are no open file descriptors or other references to the STREAMS file, then a successful
 11098 call to *fdetach()* shall be equivalent to performing the last *close()* on the attached file.

11099 **RETURN VALUE**

11100 Upon successful completion, *fdetach()* shall return 0; otherwise, it shall return -1 and set *errno* to
 11101 indicate the error.

11102 **ERRORS**11103 The *fdetach()* function shall fail if:

11104 [EACCES] Search permission is denied on a component of the path prefix.

11105 [EINVAL] The *path* argument names a file that is not currently attached.

11106 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 11107 argument.

11108 [ENAMETOOLONG]

11109 The size of a pathname exceeds {PATH_MAX} or a pathname component is
 11110 longer than {NAME_MAX}.

11111 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

11112 [ENOTDIR] A component of the path prefix is not a directory.

11113 [EPERM] The effective user ID is not the owner of *path* and the process does not have
 11114 appropriate privileges.

11115 The *fdetach()* function may fail if:

11116 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 11117 resolution of the *path* argument.

11118 [ENAMETOOLONG]

11119 Pathname resolution of a symbolic link produced an intermediate result
 11120 whose length exceeds {PATH_MAX}.

11121 **EXAMPLES**11122 **Detaching a File**

11123 The following example detaches the STREAMS-based file **/tmp/named-STREAM** from the file to
11124 which it was attached by a previous, successful call to *fattach()*. Subsequent calls to open this
11125 file refer to the underlying file, not to the STREAMS file.

```
11126 #include <stropts.h>
11127 ...
11128     char *filename = "/tmp/named-STREAM";
11129     int ret;
11130     ret = fdetach(filename);
```

11131 **APPLICATION USAGE**

11132 None.

11133 **RATIONALE**

11134 None.

11135 **FUTURE DIRECTIONS**

11136 None.

11137 **SEE ALSO**

11138 *fattach()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stropts.h>

11139 **CHANGE HISTORY**

11140 First released in Issue 4, Version 2.

11141 **Issue 5**

11142 Moved from X/OPEN UNIX extension to BASE.

11143 **Issue 6**

11144 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

11145 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
11146 [ELOOP] error condition is added.

11147 **NAME**

11148 `fdim`, `fdimf`, `fdiml` — compute positive difference between two floating-point numbers

11149 **SYNOPSIS**

11150 `#include <math.h>`

11151 `double fdim(double x, double y);`

11152 `float fdimf(float x, float y);`

11153 `long double fdiml(long double x, long double y);`

11154 **DESCRIPTION**

11155 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
11156 conflict between the requirements described here and the ISO C standard is unintentional. This
11157 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11158 These functions shall determine the positive difference between their arguments. If x is greater
11159 than y , $x-y$ is returned. If x is less than or equal to y , $+0$ is returned.

11160 An application wishing to check for error situations should set *errno* to zero and call
11161 *feclearexcept*(*FE_ALL_EXCEPT*) before calling these functions. On return, if *errno* is non-zero or
11162 *fetestexcept*(*FE_INVALID* | *FE_DIVBYZERO* | *FE_OVERFLOW* | *FE_UNDERFLOW*) is non-
11163 zero, an error has occurred.

11164 **RETURN VALUE**

11165 Upon successful completion, these functions shall return the positive difference value.

11166 If $x-y$ is positive and overflows, a range error shall occur and *fdim*(), *fdimf*(), and *fdiml*() shall
11167 return the value of the macro *HUGE_VAL*, *HUGE_VALF*, and *HUGE_VALL*, respectively.

11168 **XSI** If $x-y$ is positive and underflows, a range error may occur, and either $(x-y)$ (if representable), or
11169 0.0 (if supported), or an implementation-defined value shall be returned.

11170 **MX** If x or y is NaN, a NaN shall be returned.

11171 **ERRORS**

11172 The *fdim*() function shall fail if:

11173 **Range Error** The result overflows.

11174 If the integer expression (*math_errhandling* & *MATH_ERRNO*) is non-zero,
11175 then *errno* shall be set to [ERANGE]. If the integer expression
11176 (*math_errhandling* & *MATH_ERREXCEPT*) is non-zero, then the overflow
11177 floating-point exception shall be raised.

11178 The *fdim*() function may fail if:

11179 **Range Error** The result underflows.

11180 If the integer expression (*math_errhandling* & *MATH_ERRNO*) is non-zero,
11181 then *errno* shall be set to [ERANGE]. If the integer expression
11182 (*math_errhandling* & *MATH_ERREXCEPT*) is non-zero, then the underflow
11183 floating-point exception shall be raised.

11184 **EXAMPLES**

11185 None.

11186 **APPLICATION USAGE**

11187 On implementations supporting IEEE Std 754-1985, $x-y$ cannot underflow, and hence the 0.0
11188 return value is shaded as an extension for implementations supporting the XSI extension rather
11189 than an MX extension.

11190 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
11191 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

11192 **RATIONALE**

11193 None.

11194 **FUTURE DIRECTIONS**

11195 None.

11196 **SEE ALSO**

11197 *feclearexcept()*, *fetestexcept()*, *fmax()*, *fmin()*, the Base Definitions volume of IEEE Std 1003.1-2001,
11198 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

11199 **CHANGE HISTORY**

11200 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11201 **NAME**

11202 fdopen — associate a stream with a file descriptor

11203 **SYNOPSIS**

11204 cx #include <stdio.h>

11205 FILE *fdopen(int *fil-des*, const char **mode*);

11206

11207 **DESCRIPTION**11208 The *fdopen()* function shall associate a stream with a file descriptor.11209 The *mode* argument is a character string having one of the following values:11210 *r* or *rb* Open a file for reading.11211 *w* or *wb* Open a file for writing.11212 *a* or *ab* Open a file for writing at end-of-file.11213 *r+* or *rb+* or *r+b* Open a file for update (reading and writing).11214 *w+* or *wb+* or *w+b* Open a file for update (reading and writing).11215 *a+* or *ab+* or *a+b* Open a file for update (reading and writing) at end-of-file.11216 The meaning of these flags is exactly as specified in *fopen()*, except that modes beginning with *w*
11217 shall not cause truncation of the file.11218 Additional values for the *mode* argument may be supported by an implementation.11219 The application shall ensure that the mode of the stream as expressed by the *mode* argument is
11220 allowed by the file access mode of the open file description to which *fil-des* refers. The file
11221 position indicator associated with the new stream is set to the position indicated by the file
11222 offset associated with the file descriptor.11223 The error and end-of-file indicators for the stream shall be cleared. The *fdopen()* function may
11224 cause the *st_atime* field of the underlying file to be marked for update.11225 SHM If *fil-des* refers to a shared memory object, the result of the *fdopen()* function is unspecified.11226 TYM If *fil-des* refers to a typed memory object, the result of the *fdopen()* function is unspecified.11227 The *fdopen()* function shall preserve the offset maximum previously set for the open file
11228 description corresponding to *fil-des*.11229 **RETURN VALUE**11230 Upon successful completion, *fdopen()* shall return a pointer to a stream; otherwise, a null pointer
11231 shall be returned and *errno* set to indicate the error.11232 **ERRORS**11233 The *fdopen()* function may fail if:11234 [EBADF] The *fil-des* argument is not a valid file descriptor.11235 [EINVAL] The *mode* argument is not a valid mode.

11236 [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

11237 [EMFILE] {STREAM_MAX} streams are currently open in the calling process.

11238 [ENOMEM] Insufficient space to allocate a buffer.

11239 **EXAMPLES**

11240 None.

11241 **APPLICATION USAGE**

11242 File descriptors are obtained from calls like *open()*, *dup()*, *creat()*, or *pipe()*, which open files but
 11243 do not return streams.

11244 **RATIONALE**

11245 The file descriptor may have been obtained from *open()*, *creat()*, *pipe()*, *dup()*, or *fcntl()*;
 11246 inherited through *fork()* or *exec*; or perhaps obtained by implementation-defined means, such as
 11247 the 4.3 BSD *socket()* call.

11248 The meanings of the *mode* arguments of *fdopen()* and *fopen()* differ. With *fdopen()*, open for write
 11249 (*w* or *w+*) does not truncate, and append (*a* or *a+*) cannot create for writing. The *mode* argument
 11250 formats that include *a b* are allowed for consistency with the ISO C standard function *fopen()*.
 11251 The *b* has no effect on the resulting stream. Although not explicitly required by this volume of
 11252 IEEE Std 1003.1-2001, a good implementation of append (*a*) mode would cause the O_APPEND
 11253 flag to be set.

11254 **FUTURE DIRECTIONS**

11255 None.

11256 **SEE ALSO**

11257 Section 2.5.1 (on page 35), *fclose()*, *fopen()*, *open()*, the Base Definitions volume of
 11258 IEEE Std 1003.1-2001, <stdio.h>

11259 **CHANGE HISTORY**

11260 First released in Issue 1. Derived from Issue 1 of the SVID.

11261 **Issue 5**

11262 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

11263 Large File Summit extensions are added.

11264 **Issue 6**

11265 The following new requirements on POSIX implementations derive from alignment with the
 11266 Single UNIX Specification:

- 11267 • In the DESCRIPTION, the use and setting of the *mode* argument are changed to include
 11268 binary streams.
- 11269 • In the DESCRIPTION, text is added for large file support to indicate setting of the offset
 11270 maximum in the open file description.
- 11271 • All errors identified in the ERRORS section are added.
- 11272 • In the DESCRIPTION, text is added that the *fdopen()* function may cause *st_atime* to be
 11273 updated.

11274 The following changes were made to align with the IEEE P1003.1a draft standard:

- 11275 • Clarification is added that it is the responsibility of the application to ensure that the mode is
 11276 compatible with the open file descriptor.

11277 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
 11278 *fdopen()* results are unspecified for typed memory objects.

11279 **NAME**

11280 `feclearexcept` — clear floating-point exception

11281 **SYNOPSIS**

11282 `#include <fenv.h>`

11283 `int feclearexcept(int excepts);`

11284 **DESCRIPTION**

11285 *cx* The functionality described on this reference page is aligned with the ISO C standard. Any
11286 conflict between the requirements described here and the ISO C standard is unintentional. This
11287 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11288 The `feclearexcept()` function shall attempt to clear the supported floating-point exceptions
11289 represented by *excepts*.

11290 **RETURN VALUE**

11291 If the argument is zero or if all the specified exceptions were successfully cleared, `feclearexcept()`
11292 shall return zero. Otherwise, it shall return a non-zero value.

11293 **ERRORS**

11294 No errors are defined.

11295 **EXAMPLES**

11296 None.

11297 **APPLICATION USAGE**

11298 None.

11299 **RATIONALE**

11300 None.

11301 **FUTURE DIRECTIONS**

11302 None.

11303 **SEE ALSO**

11304 `fegetexceptflag()`, `feraiseexcept()`, `fesetexceptflag()`, `fetestexcept()`, the Base Definitions volume of
11305 IEEE Std 1003.1-2001, `<fenv.h>`

11306 **CHANGE HISTORY**

11307 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11308 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

11309 **NAME**

11310 fegetenv, fesetenv — get and set current floating-point environment

11311 **SYNOPSIS**

11312 #include <fenv.h>

11313 int fegetenv(fenv_t *envp);

11314 int fesetenv(const fenv_t *envp);

11315 **DESCRIPTION**

11316 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
11317 conflict between the requirements described here and the ISO C standard is unintentional. This
11318 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11319 The *fegetenv()* function shall attempt to store the current floating-point environment in the object
11320 pointed to by *envp*.

11321 The *fesetenv()* function shall attempt to establish the floating-point environment represented by
11322 the object pointed to by *envp*. The argument *envp* shall point to an object set by a call to
11323 *fegetenv()* or *feholdexcept()*, or equal a floating-point environment macro. The *fesetenv()* function
11324 does not raise floating-point exceptions, but only installs the state of the floating-point status
11325 flags represented through its argument.

11326 **RETURN VALUE**

11327 If the representation was successfully stored, *fegetenv()* shall return zero. Otherwise, it shall
11328 return a non-zero value. If the environment was successfully established, *fesetenv()* shall return
11329 zero. Otherwise, it shall return a non-zero value.

11330 **ERRORS**

11331 No errors are defined.

11332 **EXAMPLES**

11333 None.

11334 **APPLICATION USAGE**

11335 None.

11336 **RATIONALE**

11337 None.

11338 **FUTURE DIRECTIONS**

11339 None.

11340 **SEE ALSO**

11341 *feholdexcept()*, *feupdateenv()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**fenv.h**>

11342 **CHANGE HISTORY**

11343 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11344 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

11345 **NAME**

11346 fegetexceptflag, fesetexceptflag — get and set floating-point status flags

11347 **SYNOPSIS**

11348 #include <fenv.h>

11349 int fegetexceptflag(fexcept_t *flagp, int excepts);

11350 int fesetexceptflag(const fexcept_t *flagp, int excepts);

11351 **DESCRIPTION**

11352 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
11353 conflict between the requirements described here and the ISO C standard is unintentional. This
11354 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11355 The *fegetexceptflag()* function shall attempt to store an implementation-defined representation of
11356 the states of the floating-point status flags indicated by the argument *excepts* in the object
11357 pointed to by the argument *flagp*.

11358 The *fesetexceptflag()* function shall attempt to set the floating-point status flags indicated by the
11359 argument *excepts* to the states stored in the object pointed to by *flagp*. The value pointed to by
11360 *flagp* shall have been set by a previous call to *fegetexceptflag()* whose second argument
11361 represented at least those floating-point exceptions represented by the argument *excepts*. This
11362 function does not raise floating-point exceptions, but only sets the state of the flags.

11363 **RETURN VALUE**

11364 If the representation was successfully stored, *fegetexceptflag()* shall return zero. Otherwise, it
11365 shall return a non-zero value. If the *excepts* argument is zero or if all the specified exceptions
11366 were successfully set, *fesetexceptflag()* shall return zero. Otherwise, it shall return a non-zero
11367 value.

11368 **ERRORS**

11369 No errors are defined.

11370 **EXAMPLES**

11371 None.

11372 **APPLICATION USAGE**

11373 None.

11374 **RATIONALE**

11375 None.

11376 **FUTURE DIRECTIONS**

11377 None.

11378 **SEE ALSO**

11379 *feclearexcept()*, *feraiseexcept()*, *fetestexcept()*, the Base Definitions volume of IEEE Std 1003.1-2001,
11380 <fenv.h>

11381 **CHANGE HISTORY**

11382 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11383 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

11384 **NAME**

11385 fegetround, fesetround — get and set current rounding direction

11386 **SYNOPSIS**

```
11387 #include <fenv.h>
11388 int fegetround(void);
11389 int fesetround(int round);
```

11390 **DESCRIPTION**

11391 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 11392 conflict between the requirements described here and the ISO C standard is unintentional. This
 11393 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11394 The *fegetround()* function shall get the current rounding direction.

11395 The *fesetround()* function shall establish the rounding direction represented by its argument
 11396 *round*. If the argument is not equal to the value of a rounding direction macro, the rounding
 11397 direction is not changed.

11398 **RETURN VALUE**

11399 The *fegetround()* function shall return the value of the rounding direction macro representing the
 11400 current rounding direction or a negative value if there is no such rounding direction macro or
 11401 the current rounding direction is not determinable.

11402 The *fesetround()* function shall return a zero value if and only if the requested rounding direction
 11403 was established.

11404 **ERRORS**

11405 No errors are defined.

11406 **EXAMPLES**

11407 The following example saves, sets, and restores the rounding direction, reporting an error and
 11408 aborting if setting the rounding direction fails:

```
11409 #include <fenv.h>
11410 #include <assert.h>
11411 void f(int round_dir)
11412 {
11413     #pragma STDC FENV_ACCESS ON
11414     int save_round;
11415     int setround_ok;
11416     save_round = fegetround();
11417     setround_ok = fesetround(round_dir);
11418     assert(setround_ok == 0);
11419     /* ... */
11420     fesetround(save_round);
11421     /* ... */
11422 }
```

11423 **APPLICATION USAGE**

11424 None.

11425 **RATIONALE**

11426 None.

11427 **FUTURE DIRECTIONS**

11428 None.

11429 **SEE ALSO**

11430 The Base Definitions volume of IEEE Std 1003.1-2001, <fenv.h>

11431 **CHANGE HISTORY**

11432 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

11433 ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

11434 **NAME**

11435 feholdexcept — save current floating-point environment

11436 **SYNOPSIS**

11437 #include <fenv.h>

11438 int feholdexcept(fenv_t *envp);

11439 **DESCRIPTION**

11440 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
11441 conflict between the requirements described here and the ISO C standard is unintentional. This
11442 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11443 The *feholdexcept()* function shall save the current floating-point environment in the object
11444 pointed to by *envp*, clear the floating-point status flags, and then install a non-stop (continue on
11445 floating-point exceptions) mode, if available, for all floating-point exceptions.

11446 **RETURN VALUE**

11447 The *feholdexcept()* function shall return zero if and only if non-stop floating-point exception
11448 handling was successfully installed.

11449 **ERRORS**

11450 No errors are defined.

11451 **EXAMPLES**

11452 None.

11453 **APPLICATION USAGE**

11454 None.

11455 **RATIONALE**

11456 The *feholdexcept()* function should be effective on typical IEC 60559:1989 standard
11457 implementations which have the default non-stop mode and at least one other mode for trap
11458 handling or aborting. If the implementation provides only the non-stop mode, then installing the
11459 non-stop mode is trivial.

11460 **FUTURE DIRECTIONS**

11461 None.

11462 **SEE ALSO**

11463 *fegetenv()*, *fesetenv()*, *feupdateenv()*, the Base Definitions volume of IEEE Std 1003.1-2001,
11464 <fenv.h>

11465 **CHANGE HISTORY**

11466 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11467 **NAME**

11468 feof — test end-of-file indicator on a stream

11469 **SYNOPSIS**

11470 #include <stdio.h>

11471 int feof(FILE *stream);

11472 **DESCRIPTION**

11473 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
11474 conflict between the requirements described here and the ISO C standard is unintentional. This
11475 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11476 The *feof()* function shall test the end-of-file indicator for the stream pointed to by *stream*.11477 **RETURN VALUE**11478 The *feof()* function shall return non-zero if and only if the end-of-file indicator is set for *stream*.11479 **ERRORS**

11480 No errors are defined.

11481 **EXAMPLES**

11482 None.

11483 **APPLICATION USAGE**

11484 None.

11485 **RATIONALE**

11486 None.

11487 **FUTURE DIRECTIONS**

11488 None.

11489 **SEE ALSO**11490 *clearerr()*, *ferror()*, *fopen()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>11491 **CHANGE HISTORY**

11492 First released in Issue 1. Derived from Issue 1 of the SVID.

11493 **NAME**

11494 `feraiseexcept` — raise floating-point exception

11495 **SYNOPSIS**

11496 `#include <fenv.h>`

11497 `int feraiseexcept(int excepts);`

11498 **DESCRIPTION**

11499 *cx* The functionality described on this reference page is aligned with the ISO C standard. Any
11500 conflict between the requirements described here and the ISO C standard is unintentional. This
11501 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11502 The `feraiseexcept()` function shall attempt to raise the supported floating-point exceptions
11503 represented by the argument *excepts*. The order in which these floating-point exceptions are
11504 raised is unspecified. Whether the `feraiseexcept()` function additionally raises the inexact
11505 floating-point exception whenever it raises the overflow or underflow floating-point exception is
11506 implementation-defined.

11507 **RETURN VALUE**

11508 If the argument is zero or if all the specified exceptions were successfully raised, `feraiseexcept()`
11509 shall return zero. Otherwise, it shall return a non-zero value.

11510 **ERRORS**

11511 No errors are defined.

11512 **EXAMPLES**

11513 None.

11514 **APPLICATION USAGE**

11515 The effect is intended to be similar to that of floating-point exceptions raised by arithmetic
11516 operations. Hence, enabled traps for floating-point exceptions raised by this function are taken.

11517 **RATIONALE**

11518 Raising overflow or underflow is allowed to also raise inexact because on some architectures the
11519 only practical way to raise an exception is to execute an instruction that has the exception as a
11520 side effect. The function is not restricted to accept only valid coincident expressions for atomic
11521 operations, so the function can be used to raise exceptions accrued over several operations.

11522 **FUTURE DIRECTIONS**

11523 None.

11524 **SEE ALSO**

11525 `feclearexcept()`, `fegetexceptflag()`, `fesetexceptflag()`, `fetestexcept()`, the Base Definitions volume of
11526 IEEE Std 1003.1-2001, `<fenv.h>`

11527 **CHANGE HISTORY**

11528 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11529 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

11530 **NAME**11531 `ferror` — test error indicator on a stream11532 **SYNOPSIS**11533 `#include <stdio.h>`11534 `int ferror(FILE *stream);`11535 **DESCRIPTION**

11536 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
11537 conflict between the requirements described here and the ISO C standard is unintentional. This
11538 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11539 The `ferror()` function shall test the error indicator for the stream pointed to by *stream*.11540 **RETURN VALUE**11541 The `ferror()` function shall return non-zero if and only if the error indicator is set for *stream*.11542 **ERRORS**

11543 No errors are defined.

11544 **EXAMPLES**

11545 None.

11546 **APPLICATION USAGE**

11547 None.

11548 **RATIONALE**

11549 None.

11550 **FUTURE DIRECTIONS**

11551 None.

11552 **SEE ALSO**11553 `clearerr()`, `feof()`, `fopen()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<stdio.h>`11554 **CHANGE HISTORY**

11555 First released in Issue 1. Derived from Issue 1 of the SVID.

11556 **NAME**

11557 fesetenv — set current floating-point environment

11558 **SYNOPSIS**

11559 #include <fenv.h>

11560 int fesetenv(const fenv_t *envp);

11561 **DESCRIPTION**11562 Refer to *fegetenv()*.

11563 **NAME**

11564 fesetexceptflag — set floating-point status flags

11565 **SYNOPSIS**

11566 #include <fenv.h>

11567 int fesetexceptflag(const fexcept_t *flagp, int excepts);

11568 **DESCRIPTION**

11569 Refer to *fegetexceptflag()*.

11570 **NAME**

11571 fesetround — set current rounding direction

11572 **SYNOPSIS**

11573 #include <fenv.h>

11574 int fesetround(int *round*);

11575 **DESCRIPTION**

11576 Refer to *fegetround()*.

11577 **NAME**

11578 fetestexcept — test floating-point exception flags

11579 **SYNOPSIS**

11580 #include <fenv.h>

11581 int fetestexcept(int *excepts*);11582 **DESCRIPTION**

11583 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 11584 conflict between the requirements described here and the ISO C standard is unintentional. This
 11585 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11586 The *fetestexcept()* function shall determine which of a specified subset of the floating-point
 11587 exception flags are currently set. The *excepts* argument specifies the floating-point status flags to
 11588 be queried.

11589 **RETURN VALUE**

11590 The *fetestexcept()* function shall return the value of the bitwise-inclusive OR of the floating-point
 11591 exception macros corresponding to the currently set floating-point exceptions included in
 11592 *excepts*.

11593 **ERRORS**

11594 No errors are defined.

11595 **EXAMPLES**

11596 The following example calls function *f()* if an invalid exception is set, and then function *g()* if an
 11597 overflow exception is set:

```

11598       #include <fenv.h>
11599       /* ... */
11600       {
11601           #pragma STDC FENV_ACCESS ON
11602           int set_excepts;
11603           feclearexcept(FE_INVALID | FE_OVERFLOW);
11604           // maybe raise exceptions
11605           set_excepts = fetestexcept(FE_INVALID | FE_OVERFLOW);
11606           if (set_excepts & FE_INVALID) f();
11607           if (set_excepts & FE_OVERFLOW) g();
11608           /* ... */
11609       }

```

11610 **APPLICATION USAGE**

11611 None.

11612 **RATIONALE**

11613 None.

11614 **FUTURE DIRECTIONS**

11615 None.

11616 **SEE ALSO**

11617 *feclearexcept()*, *fegetexceptflag()*, *feraiseexcept()*, the Base Definitions volume of
 11618 IEEE Std 1003.1-2001, <fenv.h>

11619 **CHANGE HISTORY**

11620 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11621 **NAME**

11622 feupdateenv — update floating-point environment

11623 **SYNOPSIS**

11624 #include <fenv.h>

11625 int feupdateenv(const fenv_t *envp);

11626 **DESCRIPTION**

11627 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 11628 conflict between the requirements described here and the ISO C standard is unintentional. This
 11629 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11630 The *feupdateenv()* function shall attempt to save the currently raised floating-point exceptions in
 11631 its automatic storage, attempt to install the floating-point environment represented by the object
 11632 pointed to by *envp*, and then attempt to raise the saved floating-point exceptions. The argument
 11633 *envp* shall point to an object set by a call to *feholdexcept()* or *fegetenv()*, or equal a floating-point
 11634 environment macro.

11635 **RETURN VALUE**

11636 The *feupdateenv()* function shall return a zero value if and only if all the required actions were
 11637 successfully carried out.

11638 **ERRORS**

11639 No errors are defined.

11640 **EXAMPLES**

11641 The following example shows sample code to hide spurious underflow floating-point
 11642 exceptions:

```

11643 #include <fenv.h>
11644 double f(double x)
11645 {
11646     #pragma STDC FENV_ACCESS ON
11647     double result;
11648     fenv_t save_env;
11649     feholdexcept(&save_env);
11650     // compute result
11651     if (/* test spurious underflow */)
11652         feclearexcept(FE_UNDERFLOW);
11653     feupdateenv(&save_env);
11654     return result;
11655 }
```

11656 **APPLICATION USAGE**

11657 None.

11658 **RATIONALE**

11659 None.

11660 **FUTURE DIRECTIONS**

11661 None.

11662 **SEE ALSO**11663 *fegetenv()*, *feholdexcept()*, the Base Definitions volume of IEEE Std 1003.1-2001, <fenv.h>

11664 **CHANGE HISTORY**

- 11665 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.
- 11666 ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

11667 **NAME**

11668 fflush — flush a stream

11669 **SYNOPSIS**

11670 #include <stdio.h>

11671 int fflush(FILE *stream);

11672 **DESCRIPTION**

11673 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 11674 conflict between the requirements described here and the ISO C standard is unintentional. This
 11675 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11676 If *stream* points to an output stream or an update stream in which the most recent operation was
 11677 CX not input, *fflush()* shall cause any unwritten data for that stream to be written to the file, and the
 11678 *st_ctime* and *st_mtime* fields of the underlying file shall be marked for update.

11679 If *stream* is a null pointer, *fflush()* shall perform this flushing action on all streams for which the
 11680 behavior is defined above.

11681 **RETURN VALUE**

11682 Upon successful completion, *fflush()* shall return 0; otherwise, it shall set the error indicator for
 11683 CX the stream, return EOF, and set *errno* to indicate the error.

11684 **ERRORS**11685 The *fflush()* function shall fail if:

11686 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
 11687 process would be delayed in the write operation.

11688 CX [EBADF] The file descriptor underlying *stream* is not valid.

11689 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

11690 XSI [EFBIG] An attempt was made to write a file that exceeds the process' file size limit.

11691 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
 11692 offset maximum associated with the corresponding stream.

11693 CX [EINTR] The *fflush()* function was interrupted by a signal.

11694 CX [EIO] The process is a member of a background process group attempting to write
 11695 to its controlling terminal, TOSTOP is set, the process is neither ignoring nor
 11696 blocking SIGTTOU, and the process group of the process is orphaned. This
 11697 error may also be returned under implementation-defined conditions.

11698 CX [ENOSPC] There was no free space remaining on the device containing the file.

11699 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
 11700 any process. A SIGPIPE signal shall also be sent to the thread.

11701 The *fflush()* function may fail if:

11702 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
 11703 capabilities of the device.

11704 **EXAMPLES**11705 **Sending Prompts to Standard Output**

11706 The following example uses *printf()* calls to print a series of prompts for information the user
 11707 must enter from standard input. The *fflush()* calls force the output to standard output. The
 11708 *fflush()* function is used because standard output is usually buffered and the prompt may not
 11709 immediately be printed on the output or terminal. The *gets()* calls read strings from standard
 11710 input and place the results in variables, for use later in the program.

```
11711 #include <stdio.h>
11712 ...
11713 char user[100];
11714 char oldpasswd[100];
11715 char newpasswd[100];
11716 ...
11717 printf("User name: ");
11718 fflush(stdout);
11719 gets(user);

11720 printf("Old password: ");
11721 fflush(stdout);
11722 gets(oldpasswd);

11723 printf("New password: ");
11724 fflush(stdout);
11725 gets(newpasswd);
11726 ...
```

11727 **APPLICATION USAGE**

11728 None.

11729 **RATIONALE**

11730 Data buffered by the system may make determining the validity of the position of the current
 11731 file descriptor impractical. Thus, enforcing the repositioning of the file descriptor after *fflush()*
 11732 on streams open for *read()* is not mandated by IEEE Std 1003.1-2001.

11733 **FUTURE DIRECTIONS**

11734 None.

11735 **SEE ALSO**

11736 *getrlimit()*, *ulimit()*, the Base Definitions volume of IEEE Std 1003.1-2001, *<stdio.h>*

11737 **CHANGE HISTORY**

11738 First released in Issue 1. Derived from Issue 1 of the SVID.

11739 **Issue 5**

11740 Large File Summit extensions are added.

11741 **Issue 6**

11742 Extensions beyond the ISO C standard are marked.

11743 The following new requirements on POSIX implementations derive from alignment with the
 11744 Single UNIX Specification:

- 11745 • The [EFBIG] error is added as part of the large file support extensions.
- 11746 • The [ENXIO] optional error condition is added.

11747
11748

The RETURN VALUE section is updated to note that the error indicator shall be set for the stream. This is for alignment with the ISO/IEC 9899:1999 standard.

11749 **NAME**

11750 ffs — find first set bit

11751 **SYNOPSIS**

11752 xSI #include <strings.h>

11753 int ffs(int i);

11754

11755 **DESCRIPTION**11756 The *ffs()* function shall find the first bit set (beginning with the least significant bit) in *i*, and
11757 return the index of that bit. Bits are numbered starting at one (the least significant bit).11758 **RETURN VALUE**11759 The *ffs()* function shall return the index of the first bit set. If *i* is 0, then *ffs()* shall return 0.11760 **ERRORS**

11761 No errors are defined.

11762 **EXAMPLES**

11763 None.

11764 **APPLICATION USAGE**

11765 None.

11766 **RATIONALE**

11767 None.

11768 **FUTURE DIRECTIONS**

11769 None.

11770 **SEE ALSO**11771 The Base Definitions volume of IEEE Std 1003.1-2001, <**strings.h**>11772 **CHANGE HISTORY**

11773 First released in Issue 4, Version 2.

11774 **Issue 5**

11775 Moved from X/OPEN UNIX extension to BASE.

11776 NAME

11777 fgetc — get a byte from a stream

11778 SYNOPSIS

11779 #include <stdio.h>

11780 int fgetc(FILE *stream);

11781 DESCRIPTION

11782 CX The functionality described on this reference page is aligned with the ISO C standard. Any
11783 conflict between the requirements described here and the ISO C standard is unintentional. This
11784 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11785 If the end-of-file indicator for the input stream pointed to by *stream* is not set and a next byte is
11786 present, the *fgetc()* function shall obtain the next byte as an **unsigned char** converted to an **int**,
11787 from the input stream pointed to by *stream*, and advance the associated file position indicator for
11788 the stream (if defined). Since *fgetc()* operates on bytes, reading a character consisting of multiple
11789 bytes (or “a multi-byte character”) may require multiple calls to *fgetc()*.

11790 CX The *fgetc()* function may mark the *st_atime* field of the file associated with *stream* for update. The
11791 *st_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,
11792 *fgetwc()*, *fgetws()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()* using *stream* that returns
11793 data not supplied by a prior call to *ungetc()* or *ungetwc()*.

11794 RETURN VALUE

11795 Upon successful completion, *fgetc()* shall return the next byte from the input stream pointed to
11796 by *stream*. If the end-of-file indicator for the stream is set, or if the stream is at end-of-file, the
11797 end-of-file indicator for the stream shall be set and *fgetc()* shall return EOF. If a read error occurs,
11798 CX the error indicator for the stream shall be set, *fgetc()* shall return EOF, and shall set *errno* to
11799 indicate the error.

11800 ERRORS

11801 The *fgetc()* function shall fail if data needs to be read and:

11802 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
11803 process would be delayed in the *fgetc()* operation.

11804 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for
11805 reading.

11806 CX [EINTR] The read operation was terminated due to the receipt of a signal, and no data
11807 was transferred.

11808 CX [EIO] A physical I/O error has occurred, or the process is in a background process
11809 group attempting to read from its controlling terminal, and either the process
11810 is ignoring or blocking the SIGTTIN signal or the process group is orphaned.
11811 This error may also be generated for implementation-defined reasons.

11812 CX [E_OVERFLOW] The file is a regular file and an attempt was made to read at or beyond the
11813 offset maximum associated with the corresponding stream.

11814 The *fgetc()* function may fail if:

11815 CX [ENOMEM] Insufficient storage space is available.

11816 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
11817 capabilities of the device.

11818 **EXAMPLES**

11819 None.

11820 **APPLICATION USAGE**

11821 If the integer value returned by *fgetc()* is stored into a variable of type **char** and then compared
11822 against the integer constant EOF, the comparison may never succeed, because sign-extension of
11823 a variable of type **char** on widening to integer is implementation-defined.

11824 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an
11825 end-of-file condition.

11826 **RATIONALE**

11827 None.

11828 **FUTURE DIRECTIONS**

11829 None.

11830 **SEE ALSO**

11831 *feof()*, *ferror()*, *fopen()*, *getchar()*, *getc()*, the Base Definitions volume of IEEE Std 1003.1-2001,
11832 <stdio.h>

11833 **CHANGE HISTORY**

11834 First released in Issue 1. Derived from Issue 1 of the SVID.

11835 **Issue 5**

11836 Large File Summit extensions are added.

11837 **Issue 6**

11838 Extensions beyond the ISO C standard are marked.

11839 The following new requirements on POSIX implementations derive from alignment with the
11840 Single UNIX Specification:

- 11841 • The [EIO] and [EOVERFLOW] mandatory error conditions are added.
- 11842 • The [ENOMEM] and [ENXIO] optional error conditions are added.

11843 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 11844 • The DESCRIPTION is updated to clarify the behavior when the end-of-file indicator for the
11845 input stream is not set.
- 11846 • The RETURN VALUE section is updated to note that the error indicator shall be set for the
11847 stream.

11848 **NAME**

11849 fgetpos — get current file position information

11850 **SYNOPSIS**

11851 #include <stdio.h>

11852 int fgetpos(FILE *restrict *stream*, fpos_t *restrict *pos*);11853 **DESCRIPTION**

11854 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 11855 conflict between the requirements described here and the ISO C standard is unintentional. This
 11856 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11857 The *fgetpos()* function shall store the current values of the parse state (if any) and file position
 11858 indicator for the stream pointed to by *stream* in the object pointed to by *pos*. The value stored
 11859 contains unspecified information usable by *fsetpos()* for repositioning the stream to its position
 11860 at the time of the call to *fgetpos()*.

11861 **RETURN VALUE**

11862 Upon successful completion, *fgetpos()* shall return 0; otherwise, it shall return a non-zero value
 11863 and set *errno* to indicate the error.

11864 **ERRORS**11865 The *fgetpos()* function shall fail if:

11866 CX [EOVERFLOW] The current value of the file position cannot be represented correctly in an
 11867 object of type **fpos_t**.

11868 The *fgetpos()* function may fail if:

11869 CX [EBADF] The file descriptor underlying *stream* is not valid.

11870 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe, FIFO, or socket.

11871

11872 **EXAMPLES**

11873 None.

11874 **APPLICATION USAGE**

11875 None.

11876 **RATIONALE**

11877 None.

11878 **FUTURE DIRECTIONS**

11879 None.

11880 **SEE ALSO**11881 *fopen()*, *ftell()*, *rewind()*, *ungetc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>11882 **CHANGE HISTORY**

11883 First released in Issue 4. Derived from the ISO C standard.

11884 **Issue 5**

11885 Large File Summit extensions are added.

11886 **Issue 6**

11887 Extensions beyond the ISO C standard are marked.

11888 The following new requirements on POSIX implementations derive from alignment with the
 11889 Single UNIX Specification:

- 11890 • The [EBADF] and [ESPIPE] optional error conditions are added.
- 11891 An additional [ESPIPE] error condition is added for sockets.
- 11892 The prototype for *fgetpos()* is changed for alignment with the ISO/IEC 9899:1999 standard.

11893 **NAME**

11894 fgets — get a string from a stream

11895 **SYNOPSIS**

11896 #include <stdio.h>

11897 char *fgets(char *restrict *s*, int *n*, FILE *restrict *stream*);11898 **DESCRIPTION**

11899 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 11900 conflict between the requirements described here and the ISO C standard is unintentional. This
 11901 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11902 The *fgets()* function shall read bytes from *stream* into the array pointed to by *s*, until *n*−1 bytes
 11903 are read, or a <newline> is read and transferred to *s*, or an end-of-file condition is encountered.
 11904 The string is then terminated with a null byte.

11905 cx The *fgets()* function may mark the *st_atime* field of the file associated with *stream* for update. The
 11906 *st_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,
 11907 *fgetwc()*, *fgetws()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()* using *stream* that returns
 11908 data not supplied by a prior call to *ungetc()* or *ungetwc()*.

11909 **RETURN VALUE**

11910 Upon successful completion, *fgets()* shall return *s*. If the stream is at end-of-file, the end-of-file
 11911 indicator for the stream shall be set and *fgets()* shall return a null pointer. If a read error occurs,
 11912 cx the error indicator for the stream shall be set, *fgets()* shall return a null pointer, and shall set
 11913 *errno* to indicate the error.

11914 **ERRORS**11915 Refer to *fgetc()*.11916 **EXAMPLES**11917 **Reading Input**

11918 The following example uses *fgets()* to read each line of input. {LINE_MAX}, which defines the
 11919 maximum size of the input line, is defined in the <limits.h> header.

```
11920         #include <stdio.h>
11921         ...
11922         char line[LINE_MAX];
11923         ...
11924         while (fgets(line, LINE_MAX, fp) != NULL) {
11925         ...
11926         }
11927         ...
```

11928 **APPLICATION USAGE**

11929 None.

11930 **RATIONALE**

11931 None.

11932 **FUTURE DIRECTIONS**

11933 None.

11934 **SEE ALSO**

11935 *fopen()*, *fread()*, *gets()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

11936 **CHANGE HISTORY**

11937 First released in Issue 1. Derived from Issue 1 of the SVID.

11938 **Issue 6**

11939 Extensions beyond the ISO C standard are marked.

11940 The prototype for *fgets()* is changed for alignment with the ISO/IEC 9899:1999 standard.

11941 NAME

11942 fgetwc — get a wide-character code from a stream

11943 SYNOPSIS

11944 #include <stdio.h>

11945 #include <wchar.h>

11946 wint_t fgetwc(FILE *stream);

11947 DESCRIPTION

11948 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 11949 conflict between the requirements described here and the ISO C standard is unintentional. This
 11950 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

11951 The *fgetwc()* function shall obtain the next character (if present) from the input stream pointed to
 11952 by *stream*, convert that to the corresponding wide-character code, and advance the associated
 11953 file position indicator for the stream (if defined).

11954 If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

11955 CX The *fgetwc()* function may mark the *st_atime* field of the file associated with *stream* for update.
 11956 The *st_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,
 11957 *fgetwc()*, *fgetws()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()* using *stream* that returns
 11958 data not supplied by a prior call to *ungetc()* or *ungetwc()*.

11959 RETURN VALUE

11960 Upon successful completion, the *fgetwc()* function shall return the wide-character code of the
 11961 character read from the input stream pointed to by *stream* converted to a type **wint_t**. If the
 11962 stream is at end-of-file, the end-of-file indicator for the stream shall be set and *fgetwc()* shall
 11963 return WEOF. If a read error occurs, the error indicator for the stream shall be set, *fgetwc()* shall
 11964 CX return WEOF, and shall set *errno* to indicate the error. If an encoding error occurs, the error
 11965 indicator for the stream shall be set, *fgetwc()* shall return WEOF, and shall set *errno* to indicate
 11966 the error.

11967 ERRORS

11968 The *fgetwc()* function shall fail if data needs to be read and:

11969 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the
 11970 process would be delayed in the *fgetwc()* operation.

11971 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for
 11972 reading.

11973 [EILSEQ] The data obtained from the input stream does not form a valid character.

11974 CX [EINTR] The read operation was terminated due to the receipt of a signal, and no data
 11975 was transferred.

11976 CX [EIO] A physical I/O error has occurred, or the process is in a background process
 11977 group attempting to read from its controlling terminal, and either the process
 11978 is ignoring or blocking the SIGTTIN signal or the process group is orphaned.
 11979 This error may also be generated for implementation-defined reasons.

11980 CX [EOVERFLOW] The file is a regular file and an attempt was made to read at or beyond the
 11981 offset maximum associated with the corresponding stream.

11982 The *fgetwc()* function may fail if:

11983 CX [ENOMEM] Insufficient storage space is available.

11984 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
11985 capabilities of the device.

11986 EXAMPLES

11987 None.

11988 APPLICATION USAGE

11989 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an
11990 end-of-file condition.

11991 RATIONALE

11992 None.

11993 FUTURE DIRECTIONS

11994 None.

11995 SEE ALSO

11996 *feof()*, *ferror()*, *fopen()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<stdio.h>`,
11997 `<wchar.h>`

11998 CHANGE HISTORY

11999 First released in Issue 4. Derived from the MSE working draft.

12000 Issue 5

12001 The Optional Header (OH) marking is removed from `<stdio.h>`.

12002 Large File Summit extensions are added.

12003 Issue 6

12004 Extensions beyond the ISO C standard are marked.

12005 The following new requirements on POSIX implementations derive from alignment with the
12006 Single UNIX Specification:

- 12007 • The [EIO] and [EOVERFLOW] mandatory error conditions are added.
- 12008 • The [ENOMEM] and [ENXIO] optional error conditions are added.

12009 **NAME**

12010 fgetws — get a wide-character string from a stream

12011 **SYNOPSIS**

12012 #include <stdio.h>

12013 #include <wchar.h>

12014 wchar_t *fgetws(wchar_t *restrict ws, int n,

12015 FILE *restrict stream);

12016 **DESCRIPTION**

12017 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 12018 conflict between the requirements described here and the ISO C standard is unintentional. This
 12019 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

12020 The *fgetws()* function shall read characters from the *stream*, convert these to the corresponding
 12021 wide-character codes, place them in the **wchar_t** array pointed to by *ws*, until *n*−1 characters are
 12022 read, or a <newline> is read, converted, and transferred to *ws*, or an end-of-file condition is
 12023 encountered. The wide-character string, *ws*, shall then be terminated with a null wide-character
 12024 code.

12025 If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

12026 **CX** The *fgetws()* function may mark the *st_atime* field of the file associated with *stream* for update.
 12027 The *st_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,
 12028 *fgetwc()*, *fgetws()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()* using *stream* that returns
 12029 data not supplied by a prior call to *ungetc()* or *ungetwc()*.

12030 **RETURN VALUE**

12031 Upon successful completion, *fgetws()* shall return *ws*. If the stream is at end-of-file, the end-of-
 12032 file indicator for the stream shall be set and *fgetws()* shall return a null pointer. If a read error
 12033 **CX** occurs, the error indicator for the stream shall be set, *fgetws()* shall return a null pointer, and
 12034 shall set *errno* to indicate the error.

12035 **ERRORS**12036 Refer to *fgetwc()*.12037 **EXAMPLES**

12038 None.

12039 **APPLICATION USAGE**

12040 None.

12041 **RATIONALE**

12042 None.

12043 **FUTURE DIRECTIONS**

12044 None.

12045 **SEE ALSO**12046 *fopen()*, *fread()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>, <wchar.h>12047 **CHANGE HISTORY**

12048 First released in Issue 4. Derived from the MSE working draft.

12049 **Issue 5**

12050 The Optional Header (OH) marking is removed from <stdio.h>.

12051 **Issue 6**

12052 Extensions beyond the ISO C standard are marked.

12053 The prototype for *fgetws()* is changed for alignment with the ISO/IEC 9899:1999 standard.

12054 **NAME**

12055 `fileno` — map a stream pointer to a file descriptor

12056 **SYNOPSIS**

```
12057 cx        #include <stdio.h>
```

```
12058        int fileno(FILE *stream);
```

12059

12060 **DESCRIPTION**

12061 The `fileno()` function shall return the integer file descriptor associated with the stream pointed to
12062 by `stream`.

12063 **RETURN VALUE**

12064 Upon successful completion, `fileno()` shall return the integer value of the file descriptor
12065 associated with `stream`. Otherwise, the value `-1` shall be returned and `errno` set to indicate the
12066 error.

12067 **ERRORS**

12068 The `fileno()` function may fail if:

12069 [EBADF] The `stream` argument is not a valid stream.

12070 **EXAMPLES**

12071 None.

12072 **APPLICATION USAGE**

12073 None.

12074 **RATIONALE**

12075 Without some specification of which file descriptors are associated with these streams, it is
12076 impossible for an application to set up the streams for another application it starts with `fork()`
12077 and `exec`. In particular, it would not be possible to write a portable version of the `sh` command
12078 interpreter (although there may be other constraints that would prevent that portability).

12079 **FUTURE DIRECTIONS**

12080 None.

12081 **SEE ALSO**

12082 Section 2.5.1 (on page 35), `fdopen()`, `fopen()`, `stdin`, the Base Definitions volume of
12083 IEEE Std 1003.1-2001, `<stdio.h>`

12084 **CHANGE HISTORY**

12085 First released in Issue 1. Derived from Issue 1 of the SVID.

12086 **Issue 6**

12087 The following new requirements on POSIX implementations derive from alignment with the
12088 Single UNIX Specification:

- 12089 • The [EBADF] optional error condition is added.

12090 **NAME**

12091 flockfile, ftrylockfile, funlockfile — stdio locking functions

12092 **SYNOPSIS**

12093 TSF #include <stdio.h>

```
12094 void flockfile(FILE *file);
12095 int ftrylockfile(FILE *file);
12096 void funlockfile(FILE *file);
12097
```

12098 **DESCRIPTION**

12099 These functions shall provide for explicit application-level locking of stdio (**FILE ***) objects.
 12100 These functions can be used by a thread to delineate a sequence of I/O statements that are
 12101 executed as a unit.

12102 The *flockfile()* function shall acquire for a thread ownership of a (**FILE ***) object.

12103 The *ftrylockfile()* function shall acquire for a thread ownership of a (**FILE ***) object if the object is
 12104 available; *ftrylockfile()* is a non-blocking version of *flockfile()*.

12105 The *funlockfile()* function shall relinquish the ownership granted to the thread. The behavior is
 12106 undefined if a thread other than the current owner calls the *funlockfile()* function.

12107 The functions shall behave as if there is a lock count associated with each (**FILE ***) object. This
 12108 count is implicitly initialized to zero when the (**FILE ***) object is created. The (**FILE ***) object is
 12109 unlocked when the count is zero. When the count is positive, a single thread owns the (**FILE ***)
 12110 object. When the *flockfile()* function is called, if the count is zero or if the count is positive and
 12111 the caller owns the (**FILE ***) object, the count shall be incremented. Otherwise, the calling thread
 12112 shall be suspended, waiting for the count to return to zero. Each call to *funlockfile()* shall
 12113 decrement the count. This allows matching calls to *flockfile()* (or successful calls to *ftrylockfile()*)
 12114 and *funlockfile()* to be nested.

12115 All functions that reference (**FILE ***) objects shall behave as if they use *flockfile()* and *funlockfile()*
 12116 internally to obtain ownership of these (**FILE ***) objects.

12117 **RETURN VALUE**12118 None for *flockfile()* and *funlockfile()*.

12119 The *ftrylockfile()* function shall return zero for success and non-zero to indicate that the lock
 12120 cannot be acquired.

12121 **ERRORS**

12122 No errors are defined.

12123 **EXAMPLES**

12124 None.

12125 **APPLICATION USAGE**

12126 Applications using these functions may be subject to priority inversion, as discussed in the Base
 12127 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

12128 **RATIONALE**

12129 The *flockfile()* and *funlockfile()* functions provide an orthogonal mutual-exclusion lock for each
 12130 **FILE**. The *ftrylockfile()* function provides a non-blocking attempt to acquire a file lock,
 12131 analogous to *pthread_mutex_trylock()*.

12132 These locks behave as if they are the same as those used internally by *stdio* for thread-safety.
 12133 This both provides thread-safety of these functions without requiring a second level of internal
 12134 locking and allows functions in *stdio* to be implemented in terms of other *stdio* functions.

12135 Application writers and implementors should be aware that there are potential deadlock
12136 problems on **FILE** objects. For example, the line-buffered flushing semantics of *stdio* (requested
12137 via `{_IOLBF}`) require that certain input operations sometimes cause the buffered contents of
12138 implementation-defined line-buffered output streams to be flushed. If two threads each hold the
12139 lock on the other's **FILE**, deadlock ensues. This type of deadlock can be avoided by acquiring
12140 **FILE** locks in a consistent order. In particular, the line-buffered output stream deadlock can
12141 typically be avoided by acquiring locks on input streams before locks on output streams if a
12142 thread would be acquiring both.

12143 In summary, threads sharing *stdio* streams with other threads can use *flockfile()* and *funlockfile()*
12144 to cause sequences of I/O performed by a single thread to be kept bundled. The only case where
12145 the use of *flockfile()* and *funlockfile()* is required is to provide a scope protecting uses of the
12146 **_unlocked()* functions/macros. This moves the cost/performance tradeoff to the optimal point.

12147 **FUTURE DIRECTIONS**

12148 None.

12149 **SEE ALSO**

12150 *getc_unlocked()*, *putc_unlocked()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<stdio.h>`

12151 **CHANGE HISTORY**

12152 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

12153 **Issue 6**

12154 These functions are marked as part of the Thread-Safe Functions option.

12155 **NAME**

12156 floor, floorf, floorl — floor function

12157 **SYNOPSIS**

12158 #include <math.h>

12159 double floor(double x);

12160 float floorf(float x);

12161 long double floorl(long double x);

12162 **DESCRIPTION**

12163 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 12164 conflict between the requirements described here and the ISO C standard is unintentional. This
 12165 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

12166 These functions shall compute the largest integral value not greater than *x*.

12167 An application wishing to check for error situations should set *errno* to zero and call
 12168 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 12169 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 12170 zero, an error has occurred.

12171 **RETURN VALUE**

12172 Upon successful completion, these functions shall return the largest integral value not greater
 12173 than *x*, expressed as a **double**, **float**, or **long double**, as appropriate for the return type of the
 12174 function.

12175 **MX** If *x* is NaN, a NaN shall be returned.12176 If *x* is ± 0 or $\pm \text{Inf}$, *x* shall be returned.

12177 **XSI** If the correct value would cause overflow, a range error shall occur and *floor*(*x*), *floorf*(*x*), and
 12178 *floorl*(*x*) shall return the value of the macro `-HUGE_VAL`, `-HUGE_VALF`, and `-HUGE_VALL`,
 12179 respectively.

12180 **ERRORS**

12181 These functions shall fail if:

12182 **XSI** **Range Error** The result would cause an overflow.

12183 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 12184 then *errno* shall be set to [ERANGE]. If the integer expression
 12185 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 12186 floating-point exception shall be raised.

12187 **EXAMPLES**

12188 None.

12189 **APPLICATION USAGE**

12190 The integral value returned by these functions might not be expressible as an **int** or **long**. The
 12191 return value should be tested before assigning it to an integer type to avoid the undefined results
 12192 of an integer overflow.

12193 The *floor*(*x*) function can only overflow when the floating-point representation has
 12194 `DBL_MANT_DIG > DBL_MAX_EXP`.

12195 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
 12196 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

12197 **RATIONALE**

12198 None.

12199 **FUTURE DIRECTIONS**

12200 None.

12201 **SEE ALSO**12202 *ceil()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,
12203 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>12204 **CHANGE HISTORY**

12205 First released in Issue 1. Derived from Issue 1 of the SVID.

12206 **Issue 5**12207 The DESCRIPTION is updated to indicate how an application should check for an error. This
12208 text was previously published in the APPLICATION USAGE section.12209 **Issue 6**12210 The *floorf()* and *floorl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.12211 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
12212 revised to align with the ISO/IEC 9899:1999 standard.12213 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
12214 marked.

12215 **NAME**

12216 fma, fmaf, fmal — floating-point multiply-add

12217 **SYNOPSIS**

12218 #include <math.h>

12219 double fma(double x, double y, double z);

12220 float fmaf(float x, float y, float z);

12221 long double fmal(long double x, long double y, long double z);

12222 **DESCRIPTION**

12223 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 12224 conflict between the requirements described here and the ISO C standard is unintentional. This
 12225 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

12226 These functions shall compute $(x * y) + z$, rounded as one ternary operation: they shall compute
 12227 the value (as if) to infinite precision and round once to the result format, according to the
 12228 rounding mode characterized by the value of FLT_ROUNDS.

12229 An application wishing to check for error situations should set *errno* to zero and call
 12230 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 12231 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 12232 zero, an error has occurred.

12233 **RETURN VALUE**

12234 Upon successful completion, these functions shall return $(x * y) + z$, rounded as one ternary
 12235 operation.

12236 **MX** If *x* or *y* are NaN, a NaN shall be returned.

12237 If *x* multiplied by *y* is an exact infinity and *z* is also an infinity but with the opposite sign, a
 12238 domain error shall occur, and either a NaN (if supported), or an implementation-defined value
 12239 shall be returned.

12240 If one of *x* and *y* is infinite, the other is zero, and *z* is not a NaN, a domain error shall occur, and
 12241 either a NaN (if supported), or an implementation-defined value shall be returned.

12242 If one of *x* and *y* is infinite, the other is zero, and *z* is a NaN, a NaN shall be returned and a
 12243 domain error may occur.

12244 If $x * y$ is not $0 * \text{Inf}$ nor $\text{Inf} * 0$ and *z* is a NaN, a NaN shall be returned.

12245 **ERRORS**

12246 These functions shall fail if:

12247 **MX** **Domain Error** The value of $x * y + z$ is invalid, or the value $x * y$ is invalid and *z* is not a NaN.

12248 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 12249 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 12250 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 12251 shall be raised.

12252 **MX** **Range Error** The result overflows.

12253 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 12254 then *errno* shall be set to [ERANGE]. If the integer expression
 12255 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
 12256 floating-point exception shall be raised.

12257 These functions may fail if:

12258 MX **Domain Error** The value $x*y$ is invalid and z is a NaN.

12259 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,
 12260 then *errno* shall be set to [EDOM]. If the integer expression `(math_errhandling`
 12261 `& MATH_ERREXCEPT)` is non-zero, then the invalid floating-point exception
 12262 shall be raised.

12263 MX **Range Error** The result underflows.

12264 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,
 12265 then *errno* shall be set to [ERANGE]. If the integer expression
 12266 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the underflow
 12267 floating-point exception shall be raised.

12268 EXAMPLES

12269 None.

12270 APPLICATION USAGE

12271 On error, the expressions `(math_errhandling & MATH_ERRNO)` and `(math_errhandling &`
 12272 `MATH_ERREXCEPT)` are independent of each other, but at least one of them must be non-zero.

12273 RATIONALE

12274 In many cases, clever use of floating (*fused*) multiply-add leads to much improved code; but its
 12275 unexpected use by the compiler can undermine carefully written code. The `FP_CONTRACT`
 12276 macro can be used to disallow use of floating multiply-add; and the `fma()` function guarantees
 12277 its use where desired. Many current machines provide hardware floating multiply-add
 12278 instructions; software implementation can be used for others.

12279 FUTURE DIRECTIONS

12280 None.

12281 SEE ALSO

12282 `feclearexcept()`, `fetestexcept()`, the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18,
 12283 Treatment of Error Conditions for Mathematical Functions, <math.h>

12284 CHANGE HISTORY

12285 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

12286 **NAME**

12287 fmax, fmaxf, fmaxl — determine maximum numeric value of two floating-point numbers

12288 **SYNOPSIS**

12289 #include <math.h>

12290 double fmax(double x, double y);

12291 float fmaxf(float x, float y);

12292 long double fmaxl(long double x, long double y);

12293 **DESCRIPTION**

12294 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
12295 conflict between the requirements described here and the ISO C standard is unintentional. This
12296 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

12297 These functions shall determine the maximum numeric value of their arguments. NaN
12298 arguments shall be treated as missing data: if one argument is a NaN and the other numeric,
12299 then these functions shall choose the numeric value.

12300 **RETURN VALUE**

12301 Upon successful completion, these functions shall return the maximum numeric value of their
12302 arguments.

12303 If just one argument is a NaN, the other argument shall be returned.

12304 **MX** If *x* and *y* are NaN, a NaN shall be returned.

12305 **ERRORS**

12306 No errors are defined.

12307 **EXAMPLES**

12308 None.

12309 **APPLICATION USAGE**

12310 None.

12311 **RATIONALE**

12312 None.

12313 **FUTURE DIRECTIONS**

12314 None.

12315 **SEE ALSO**

12316 *fdim()*, *fmin()*, the Base Definitions volume of IEEE Std 1003.1-2001, <math.h>

12317 **CHANGE HISTORY**

12318 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

12319 **NAME**

12320 `fmin`, `fminf`, `fminl` — determine minimum numeric value of two floating-point numbers

12321 **SYNOPSIS**

12322 `#include <math.h>`

12323 `double fmin(double x, double y);`

12324 `float fminf(float x, float y);`

12325 `long double fminl(long double x, long double y);`

12326 **DESCRIPTION**

12327 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
12328 conflict between the requirements described here and the ISO C standard is unintentional. This
12329 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

12330 These functions shall determine the minimum numeric value of their arguments. NaN
12331 arguments shall be treated as missing data: if one argument is a NaN and the other numeric,
12332 then these functions shall choose the numeric value.

12333 **RETURN VALUE**

12334 Upon successful completion, these functions shall return the minimum numeric value of their
12335 arguments.

12336 If just one argument is a NaN, the other argument shall be returned.

12337 **MX** If *x* and *y* are NaN, a NaN shall be returned.

12338 **ERRORS**

12339 No errors are defined.

12340 **EXAMPLES**

12341 None.

12342 **APPLICATION USAGE**

12343 None.

12344 **RATIONALE**

12345 None.

12346 **FUTURE DIRECTIONS**

12347 None.

12348 **SEE ALSO**

12349 `fdim()`, `fmax()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<math.h>`

12350 **CHANGE HISTORY**

12351 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

12352 **NAME**

12353 fmod, fmodf, fmodl — floating-point remainder value function

12354 **SYNOPSIS**

12355 #include <math.h>

12356 double fmod(double x, double y);

12357 float fmodf(float x, float y);

12358 long double fmodl(long double x, long double y);

12359 **DESCRIPTION**

12360 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 12361 conflict between the requirements described here and the ISO C standard is unintentional. This
 12362 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

12363 These functions shall return the floating-point remainder of the division of x by y .

12364 An application wishing to check for error situations should set *errno* to zero and call
 12365 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 12366 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 12367 zero, an error has occurred.

12368 **RETURN VALUE**

12369 These functions shall return the value $x-i*y$, for some integer i such that, if y is non-zero, the
 12370 result has the same sign as x and magnitude less than the magnitude of y .

12371 If the correct value would cause underflow, and is not representable, a range error may occur,
 12372 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.

12373 **MX** If x or y is NaN, a NaN shall be returned.

12374 If y is zero, a domain error shall occur, and either a NaN (if supported), or an implementation-
 12375 defined value shall be returned.

12376 If x is infinite, a domain error shall occur, and either a NaN (if supported), or an
 12377 implementation-defined value shall be returned.

12378 If x is ± 0 and y is not zero, ± 0 shall be returned.12379 If x is not infinite and y is $\pm \text{Inf}$, x shall be returned.

12380 If the correct value would cause underflow, and is representable, a range error may occur and
 12381 the correct value shall be returned.

12382 **ERRORS**

12383 These functions shall fail if:

12384 **MX** **Domain Error** The x argument is infinite or y is zero.

12385 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 12386 then *errno* shall be set to [EDOM]. If the integer expression (math_errhandling
 12387 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 12388 shall be raised.

12389 These functions may fail if:

12390 **Range Error** The result underflows.

12391 If the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
 12392 then *errno* shall be set to [ERANGE]. If the integer expression
 12393 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the underflow
 12394 floating-point exception shall be raised.

12395 **EXAMPLES**

12396 None.

12397 **APPLICATION USAGE**

12398 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
12399 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

12400 **RATIONALE**

12401 None.

12402 **FUTURE DIRECTIONS**

12403 None.

12404 **SEE ALSO**

12405 *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,
12406 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

12407 **CHANGE HISTORY**

12408 First released in Issue 1. Derived from Issue 1 of the SVID.

12409 **Issue 5**

12410 The DESCRIPTION is updated to indicate how an application should check for an error. This
12411 text was previously published in the APPLICATION USAGE section.

12412 **Issue 6**12413 The behavior for when the *y* argument is zero is now defined.

12414 The *fmodf()* and *fmodl()* functions are added for alignment with the ISO/IEC 9899:1999
12415 standard.

12416 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
12417 revised to align with the ISO/IEC 9899:1999 standard.

12418 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
12419 marked.

12420 **NAME**12421 `fmtmsg` — display a message in the specified format on standard error and/or a system console12422 **SYNOPSIS**12423 XSI

```
#include <fmtmsg.h>
```

12424

```
int fmtmsg(long classification, const char *label, int severity,  
12425 const char *text, const char *action, const char *tag);
```

12426

12427 **DESCRIPTION**12428 The `fmtmsg()` function shall display messages in a specified format instead of the traditional
12429 `printf()` function.12430 Based on a message's classification component, `fmtmsg()` shall write a formatted message either
12431 to standard error, to the console, or to both.12432 A formatted message consists of up to five components as defined below. The component
12433 *classification* is not part of a message displayed to the user, but defines the source of the message
12434 and directs the display of the formatted message.12435 *classification* Contains the sum of identifying values constructed from the constants defined
12436 below. Any one identifier from a subclass may be used in combination with a
12437 single identifier from a different subclass. Two or more identifiers from the
12438 same subclass should not be used together, with the exception of identifiers
12439 from the display subclass. (Both display subclass identifiers may be used so
12440 that messages can be displayed to both standard error and the system
12441 console.)12442 **Major Classifications**12443 Identifies the source of the condition. Identifiers are: MM_HARD
12444 (hardware), MM_SOFT (software), and MM_FIRM (firmware).12445 **Message Source Subclassifications**12446 Identifies the type of software in which the problem is detected.
12447 Identifiers are: MM_APPL (application), MM_UTIL (utility), and
12448 MM_OPYSYS (operating system).12449 **Display Subclassifications**12450 Indicates where the message is to be displayed. Identifiers are:
12451 MM_PRINT to display the message on the standard error stream,
12452 MM_CONSOLE to display the message on the system console. One or
12453 both identifiers may be used.12454 **Status Subclassifications**12455 Indicates whether the application can recover from the condition.
12456 Identifiers are: MM_RECOVER (recoverable) and MM_NRECOV (non-
12457 recoverable).12458 An additional identifier, MM_NULLMC, indicates that no classification
12459 component is supplied for the message.12460 *label* Identifies the source of the message. The format is two fields separated by a
12461 colon. The first field is up to 10 bytes, the second is up to 14 bytes.12462 *severity* Indicates the seriousness of the condition. Identifiers for the levels of *severity*
12463 are:

12464		MM_HALT	Indicates that the application has encountered a severe fault and is halting. Produces the string "HALT".
12465			
12466		MM_ERROR	Indicates that the application has detected a fault. Produces the string "ERROR".
12467			
12468		MM_WARNING	Indicates a condition that is out of the ordinary, that might be a problem, and should be watched. Produces the string "WARNING".
12469			
12470			
12471		MM_INFO	Provides information about a condition that is not in error. Produces the string "INFO".
12472			
12473		MM_NOSEV	Indicates that no severity level is supplied for the message.
12474	<i>text</i>		Describes the error condition that produced the message. The character string is not limited to a specific size. If the character string is empty, then the text produced is unspecified.
12475			
12476			
12477	<i>action</i>		Describes the first step to be taken in the error-recovery process. The <i>fmtmsg()</i> function precedes the action string with the prefix: "TO FIX:". The <i>action</i> string is not limited to a specific size.
12478			
12479			
12480	<i>tag</i>		An identifier that references on-line documentation for the message. Suggested usage is that <i>tag</i> includes the <i>label</i> and a unique identifying number. A sample <i>tag</i> is "XSI:cat:146".
12481			
12482			
12483			The <i>MSGVERB</i> environment variable (for message verbosity) shall determine for <i>fmtmsg()</i> which message components it is to select when writing messages to standard error. The value of <i>MSGVERB</i> shall be a colon-separated list of optional keywords. Valid keywords are: <i>label</i> , <i>severity</i> , <i>text</i> , <i>action</i> , and <i>tag</i> . If <i>MSGVERB</i> contains a keyword for a component and the component's value is not the component's null value, <i>fmtmsg()</i> shall include that component in the message when writing the message to standard error. If <i>MSGVERB</i> does not include a keyword for a message component, that component shall not be included in the display of the message. The keywords may appear in any order. If <i>MSGVERB</i> is not defined, if its value is the null string, if its value is not of the correct format, or if it contains keywords other than the valid ones listed above, <i>fmtmsg()</i> shall select all components.
12484			
12485			
12486			
12487			
12488			
12489			
12490			
12491			
12492			
12493			<i>MSGVERB</i> shall determine which components are selected for display to standard error. All message components shall be included in console messages.
12494			
12495	RETURN VALUE		
12496			The <i>fmtmsg()</i> function shall return one of the following values:
12497	MM_OK		The function succeeded.
12498	MM_NOTOK		The function failed completely.
12499	MM_NOMSG		The function was unable to generate a message on standard error, but otherwise succeeded.
12500			
12501	MM_NOCON		The function was unable to generate a console message, but otherwise succeeded.
12502			
12503	ERRORS		
12504			None.

12505 **EXAMPLES**

12506 1. The following example of *fmtmsg()*:

```
12507     fmtmsg(MM_PRINT, "XSI:cat", MM_ERROR, "illegal option",  
12508     "refer to cat in user's reference manual", "XSI:cat:001")
```

12509 produces a complete message in the specified message format:

```
12510     XSI:cat: ERROR: illegal option  
12511     TO FIX: refer to cat in user's reference manual XSI:cat:001
```

12512 2. When the environment variable *MSGVERB* is set as follows:

```
12513     MSGVERB=severity:text:action
```

12514 and Example 1 is used, *fmtmsg()* produces:

```
12515     ERROR: illegal option  
12516     TO FIX: refer to cat in user's reference manual
```

12517 **APPLICATION USAGE**

12518 One or more message components may be systematically omitted from messages generated by
12519 an application by using the null value of the argument for that component.

12520 **RATIONALE**

12521 None.

12522 **FUTURE DIRECTIONS**

12523 None.

12524 **SEE ALSO**

12525 *printf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <*fmtmsg.h*>

12526 **CHANGE HISTORY**

12527 First released in Issue 4, Version 2.

12528 **Issue 5**

12529 Moved from X/OPEN UNIX extension to BASE.

12530 NAME

12531 fnmatch — match a filename or a pathname

12532 SYNOPSIS

12533 #include <fnmatch.h>

12534 int fnmatch(const char **pattern*, const char **string*, int *flags*);

12535 DESCRIPTION

12536 The *fnmatch()* function shall match patterns as described in the Shell and Utilities volume of
 12537 IEEE Std 1003.1-2001, Section 2.13.1, Patterns Matching a Single Character, and Section 2.13.2,
 12538 Patterns Matching Multiple Characters. It checks the string specified by the *string* argument to
 12539 see if it matches the pattern specified by the *pattern* argument.

12540 The *flags* argument shall modify the interpretation of *pattern* and *string*. It is the bitwise-inclusive
 12541 OR of zero or more of the flags defined in <fnmatch.h>. If the FNM_PATHNAME flag is set in
 12542 *flags*, then a slash character ('/') in *string* shall be explicitly matched by a slash in *pattern*; it shall
 12543 not be matched by either the asterisk or question-mark special characters, nor by a bracket
 12544 expression. If the FNM_PATHNAME flag is not set, the slash character shall be treated as an
 12545 ordinary character.

12546 If FNM_NOESCAPE is not set in *flags*, a backslash character ('\ ') in *pattern* followed by any
 12547 other character shall match that second character in *string*. In particular, "\\\" shall match a
 12548 backslash in *string*. If FNM_NOESCAPE is set, a backslash character shall be treated as an
 12549 ordinary character.

12550 If FNM_PERIOD is set in *flags*, then a leading period ('.') in *string* shall match a period in
 12551 *pattern*; as described by rule 2 in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section
 12552 2.13.3, Patterns Used for Filename Expansion where the location of “leading” is indicated by the
 12553 value of FNM_PATHNAME:

- 12554 • If FNM_PATHNAME is set, a period is “leading” if it is the first character in *string* or if it
 12555 immediately follows a slash.
- 12556 • If FNM_PATHNAME is not set, a period is “leading” only if it is the first character of *string*.

12557 If FNM_PERIOD is not set, then no special restrictions are placed on matching a period.

12558 RETURN VALUE

12559 If *string* matches the pattern specified by *pattern*, then *fnmatch()* shall return 0. If there is no
 12560 match, *fnmatch()* shall return FNM_NOMATCH, which is defined in <fnmatch.h>. If an error
 12561 occurs, *fnmatch()* shall return another non-zero value.

12562 ERRORS

12563 No errors are defined.

12564 EXAMPLES

12565 None.

12566 APPLICATION USAGE

12567 The *fnmatch()* function has two major uses. It could be used by an application or utility that
 12568 needs to read a directory and apply a pattern against each entry. The *find* utility is an example of
 12569 this. It can also be used by the *pax* utility to process its *pattern* operands, or by applications that
 12570 need to match strings in a similar manner.

12571 The name *fnmatch()* is intended to imply *filename* match, rather than *pathname* match. The default
 12572 action of this function is to match filenames, rather than pathnames, since it gives no special
 12573 significance to the slash character. With the FNM_PATHNAME flag, *fnmatch()* does match
 12574 pathnames, but without tilde expansion, parameter expansion, or special treatment for a period

12575 at the beginning of a filename.

12576 **RATIONALE**

12577 This function replaced the REG_FILENAME flag of *regcomp()* in early proposals of this volume
12578 of IEEE Std 1003.1-2001. It provides virtually the same functionality as the *regcomp()* and
12579 *regexec()* functions using the REG_FILENAME and REG_FSLASH flags (the REG_FSLASH flag
12580 was proposed for *regcomp()*, and would have had the opposite effect from FNM_PATHNAME),
12581 but with a simpler function and less system overhead.

12582 **FUTURE DIRECTIONS**

12583 None.

12584 **SEE ALSO**

12585 *glob()*, *wordexp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <fnmatch.h>, the Shell
12586 and Utilities volume of IEEE Std 1003.1-2001

12587 **CHANGE HISTORY**

12588 First released in Issue 4. Derived from the ISO POSIX-2 standard.

12589 **Issue 5**

12590 Moved from POSIX2 C-language Binding to BASE.

12591 NAME

12592 fopen — open a stream

12593 SYNOPSIS

12594 #include <stdio.h>

12595 FILE *fopen(const char *restrict filename, const char *restrict mode);

12596 DESCRIPTION

12597 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 12598 conflict between the requirements described here and the ISO C standard is unintentional. This
 12599 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

12600 The *fopen()* function shall open the file whose pathname is the string pointed to by *filename*, and
 12601 associates a stream with it.

12602 The *mode* argument points to a string. If the string is one of the following, the file shall be opened
 12603 in the indicated mode. Otherwise, the behavior is undefined.

12604	<i>r</i> or <i>rb</i>	Open file for reading.
12605	<i>w</i> or <i>wb</i>	Truncate to zero length or create file for writing.
12606	<i>a</i> or <i>ab</i>	Append; open or create file for writing at end-of-file.
12607	<i>r+</i> or <i>rb+</i> or <i>r+b</i>	Open file for update (reading and writing).
12608	<i>w+</i> or <i>wb+</i> or <i>w+b</i>	Truncate to zero length or create file for update.
12609	<i>a+</i> or <i>ab+</i> or <i>a+b</i>	Append; open or create file for update, writing at end-of-file.

12610 CX The character 'b' shall have no effect, but is allowed for ISO C standard conformance. Opening
 12611 a file with read mode (*r* as the first character in the *mode* argument) shall fail if the file does not
 12612 exist or cannot be read.

12613 Opening a file with append mode (*a* as the first character in the *mode* argument) shall cause all
 12614 subsequent writes to the file to be forced to the then current end-of-file, regardless of intervening
 12615 calls to *fseek()*.

12616 When a file is opened with update mode ('+' as the second or third character in the *mode*
 12617 argument), both input and output may be performed on the associated stream. However, the
 12618 application shall ensure that output is not directly followed by input without an intervening call
 12619 to *fflush()* or to a file positioning function (*fseek()*, *fsetpos()*, or *rewind()*), and input is not directly
 12620 followed by output without an intervening call to a file positioning function, unless the input
 12621 operation encounters end-of-file.

12622 When opened, a stream is fully buffered if and only if it can be determined not to refer to an
 12623 interactive device. The error and end-of-file indicators for the stream shall be cleared.

12624 CX If *mode* is *w*, *wb*, *a*, *ab*, *w+*, *wb+*, *w+b*, *a+*, *ab+*, or *a+b*, and the file did not previously exist, upon
 12625 successful completion, the *fopen()* function shall mark for update the *st_atime*, *st_ctime*, and
 12626 *st_mtime* fields of the file and the *st_ctime* and *st_mtime* fields of the parent directory.

12627 If *mode* is *w*, *wb*, *w+*, *wb+*, or *w+b*, and the file did previously exist, upon successful completion,
 12628 *fopen()* shall mark for update the *st_ctime* and *st_mtime* fields of the file. The *fopen()* function
 12629 shall allocate a file descriptor as *open()* does.

12630 XSI After a successful call to the *fopen()* function, the orientation of the stream shall be cleared, the
 12631 encoding rule shall be cleared, and the associated **mbstate_t** object shall be set to describe an
 12632 initial conversion state.

12633 CX The largest value that can be represented correctly in an object of type `off_t` shall be established
 12634 as the offset maximum in the open file description.

12635 RETURN VALUE

12636 Upon successful completion, `fopen()` shall return a pointer to the object controlling the stream.
 12637 CX Otherwise, a null pointer shall be returned, and `errno` shall be set to indicate the error.

12638 ERRORS

12639 The `fopen()` function shall fail if:

12640 CX [EACCES] Search permission is denied on a component of the path prefix, or the file
 12641 exists and the permissions specified by `mode` are denied, or the file does not
 12642 exist and write permission is denied for the parent directory of the file to be
 12643 created.

12644 CX [EINTR] A signal was caught during `fopen()`.

12645 CX [EISDIR] The named file is a directory and `mode` requires write access.

12646 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the `path`
 12647 argument.

12648 CX [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

12649 CX [ENAMETOOLONG]
 12650 The length of the `filename` argument exceeds {PATH_MAX} or a pathname
 12651 component is longer than {NAME_MAX}.

12652 CX [ENFILE] The maximum allowable number of files is currently open in the system.

12653 CX [ENOENT] A component of `filename` does not name an existing file or `filename` is an empty
 12654 string.

12655 CX [ENOSPC] The directory or file system that would contain the new file cannot be
 12656 expanded, the file does not exist, and the file was to be created.

12657 CX [ENOTDIR] A component of the path prefix is not a directory.

12658 CX [ENXIO] The named file is a character special or block special file, and the device
 12659 associated with this special file does not exist.

12660 CX [EOVERFLOW] The named file is a regular file and the size of the file cannot be represented
 12661 correctly in an object of type `off_t`.

12662 CX [EROFS] The named file resides on a read-only file system and `mode` requires write
 12663 access.

12664 The `fopen()` function may fail if:

12665 CX [EINVAL] The value of the `mode` argument is not valid.

12666 CX [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 12667 resolution of the `path` argument.

12668 CX [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

12669 CX [EMFILE] {STREAM_MAX} streams are currently open in the calling process.

12670 CX [ENAMETOOLONG]
 12671 Pathname resolution of a symbolic link produced an intermediate result
 12672 whose length exceeds {PATH_MAX}.

12673 CX [ENOMEM] Insufficient storage space is available.

12674 CX [ETXTBSY] The file is a pure procedure (shared text) file that is being executed and *mode*
12675 requires write access.

12676 EXAMPLES

12677 Opening a File

12678 The following example tries to open the file named **file** for reading. The *fopen()* function returns
12679 a file pointer that is used in subsequent *fgets()* and *fclose()* calls. If the program cannot open the
12680 file, it just ignores it.

```
12681 #include <stdio.h>
12682 ...
12683 FILE *fp;
12684 ...
12685 void rgrep(const char *file)
12686 {
12687     ...
12688     if ((fp = fopen(file, "r")) == NULL)
12689         return;
12690     ...
12691 }
```

12692 APPLICATION USAGE

12693 None.

12694 RATIONALE

12695 None.

12696 FUTURE DIRECTIONS

12697 None.

12698 SEE ALSO

12699 *fclose()*, *fdopen()*, *freopen()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

12700 CHANGE HISTORY

12701 First released in Issue 1. Derived from Issue 1 of the SVID.

12702 Issue 5

12703 Large File Summit extensions are added.

12704 Issue 6

12705 Extensions beyond the ISO C standard are marked.

12706 The following new requirements on POSIX implementations derive from alignment with the
12707 Single UNIX Specification:

- 12708 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file
12709 description. This change is to support large files.
- 12710 • In the ERRORS section, the [Eoverflow] condition is added. This change is to support
12711 large files.
- 12712 • The [ELOOP] mandatory error condition is added.
- 12713 • The [EINVAL], [EMFILE], [ENAMETOOLONG], [ENOMEM], and [ETXTBSY] optional error
12714 conditions are added.

- 12715 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.
- 12716 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:
- 12717 • The prototype for *fopen()* is updated.
 - 12718 • The DESCRIPTION is updated to note that if the argument *mode* points to a string other than
12719 those listed, then the behavior is undefined.
- 12720 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
12721 [ELOOP] error condition is added.

12722 NAME

12723 fork — create a new process

12724 SYNOPSIS

12725 #include <unistd.h>

12726 pid_t fork(void);

12727 DESCRIPTION

12728 The *fork()* function shall create a new process. The new process (child process) shall be an exact
12729 copy of the calling process (parent process) except as detailed below:

- 12730 • The child process shall have a unique process ID.
- 12731 • The child process ID also shall not match any active process group ID.
- 12732 • The child process shall have a different parent process ID, which shall be the process ID of
12733 the calling process.
- 12734 • The child process shall have its own copy of the parent's file descriptors. Each of the child's
12735 file descriptors shall refer to the same open file description with the corresponding file
12736 descriptor of the parent.
- 12737 • The child process shall have its own copy of the parent's open directory streams. Each open
12738 directory stream in the child process may share directory stream positioning with the
12739 corresponding directory stream of the parent.
- 12740 XSI • The child process shall have its own copy of the parent's message catalog descriptors.
- 12741 • The child process' values of *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime* shall be set to 0.
- 12742 • The time left until an alarm clock signal shall be reset to zero, and the alarm, if any, shall be
12743 canceled; see *alarm()*.
- 12744 XSI • All *semadj* values shall be cleared.
- 12745 • File locks set by the parent process shall not be inherited by the child process.
- 12746 • The set of signals pending for the child process shall be initialized to the empty set.
- 12747 XSI • Interval timers shall be reset in the child process.
- 12748 SEM • Any semaphores that are open in the parent process shall also be open in the child process.
- 12749 ML • The child process shall not inherit any address space memory locks established by the parent
12750 process via calls to *mlockall()* or *mlock()*.
- 12751 MF|SHM • Memory mappings created in the parent shall be retained in the child process.
12752 MAP_PRIVATE mappings inherited from the parent shall also be MAP_PRIVATE mappings
12753 in the child, and any modifications to the data in these mappings made by the parent prior to
12754 calling *fork()* shall be visible to the child. Any modifications to the data in MAP_PRIVATE
12755 mappings made by the parent after *fork()* returns shall be visible only to the parent.
12756 Modifications to the data in MAP_PRIVATE mappings made by the child shall be visible only
12757 to the child.
- 12758 PS • For the SCHED_FIFO and SCHED_RR scheduling policies, the child process shall inherit the
12759 policy and priority settings of the parent process during a *fork()* function. For other
12760 scheduling policies, the policy and priority settings on *fork()* are implementation-defined.
- 12761 TMR • Per-process timers created by the parent shall not be inherited by the child process.
- 12762 MSG • The child process shall have its own copy of the message queue descriptors of the parent.
12763 Each of the message descriptors of the child shall refer to the same open message queue

- 12764 description as the corresponding message descriptor of the parent.
- 12765 AIO • No asynchronous input or asynchronous output operations shall be inherited by the child
12766 process.
- 12767 • A process shall be created with a single thread. If a multi-threaded process calls *fork()*, the
12768 new process shall contain a replica of the calling thread and its entire address space, possibly
12769 including the states of mutexes and other resources. Consequently, to avoid errors, the child
12770 process may only execute async-signal-safe operations until such time as one of the *exec*
12771 THR functions is called. Fork handlers may be established by means of the *pthread_atfork()*
12772 function in order to maintain application invariants across *fork()* calls.
- 12773 When the application calls *fork()* from a signal handler and any of the fork handlers
12774 registered by *pthread_atfork()* calls a function that is not asynch-signal-safe, the behavior is
12775 undefined.
- 12776 TRC TRI • If the Trace option and the Trace Inherit option are both supported:
- 12777 If the calling process was being traced in a trace stream that had its inheritance policy set to
12778 POSIX_TRACE_INHERITED, the child process shall be traced into that trace stream, and the
12779 child process shall inherit the parent's mapping of trace event names to trace event type
12780 identifiers. If the trace stream in which the calling process was being traced had its
12781 inheritance policy set to POSIX_TRACE_CLOSE_FOR_CHILD, the child process shall not be
12782 traced into that trace stream. The inheritance policy is set by a call to the
12783 *posix_trace_attr_setinherited()* function.
- 12784 TRC • If the Trace option is supported, but the Trace Inherit option is not supported:
- 12785 The child process shall not be traced into any of the trace streams of its parent process.
- 12786 TRC • If the Trace option is supported, the child process of a trace controller process shall not
12787 control the trace streams controlled by its parent process.
- 12788 CPT • The initial value of the CPU-time clock of the child process shall be set to zero.
- 12789 TCT • The initial value of the CPU-time clock of the single thread of the child process shall be set to
12790 zero.
- 12791 All other process characteristics defined by IEEE Std 1003.1-2001 shall be the same in the parent
12792 and child processes. The inheritance of process characteristics not defined by
12793 IEEE Std 1003.1-2001 is unspecified by IEEE Std 1003.1-2001.
- 12794 After *fork()*, both the parent and the child processes shall be capable of executing independently
12795 before either one terminates.
- 12796 **RETURN VALUE**
- 12797 Upon successful completion, *fork()* shall return 0 to the child process and shall return the
12798 process ID of the child process to the parent process. Both processes shall continue to execute
12799 from the *fork()* function. Otherwise, -1 shall be returned to the parent process, no child process
12800 shall be created, and *errno* shall be set to indicate the error.
- 12801 **ERRORS**
- 12802 The *fork()* function shall fail if:
- 12803 [EAGAIN] The system lacked the necessary resources to create another process, or the
12804 system-imposed limit on the total number of processes under execution
12805 system-wide or by a single user {CHILD_MAX} would be exceeded.

12806 The *fork()* function may fail if:

12807 [ENOMEM] Insufficient storage space is available.

12808 EXAMPLES

12809 None.

12810 APPLICATION USAGE

12811 None.

12812 RATIONALE

12813 Many historical implementations have timing windows where a signal sent to a process group
 12814 (for example, an interactive SIGINT) just prior to or during execution of *fork()* is delivered to the
 12815 parent following the *fork()* but not to the child because the *fork()* code clears the child's set of
 12816 pending signals. This volume of IEEE Std 1003.1-2001 does not require, or even permit, this
 12817 behavior. However, it is pragmatic to expect that problems of this nature may continue to exist
 12818 in implementations that appear to conform to this volume of IEEE Std 1003.1-2001 and pass
 12819 available verification suites. This behavior is only a consequence of the implementation failing to
 12820 make the interval between signal generation and delivery totally invisible. From the
 12821 application's perspective, a *fork()* call should appear atomic. A signal that is generated prior to
 12822 the *fork()* should be delivered prior to the *fork()*. A signal sent to the process group after the
 12823 *fork()* should be delivered to both parent and child. The implementation may actually initialize
 12824 internal data structures corresponding to the child's set of pending signals to include signals
 12825 sent to the process group during the *fork()*. Since the *fork()* call can be considered as atomic
 12826 from the application's perspective, the set would be initialized as empty and such signals would
 12827 have arrived after the *fork()*; see also <signal.h>.

12828 One approach that has been suggested to address the problem of signal inheritance across *fork()*
 12829 is to add an [EINTR] error, which would be returned when a signal is detected during the call.
 12830 While this is preferable to losing signals, it was not considered an optimal solution. Although it
 12831 is not recommended for this purpose, such an error would be an allowable extension for an
 12832 implementation.

12833 The [ENOMEM] error value is reserved for those implementations that detect and distinguish
 12834 such a condition. This condition occurs when an implementation detects that there is not enough
 12835 memory to create the process. This is intended to be returned when [EAGAIN] is inappropriate
 12836 because there can never be enough memory (either primary or secondary storage) to perform the
 12837 operation. Since *fork()* duplicates an existing process, this must be a condition where there is
 12838 sufficient memory for one such process, but not for two. Many historical implementations
 12839 actually return [ENOMEM] due to temporary lack of memory, a case that is not generally
 12840 distinct from [EAGAIN] from the perspective of a conforming application.

12841 Part of the reason for including the optional error [ENOMEM] is because the SVID specifies it
 12842 and it should be reserved for the error condition specified there. The condition is not applicable
 12843 on many implementations.

12844 IEEE Std 1003.1-1988 neglected to require concurrent execution of the parent and child of *fork()*.
 12845 A system that single-threads processes was clearly not intended and is considered an
 12846 unacceptable "toy implementation" of this volume of IEEE Std 1003.1-2001. The only objection
 12847 anticipated to the phrase "executing independently" is testability, but this assertion should be
 12848 testable. Such tests require that both the parent and child can block on a detectable action of the
 12849 other, such as a write to a pipe or a signal. An interactive exchange of such actions should be
 12850 possible for the system to conform to the intent of this volume of IEEE Std 1003.1-2001.

12851 The [EAGAIN] error exists to warn applications that such a condition might occur. Whether it
 12852 occurs or not is not in any practical sense under the control of the application because the
 12853 condition is usually a consequence of the user's use of the system, not of the application's code.

12854 Thus, no application can or should rely upon its occurrence under any circumstances, nor
 12855 should the exact semantics of what concept of “user” is used be of concern to the application
 12856 writer. Validation writers should be cognizant of this limitation.

12857 There are two reasons why POSIX programmers call *fork()*. One reason is to create a new thread
 12858 of control within the same program (which was originally only possible in POSIX by creating a
 12859 new process); the other is to create a new process running a different program. In the latter case,
 12860 the call to *fork()* is soon followed by a call to one of the *exec* functions.

12861 The general problem with making *fork()* work in a multi-threaded world is what to do with all
 12862 of the threads. There are two alternatives. One is to copy all of the threads into the new process.
 12863 This causes the programmer or implementation to deal with threads that are suspended on
 12864 system calls or that might be about to execute system calls that should not be executed in the
 12865 new process. The other alternative is to copy only the thread that calls *fork()*. This creates the
 12866 difficulty that the state of process-local resources is usually held in process memory. If a thread
 12867 that is not calling *fork()* holds a resource, that resource is never released in the child process
 12868 because the thread whose job it is to release the resource does not exist in the child process.

12869 When a programmer is writing a multi-threaded program, the first described use of *fork()*,
 12870 creating new threads in the same program, is provided by the *pthread_create()* function. The
 12871 *fork()* function is thus used only to run new programs, and the effects of calling functions that
 12872 require certain resources between the call to *fork()* and the call to an *exec* function are undefined.

12873 The addition of the *forkall()* function to the standard was considered and rejected. The *forkall()*
 12874 function lets all the threads in the parent be duplicated in the child. This essentially duplicates
 12875 the state of the parent in the child. This allows threads in the child to continue processing and
 12876 allows locks and the state to be preserved without explicit *pthread_atfork()* code. The calling
 12877 process has to ensure that the threads processing state that is shared between the parent and
 12878 child (that is, file descriptors or MAP_SHARED memory) behaves properly after *forkall()*. For
 12879 example, if a thread is reading a file descriptor in the parent when *forkall()* is called, then two
 12880 threads (one in the parent and one in the child) are reading the file descriptor after the *forkall()*.
 12881 If this is not desired behavior, the parent process has to synchronize with such threads before
 12882 calling *forkall()*.

12883 While the *fork()* function is async-signal-safe, there is no way for an implementation to
 12884 determine whether the fork handlers established by *pthread_atfork()* are async-signal-safe. The
 12885 fork handlers may attempt to execute portions of the implementation that are not async-signal-
 12886 safe, such as those that are protected by mutexes, leading to a deadlock condition. It is therefore
 12887 undefined for the fork handlers to execute functions that are not async-signal-safe when *fork()* is
 12888 called from a signal handler.

12889 When *forkall()* is called, threads, other than the calling thread, that are in functions that can
 12890 return with an [EINTR] error may have those functions return [EINTR] if the implementation
 12891 cannot ensure that the function behaves correctly in the parent and child. In particular,
 12892 *pthread_cond_wait()* and *pthread_cond_timedwait()* need to return in order to ensure that the
 12893 condition has not changed. These functions can be awakened by a spurious condition wakeup
 12894 rather than returning [EINTR].

12895 FUTURE DIRECTIONS

12896 None.

12897 SEE ALSO

12898 *alarm()*, *exec*, *fcntl()*, *posix_trace_attr_getinherited()*, *posix_trace_trid_eventid_open()*,
 12899 *pthread_atfork()*, *semop()*, *signal()*, *times()*, the Base Definitions volume of IEEE Std 1003.1-2001,
 12900 <sys/types.h>, <unistd.h>

12901 **CHANGE HISTORY**

12902 First released in Issue 1. Derived from Issue 1 of the SVID.

12903 **Issue 5**

12904 The DESCRIPTION is changed for alignment with the POSIX Realtime Extension and the POSIX
12905 Threads Extension.

12906 **Issue 6**

12907 The following new requirements on POSIX implementations derive from alignment with the
12908 Single UNIX Specification:

- 12909 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
12910 required for conforming implementations of previous POSIX specifications, it was not
12911 required for UNIX applications.

12912 The following changes were made to align with the IEEE P1003.1a draft standard:

- 12913 • The effect of `fork()` on a pending alarm call in the child process is clarified.

12914 The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.

12915 The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

12916 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/17 is applied, adding text to the |
12917 DESCRIPTION and RATIONALE relating to fork handlers registered by the `pthread_atfork()` |
12918 function and async-signal safety. |

12919 **NAME**

12920 fpathconf, pathconf — get configurable pathname variables

12921 **SYNOPSIS**

12922 #include <unistd.h>

12923 long fpathconf(int *fildev*, int *name*);12924 long pathconf(const char **path*, int *name*);12925 **DESCRIPTION**12926 The *fpathconf()* and *pathconf()* functions shall determine the current value of a configurable limit
12927 or option (*variable*) that is associated with a file or directory.12928 For *pathconf()*, the *path* argument points to the pathname of a file or directory.12929 For *fpathconf()*, the *fildev* argument is an open file descriptor.12930 The *name* argument represents the variable to be queried relative to that file or directory.
12931 Implementations shall support all of the variables listed in the following table and may support
12932 others. The variables in the following table come from <limits.h> or <unistd.h> and the
12933 symbolic constants, defined in <unistd.h>, are the corresponding values used for *name*.

12934

12935

12936

12937

12938

12939

12940

12941

12942

12943

12944

12945

12946

12947

12948

12949

12950

12951

12952

12953

12954

Variable	Value of <i>name</i>	Requirements
{FILESIZEBITS}	_PC_FILESIZEBITS	3, 4
{LINK_MAX}	_PC_LINK_MAX	1
{MAX_CANON}	_PC_MAX_CANON	2
{MAX_INPUT}	_PC_MAX_INPUT	2
{NAME_MAX}	_PC_NAME_MAX	3, 4
{PATH_MAX}	_PC_PATH_MAX	4, 5
{PIPE_BUF}	_PC_PIPE_BUF	6
{POSIX_ALLOC_SIZE_MIN}	_PC_ALLOC_SIZE_MIN	
{POSIX_REC_INCR_XFER_SIZE}	_PC_REC_INCR_XFER_SIZE	
{POSIX_REC_MAX_XFER_SIZE}	_PC_REC_MAX_XFER_SIZE	
{POSIX_REC_MIN_XFER_SIZE}	_PC_REC_MIN_XFER_SIZE	
{POSIX_REC_XFER_ALIGN}	_PC_REC_XFER_ALIGN	
{SYMLINK_MAX}	_PC_SYMLINK_MAX	4, 9
_POSIX_CHOWN_RESTRICTED	_PC_CHOWN_RESTRICTED	7
_POSIX_NO_TRUNC	_PC_NO_TRUNC	3, 4
_POSIX_VDISABLE	_PC_VDISABLE	2
_POSIX_ASYNC_IO	_PC_ASYNC_IO	8
_POSIX_PRIO_IO	_PC_PRIO_IO	8
_POSIX_SYNC_IO	_PC_SYNC_IO	8

12955

Requirements

12956

1. If *path* or *fildev* refers to a directory, the value returned shall apply to the directory itself.

12957

12958

2. If *path* or *fildev* does not refer to a terminal file, it is unspecified whether an implementation supports an association of the variable name with the specified file.

12959

12960

3. If *path* or *fildev* refers to a directory, the value returned shall apply to filenames within the directory.

12961

12962

4. If *path* or *fildev* does not refer to a directory, it is unspecified whether an implementation supports an association of the variable name with the specified file.

- 12963 5. If *path* or *filde*s refers to a directory, the value returned shall be the maximum length of a
12964 relative pathname when the specified directory is the working directory.
- 12965 6. If *path* refers to a FIFO, or *filde*s refers to a pipe or FIFO, the value returned shall apply to
12966 the referenced object. If *path* or *filde*s refers to a directory, the value returned shall apply to
12967 any FIFO that exists or can be created within the directory. If *path* or *filde*s refers to any
12968 other type of file, it is unspecified whether an implementation supports an association of
12969 the variable name with the specified file.
- 12970 7. If *path* or *filde*s refers to a directory, the value returned shall apply to any files, other than
12971 directories, that exist or can be created within the directory.
- 12972 8. If *path* or *filde*s refers to a directory, it is unspecified whether an implementation supports
12973 an association of the variable name with the specified file.
- 12974 9. If *path* or *filde*s refers to a directory, the value returned shall be the maximum length of the
12975 string that a symbolic link in that directory can contain.

12976 **RETURN VALUE**

12977 If *name* is an invalid value, both *pathconf*() and *fpathconf*() shall return -1 and set *errno* to
12978 indicate the error.

12979 If the variable corresponding to *name* has no limit for the *path* or file descriptor, both *pathconf*()
12980 and *fpathconf*() shall return -1 without changing *errno*. If the implementation needs to use *path*
12981 to determine the value of *name* and the implementation does not support the association of *name*
12982 with the file specified by *path*, or if the process did not have appropriate privileges to query the
12983 file specified by *path*, or *path* does not exist, *pathconf*() shall return -1 and set *errno* to indicate the
12984 error.

12985 If the implementation needs to use *filde*s to determine the value of *name* and the implementation
12986 does not support the association of *name* with the file specified by *filde*s, or if *filde*s is an invalid
12987 file descriptor, *fpathconf*() shall return -1 and set *errno* to indicate the error.

12988 Otherwise, *pathconf*() or *fpathconf*() shall return the current variable value for the file or
12989 directory without changing *errno*. The value returned shall not be more restrictive than the
12990 corresponding value available to the application when it was compiled with the
12991 implementation's `<limits.h>` or `<unistd.h>`.

12992 **ERRORS**

12993 The *pathconf*() function shall fail if:

- 12994 [EINVAL] The value of *name* is not valid.
- 12995 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
12996 argument.
- 12997 The *pathconf*() function may fail if:
- 12998 [EACCES] Search permission is denied for a component of the path prefix.
- 12999 [EINVAL] The implementation does not support an association of the variable *name* with
13000 the specified file.
- 13001 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
13002 resolution of the *path* argument.
- 13003 [ENAMETOOLONG]
13004 The length of the *path* argument exceeds {PATH_MAX} or a pathname
13005 component is longer than {NAME_MAX}.

- 13006 [ENAMETOOLONG]
 13007 As a result of encountering a symbolic link in resolution of the *path* argument,
 13008 the length of the substituted pathname string exceeded {PATH_MAX}.
- 13009 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- 13010 [ENOTDIR] A component of the path prefix is not a directory.
- 13011 The *fpathconf()* function shall fail if:
- 13012 [EINVAL] The value of *name* is not valid.
- 13013 The *fpathconf()* function may fail if:
- 13014 [EBADF] The *fildev* argument is not a valid file descriptor.
- 13015 [EINVAL] The implementation does not support an association of the variable *name* with
 13016 the specified file.
- 13017 **EXAMPLES**
- 13018 None.
- 13019 **APPLICATION USAGE**
- 13020 None.
- 13021 **RATIONALE**
- 13022 The *pathconf()* function was proposed immediately after the *sysconf()* function when it was
 13023 realized that some configurable values may differ across file system, directory, or device
 13024 boundaries.
- 13025 For example, {NAME_MAX} frequently changes between System V and BSD-based file systems;
 13026 System V uses a maximum of 14, BSD 255. On an implementation that provides both types of file
 13027 systems, an application would be forced to limit all pathname components to 14 bytes, as this
 13028 would be the value specified in <limits.h> on such a system.
- 13029 Therefore, various useful values can be queried on any pathname or file descriptor, assuming
 13030 that the appropriate permissions are in place.
- 13031 The value returned for the variable {PATH_MAX} indicates the longest relative pathname that
 13032 could be given if the specified directory is the process' current working directory. A process may
 13033 not always be able to generate a name that long and use it if a subdirectory in the pathname
 13034 crosses into a more restrictive file system.
- 13035 The value returned for the variable _POSIX_CHOWN_RESTRICTED also applies to directories
 13036 that do not have file systems mounted on them. The value may change when crossing a mount
 13037 point, so applications that need to know should check for each directory. (An even easier check
 13038 is to try the *chown()* function and look for an error in case it happens.)
- 13039 Unlike the values returned by *sysconf()*, the pathname-oriented variables are potentially more
 13040 volatile and are not guaranteed to remain constant throughout the process' lifetime. For
 13041 example, in between two calls to *pathconf()*, the file system in question may have been
 13042 unmounted and remounted with different characteristics.
- 13043 Also note that most of the errors are optional. If one of the variables always has the same value
 13044 on an implementation, the implementation need not look at *path* or *fildev* to return that value and
 13045 is, therefore, not required to detect any of the errors except the meaning of [EINVAL] that
 13046 indicates that the value of *name* is not valid for that variable.
- 13047 If the value of any of the limits is unspecified (logically infinite), they will not be defined in
 13048 <limits.h> and the *pathconf()* and *fpathconf()* functions return -1 without changing *errno*. This
 13049 can be distinguished from the case of giving an unrecognized *name* argument because *errno* is set

- 13050 to [EINVAL] in this case.
- 13051 Since `-1` is a valid return value for the `pathconf()` and `fpathconf()` functions, applications should
13052 set `errno` to zero before calling them and check `errno` only if the return value is `-1`.
- 13053 For the case of `{SYMLINK_MAX}`, since both `pathconf()` and `open()` follow symbolic links, there
13054 is no way that `path` or `filenames` could refer to a symbolic link.
- 13055 **FUTURE DIRECTIONS**
- 13056 None.
- 13057 **SEE ALSO**
- 13058 `confstr()`, `sysconf()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<limits.h>`, `<unistd.h>`,
13059 the Shell and Utilities volume of IEEE Std 1003.1-2001
- 13060 **CHANGE HISTORY**
- 13061 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.
- 13062 **Issue 5**
- 13063 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.
- 13064 Large File Summit extensions are added.
- 13065 **Issue 6**
- 13066 The following new requirements on POSIX implementations derive from alignment with the
13067 Single UNIX Specification:
- 13068 • The DESCRIPTION is updated to include `{FILESIZEBITS}`.
 - 13069 • The `[ELOOP]` mandatory error condition is added.
 - 13070 • A second `[ENAMETOOLONG]` is added as an optional error condition.
- 13071 The following changes were made to align with the IEEE P1003.1a draft standard:
- 13072 • The `_PC_SYMLINK_MAX` entry is added to the table in the DESCRIPTION.
- 13073 The following `pathconf()` variables and their associated names are added for alignment with
13074 IEEE Std 1003.1d-1999:
- 13075 `{POSIX_ALLOC_SIZE_MIN}`
 - 13076 `{POSIX_REC_INCR_XFER_SIZE}`
 - 13077 `{POSIX_REC_MAX_XFER_SIZE}`
 - 13078 `{POSIX_REC_MIN_XFER_SIZE}`
 - 13079 `{POSIX_REC_XFER_ALIGN}`
- 13080 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/18 is applied, changing the fourth
13081 paragraph of the DESCRIPTION and removing shading and margin markers from the table. This
13082 change is needed since implementations are required to support all of these symbols.

13083 **NAME**

13084 fpclassify — classify real floating type

13085 **SYNOPSIS**

13086 #include <math.h>

13087 int fpclassify(real-floating x);

13088 **DESCRIPTION**

13089 cx The functionality described on this reference page is aligned with the ISO C standard. Any
13090 conflict between the requirements described here and the ISO C standard is unintentional. This
13091 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13092 The *fpclassify()* macro shall classify its argument value as NaN, infinite, normal, subnormal,
13093 zero, or into another implementation-defined category. First, an argument represented in a
13094 format wider than its semantic type is converted to its semantic type. Then classification is based
13095 on the type of the argument.

13096 **RETURN VALUE**

13097 The *fpclassify()* macro shall return the value of the number classification macro appropriate to
13098 the value of its argument.

13099 **ERRORS**

13100 No errors are defined.

13101 **EXAMPLES**

13102 None.

13103 **APPLICATION USAGE**

13104 None.

13105 **RATIONALE**

13106 None.

13107 **FUTURE DIRECTIONS**

13108 None.

13109 **SEE ALSO**

13110 *isfinite()*, *isinf()*, *isnan()*, *isnormal()*, *signbit()*, the Base Definitions volume of
13111 IEEE Std 1003.1-2001, <math.h>

13112 **CHANGE HISTORY**

13113 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

13114 NAME

13115 fprintf, printf, snprintf, sprintf — print formatted output

13116 SYNOPSIS

13117 #include <stdio.h>

13118 int fprintf(FILE *restrict *stream*, const char *restrict *format*, ...);13119 int printf(const char *restrict *format*, ...);13120 int snprintf(char *restrict *s*, size_t *n*,13121 const char *restrict *format*, ...);13122 int sprintf(char *restrict *s*, const char *restrict *format*, ...);

13123 DESCRIPTION

13124 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 13125 conflict between the requirements described here and the ISO C standard is unintentional. This
 13126 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13127 The *fprintf()* function shall place output on the named output *stream*. The *printf()* function shall
 13128 place output on the standard output stream *stdout*. The *sprintf()* function shall place output
 13129 followed by the null byte, '\0', in consecutive bytes starting at *s*; it is the user's responsibility
 13130 to ensure that enough space is available.

13131 The *snprintf()* function shall be equivalent to *sprintf()*, with the addition of the *n* argument
 13132 which states the size of the buffer referred to by *s*. If *n* is zero, nothing shall be written and *s* may
 13133 be a null pointer. Otherwise, output bytes beyond the *n*-1st shall be discarded instead of being
 13134 written to the array, and a null byte is written at the end of the bytes actually written into the
 13135 array.

13136 If copying takes place between objects that overlap as a result of a call to *sprintf()* or *snprintf()*,
 13137 the results are undefined.

13138 Each of these functions converts, formats, and prints its arguments under control of the *format*.
 13139 The *format* is a character string, beginning and ending in its initial shift state, if any. The *format* is
 13140 composed of zero or more directives: *ordinary characters*, which are simply copied to the output
 13141 stream, and *conversion specifications*, each of which shall result in the fetching of zero or more
 13142 arguments. The results are undefined if there are insufficient arguments for the *format*. If the
 13143 *format* is exhausted while arguments remain, the excess arguments shall be evaluated but are
 13144 otherwise ignored.

13145 XSI Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than
 13146 to the next unused argument. In this case, the conversion specifier character % (see below) is
 13147 replaced by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL_ARGMAX}],
 13148 giving the position of the argument in the argument list. This feature provides for the definition
 13149 of format strings that select arguments in an order appropriate to specific languages (see the
 13150 EXAMPLES section).

13151 The *format* can contain either numbered argument conversion specifications (that is, "%n\$" and
 13152 "*m\$"), or unnumbered argument conversion specifications (that is, % and *), but not both. The
 13153 only exception to this is that %% can be mixed with the "%n\$" form. The results of mixing
 13154 numbered and unnumbered argument specifications in a *format* string are undefined. When
 13155 numbered argument specifications are used, specifying the *N*th argument requires that all the
 13156 leading arguments, from the first to the (*N*-1)th, are specified in the format string.

13157 In format strings containing the "%n\$" form of conversion specification, numbered arguments
 13158 in the argument list can be referenced from the format string as many times as required.

13159 In format strings containing the % form of conversion specification, each conversion specification
 13160 uses the first unused argument in the argument list.

- 13161 CX All forms of the *fprintf()* functions allow for the insertion of a language-dependent radix character in the output string. The radix character is defined in the program's locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a period ('.').
- 13162
- 13163
- 13164
- 13165 XSI Each conversion specification is introduced by the '%' character or by the character sequence "%n\$", after which the following appear in sequence:
- 13166
- 13167 • Zero or more *flags* (in any order), which modify the meaning of the conversion specification.
 - 13168 • An optional minimum *field width*. If the converted value has fewer bytes than the field width, it shall be padded with spaces by default on the left; it shall be padded on the right if the left-adjustment flag ('-'), described below, is given to the field width. The field width takes the form of an asterisk ('*'), described below, or a decimal integer.
 - 13169
 - 13170
 - 13171
 - 13172 • An optional *precision* that gives the minimum number of digits to appear for the d, i, o, u, x, and X conversion specifiers; the number of digits to appear after the radix character for the a, A, e, E, f, and F conversion specifiers; the maximum number of significant digits for the g and G conversion specifiers; or the maximum number of bytes to be printed from a string in the s and S conversion specifiers. The precision takes the form of a period ('.') followed either by an asterisk ('*'), described below, or an optional decimal digit string, where a null digit string is treated as zero. If a precision appears with any other conversion specifier, the behavior is undefined.
 - 13173
 - 13174
 - 13175
 - 13176 XSI
 - 13177
 - 13178
 - 13179
 - 13180 • An optional length modifier that specifies the size of the argument.
 - 13181 • A *conversion specifier* character that indicates the type of conversion to be applied.
- 13182 A field width, or precision, or both, may be indicated by an asterisk ('*'). In this case an argument of type *int* supplies the field width or precision. Applications shall ensure that arguments specifying field width, or precision, or both appear in that order before the argument, if any, to be converted. A negative field width is taken as a '-' flag followed by a positive field width. A negative precision is taken as if the precision were omitted. In format strings containing the "%n\$" form of a conversion specification, a field width or precision may be indicated by the sequence "*m\$", where *m* is a decimal integer in the range [1,{NL_ARGMAX}] giving the position in the argument list (after the *format* argument) of an integer argument containing the field width or precision, for example:
- 13183
- 13184
- 13185
- 13186 XSI
- 13187
- 13188
- 13189
- 13190
- ```
13191 printf("%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```
- 13192 The flag characters and their meanings are:
- 13193 XSI ' The integer portion of the result of a decimal conversion (%i, %d, %u, %f, %F, %g, or %G) shall be formatted with thousands' grouping characters. For other conversions the behavior is undefined. The non-monetary grouping character is used.
  - 13194
  - 13195
  - 13196 - The result of the conversion shall be left-justified within the field. The conversion is right-justified if this flag is not specified.
  - 13197
  - 13198 + The result of a signed conversion shall always begin with a sign ('+' or '-'). The conversion shall begin with a sign only when a negative value is converted if this flag is not specified.
  - 13199
  - 13200
  - 13201 <space> If the first character of a signed conversion is not a sign or if a signed conversion results in no characters, a <space> shall be prefixed to the result. This means that if the <space> and '+' flags both appear, the <space> flag shall be ignored.
  - 13202
  - 13203
  - 13204 # Specifies that the value is to be converted to an alternative form. For o conversion, it increases the precision (if necessary) to force the first digit of the result to be zero. For x
  - 13205

13206 or X conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For a, A,  
 13207 e, E, f, F, g, and G conversion specifiers, the result shall always contain a radix  
 13208 character, even if no digits follow the radix character. Without this flag, a radix  
 13209 character appears in the result of these conversions only if a digit follows it. For g and G  
 13210 conversion specifiers, trailing zeros shall *not* be removed from the result as they  
 13211 normally are. For other conversion specifiers, the behavior is undefined.

13212 0 For d, i, o, u, x, X, a, A, e, E, f, F, g, and G conversion specifiers, leading zeros  
 13213 (following any indication of sign or base) are used to pad to the field width; no space  
 13214 padding is performed. If the '0' and '-' flags both appear, the '0' flag is ignored. For  
 13215 d, i, o, u, x, and X conversion specifiers, if a precision is specified, the '0' flag is  
 13216 XSI ignored. If the '0' and '' flags both appear, the grouping characters are inserted  
 13217 before zero padding. For other conversions, the behavior is undefined.

13218 The length modifiers and their meanings are:

13219 hh Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **signed char**  
 13220 or **unsigned char** argument (the argument will have been promoted according to the  
 13221 integer promotions, but its value shall be converted to **signed char** or **unsigned char**  
 13222 before printing); or that a following n conversion specifier applies to a pointer to a  
 13223 **signed char** argument.

13224 h Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **short** or  
 13225 **unsigned short** argument (the argument will have been promoted according to the  
 13226 integer promotions, but its value shall be converted to **short** or **unsigned short** before  
 13227 printing); or that a following n conversion specifier applies to a pointer to a **short**  
 13228 argument.

13229 l (ell) Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **long** or  
 13230 **unsigned long** argument; that a following n conversion specifier applies to a pointer to  
 13231 a **long** argument; that a following c conversion specifier applies to a **wint\_t** argument;  
 13232 that a following s conversion specifier applies to a pointer to a **wchar\_t** argument; or  
 13233 has no effect on a following a, A, e, E, f, F, g, or G conversion specifier.

13234 ll (ell-ell) Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **long long** or  
 13235 **unsigned long long** argument; or that a following n conversion specifier applies to a  
 13236 pointer to a **long long** argument.

13238 j Specifies that a following d, i, o, u, x, or X conversion specifier applies to an **intmax\_t**  
 13239 or **uintmax\_t** argument; or that a following n conversion specifier applies to a pointer  
 13240 to an **intmax\_t** argument.

13241 z Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **size\_t** or the  
 13242 corresponding signed integer type argument; or that a following n conversion specifier  
 13243 applies to a pointer to a signed integer type corresponding to a **size\_t** argument.

13244 t Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **ptrdiff\_t** or  
 13245 the corresponding **unsigned** type argument; or that a following n conversion specifier  
 13246 applies to a pointer to a **ptrdiff\_t** argument.

13247 L Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to a **long**  
 13248 **double** argument.

13249 If a length modifier appears with any conversion specifier other than as specified above, the  
 13250 behavior is undefined.

|       |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 13251 |      | The conversion specifiers and their meanings are:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 13252 | d, i | The <b>int</b> argument shall be converted to a signed decimal in the style " <code>[-] dddd</code> ". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.                                                                                                                                                                                                                                                                                         |
| 13253 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13254 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13255 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13256 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13257 | o    | The <b>unsigned</b> argument shall be converted to unsigned octal format in the style " <code>ddd</code> ". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.                                                                                                                                                                                                                                                                                    |
| 13258 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13259 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13260 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13261 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13262 | u    | The <b>unsigned</b> argument shall be converted to unsigned decimal format in the style " <code>ddd</code> ". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.                                                                                                                                                                                                                                                                                  |
| 13263 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13264 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13265 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13266 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13267 | x    | The <b>unsigned</b> argument shall be converted to unsigned hexadecimal format in the style " <code>ddd</code> "; the letters "abcdef" are used. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.                                                                                                                                                                                                                                               |
| 13268 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13269 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13270 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13271 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13272 | X    | Equivalent to the x conversion specifier, except that letters "ABCDEF" are used instead of "abcdef".                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 13273 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13274 | f, F | The <b>double</b> argument shall be converted to decimal notation in the style " <code>[-] ddd.ddd</code> ", where the number of digits after the radix character is equal to the precision specification. If the precision is missing, it shall be taken as 6; if the precision is explicitly zero and no '#' flag is present, no radix character shall appear. If a radix character appears, at least one digit appears before it. The low-order digit shall be rounded in an implementation-defined manner.                                                                                                                                                                                |
| 13275 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13276 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13277 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13278 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13279 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13280 |      | A <b>double</b> argument representing an infinity shall be converted in one of the styles " <code>[-] inf</code> " or " <code>[-] infinity</code> "; which style is implementation-defined. A <b>double</b> argument representing a NaN shall be converted in one of the styles " <code>[-] nan(n-char-sequence)</code> " or " <code>[-] nan</code> "; which style, and the meaning of any <i>n-char-sequence</i> , is implementation-defined. The F conversion specifier produces "INF", "INFINITY", or "NAN" instead of "inf", "infinity", or "nan", respectively.                                                                                                                          |
| 13281 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13282 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13283 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13284 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13285 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13286 | e, E | The <b>double</b> argument shall be converted in the style " <code>[-] d.ddde±dd</code> ", where there is one digit before the radix character (which is non-zero if the argument is non-zero) and the number of digits after it is equal to the precision; if the precision is missing, it shall be taken as 6; if the precision is zero and no '#' flag is present, no radix character shall appear. The low-order digit shall be rounded in an implementation-defined manner. The E conversion specifier shall produce a number with 'E' instead of 'e' introducing the exponent. The exponent shall always contain at least two digits. If the value is zero, the exponent shall be zero. |
| 13287 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13288 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13289 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13290 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13291 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13292 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13293 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13294 |      | A <b>double</b> argument representing an infinity or NaN shall be converted in the style of an f or F conversion specifier.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13295 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 13296 | g, G | The <b>double</b> argument shall be converted in the style f or e (or in the style F or E in the case of a G conversion specifier), with the precision specifying the number of significant                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 13297 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

|       |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 13298 |      | digits. If an explicit precision is zero, it shall be taken as 1. The style used depends on the value converted; style <i>e</i> (or <i>E</i> ) shall be used only if the exponent resulting from such a conversion is less than $-4$ or greater than or equal to the precision. Trailing zeros shall be removed from the fractional portion of the result; a radix character shall appear only if it is followed by a digit or a '#' flag is present.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 13299 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13300 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13301 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13302 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13303 |      | A <b>double</b> argument representing an infinity or NaN shall be converted in the style of an <i>f</i> or <i>F</i> conversion specifier.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 13304 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13305 | a, A | A <b>double</b> argument representing a floating-point number shall be converted in the style " <i>[-] 0xh.hhhhp±d</i> ", where there is one hexadecimal digit (which shall be non-zero if the argument is a normalized floating-point number and is otherwise unspecified) before the decimal-point character and the number of hexadecimal digits after it is equal to the precision; if the precision is missing and FLT_RADIX is a power of 2, then the precision shall be sufficient for an exact representation of the value; if the precision is missing and FLT_RADIX is not a power of 2, then the precision shall be sufficient to distinguish values of type <b>double</b> , except that trailing zeros may be omitted; if the precision is zero and the '#' flag is not specified, no decimal-point character shall appear. The letters "abcdef" shall be used for a conversion and the letters "ABCDEF" for A conversion. The A conversion specifier produces a number with 'X' and 'P' instead of 'x' and 'p'. The exponent shall always contain at least one digit, and only as many more digits as necessary to represent the decimal exponent of 2. If the value is zero, the exponent shall be zero. |
| 13306 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13307 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13308 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13309 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13310 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13311 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13312 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13313 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13314 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13315 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13316 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13317 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13318 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13319 |      | A <b>double</b> argument representing an infinity or NaN shall be converted in the style of an <i>f</i> or <i>F</i> conversion specifier.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 13320 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13321 | c    | The <b>int</b> argument shall be converted to an <b>unsigned char</b> , and the resulting byte shall be written.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 13322 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13323 |      | If an <i>l</i> ( <i>ell</i> ) qualifier is present, the <b>wint_t</b> argument shall be converted as if by an <i>ls</i> conversion specification with no precision and an argument that points to a two-element array of type <b>wchar_t</b> , the first element of which contains the <b>wint_t</b> argument to the <i>ls</i> conversion specification and the second element contains a null wide character.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 13324 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13325 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13326 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13327 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13328 | s    | The argument shall be a pointer to an array of <b>char</b> . Bytes from the array shall be written up to (but not including) any terminating null byte. If the precision is specified, no more than that many bytes shall be written. If the precision is not specified or is greater than the size of the array, the application shall ensure that the array contains a null byte.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 13329 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13330 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13331 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13332 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13333 |      | If an <i>l</i> ( <i>ell</i> ) qualifier is present, the argument shall be a pointer to an array of type <b>wchar_t</b> . Wide characters from the array shall be converted to characters (each as if by a call to the <i>wcrtomb()</i> function, with the conversion state described by an <b>mbstate_t</b> object initialized to zero before the first wide character is converted) up to and including a terminating null wide character. The resulting characters shall be written up to (but not including) the terminating null character (byte). If no precision is specified, the application shall ensure that the array contains a null wide character. If a precision is specified, no more than that many characters (bytes) shall be written (including shift sequences, if any), and the array shall contain a null wide character if, to equal the character sequence length given by the precision, the function would need to access a wide character one past the end of the array. In no case shall a partial character be written.                                                                                                                                                                  |
| 13334 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13335 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13336 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13337 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13338 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13339 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13340 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13341 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13342 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13343 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 13344 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

|           |                     |                                                                                                                                                                                          |
|-----------|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 13345     | p                   | The argument shall be a pointer to <b>void</b> . The value of the pointer is converted to a                                                                                              |
| 13346     |                     | sequence of printable characters, in an implementation-defined manner.                                                                                                                   |
| 13347     | n                   | The argument shall be a pointer to an integer into which is written the number of bytes                                                                                                  |
| 13348     |                     | written to the output so far by this call to one of the <i>fprintf()</i> functions. No argument is                                                                                       |
| 13349     |                     | converted.                                                                                                                                                                               |
| 13350 XSI | C                   | Equivalent to <code>lc</code> .                                                                                                                                                          |
| 13351 XSI | S                   | Equivalent to <code>ls</code> .                                                                                                                                                          |
| 13352     | %                   | Print a '%' character; no argument is converted. The complete conversion specification                                                                                                   |
| 13353     |                     | shall be <code>%%</code> .                                                                                                                                                               |
| 13354     |                     | If a conversion specification does not match one of the above forms, the behavior is undefined. If                                                                                       |
| 13355     |                     | any argument is not the correct type for the corresponding conversion specification, the                                                                                                 |
| 13356     |                     | behavior is undefined.                                                                                                                                                                   |
| 13357     |                     | In no case shall a nonexistent or small field width cause truncation of a field; if the result of a                                                                                      |
| 13358     |                     | conversion is wider than the field width, the field shall be expanded to contain the conversion                                                                                          |
| 13359     |                     | result. Characters generated by <i>fprintf()</i> and <i>printf()</i> are printed as if <i>fputc()</i> had been called.                                                                   |
| 13360     |                     | For the <code>a</code> and <code>A</code> conversion specifiers, if <code>FLT_RADIX</code> is a power of 2, the value shall be correctly                                                 |
| 13361     |                     | rounded to a hexadecimal floating number with the given precision.                                                                                                                       |
| 13362     |                     | For <code>a</code> and <code>A</code> conversions, if <code>FLT_RADIX</code> is not a power of 2 and the result is not exactly                                                           |
| 13363     |                     | representable in the given precision, the result should be one of the two adjacent numbers in                                                                                            |
| 13364     |                     | hexadecimal floating style with the given precision, with the extra stipulation that the error                                                                                           |
| 13365     |                     | should have a correct sign for the current rounding direction.                                                                                                                           |
| 13366     |                     | For the <code>e</code> , <code>E</code> , <code>f</code> , <code>F</code> , <code>g</code> , and <code>G</code> conversion specifiers, if the number of significant decimal digits is at |
| 13367     |                     | most <code>DECIMAL_DIG</code> , then the result should be correctly rounded. If the number of significant                                                                                |
| 13368     |                     | decimal digits is more than <code>DECIMAL_DIG</code> but the source value is exactly representable with                                                                                  |
| 13369     |                     | <code>DECIMAL_DIG</code> digits, then the result should be an exact representation with trailing zeros.                                                                                  |
| 13370     |                     | Otherwise, the source value is bounded by two adjacent decimal strings $L < U$ , both having                                                                                             |
| 13371     |                     | <code>DECIMAL_DIG</code> significant digits; the value of the resultant decimal string $D$ should satisfy $L \leq$                                                                       |
| 13372     |                     | $D \leq U$ , with the extra stipulation that the error should have a correct sign for the current                                                                                        |
| 13373     |                     | rounding direction.                                                                                                                                                                      |
| 13374 CX  |                     | The <code>st_ctime</code> and <code>st_mtime</code> fields of the file shall be marked for update between the call to a                                                                  |
| 13375     |                     | successful execution of <i>fprintf()</i> or <i>printf()</i> and the next successful completion of a call to <i>fflush()</i>                                                              |
| 13376     |                     | or <i>fclose()</i> on the same stream or a call to <i>exit()</i> or <i>abort()</i> .                                                                                                     |
| 13377     | <b>RETURN VALUE</b> |                                                                                                                                                                                          |
| 13378     |                     | Upon successful completion, the <i>fprintf()</i> and <i>printf()</i> functions shall return the number of bytes                                                                          |
| 13379     |                     | transmitted.                                                                                                                                                                             |
| 13380     |                     | Upon successful completion, the <i>sprintf()</i> function shall return the number of bytes written to <code>s</code> ,                                                                   |
| 13381     |                     | excluding the terminating null byte.                                                                                                                                                     |
| 13382     |                     | Upon successful completion, the <i>snprintf()</i> function shall return the number of bytes that would                                                                                   |
| 13383     |                     | be written to <code>s</code> had <code>n</code> been sufficiently large excluding the terminating null byte.                                                                             |
| 13384     |                     | If an output error was encountered, these functions shall return a negative value.                                                                                                       |
| 13385     |                     | If the value of <code>n</code> is zero on a call to <i>snprintf()</i> , nothing shall be written, the number of bytes that                                                               |
| 13386     |                     | would have been written had <code>n</code> been sufficiently large excluding the terminating null shall be                                                                               |
| 13387     |                     | returned, and <code>s</code> may be a null pointer.                                                                                                                                      |

13388 **ERRORS**

13389 For the conditions under which *fprintf()* and *printf()* fail and may fail, refer to *fputc()* or  
13390 *fputwc()*.

13391 In addition, all forms of *fprintf()* may fail if:

13392 XSI [EILSEQ] A wide-character code that does not correspond to a valid character has been  
13393 detected.

13394 XSI [EINVAL] There are insufficient arguments.

13395 The *printf()* and *fprintf()* functions may fail if:

13396 XSI [ENOMEM] Insufficient storage space is available.

13397 The *snprintf()* function shall fail if:

13398 XSI [EOVERFLOW] The value of *n* is greater than {INT\_MAX} or the number of bytes needed to  
13399 hold the output excluding the terminating null is greater than {INT\_MAX}.

13400 **EXAMPLES**13401 **Printing Language-Independent Date and Time**

13402 The following statement can be used to print date and time using a language-independent  
13403 format:

13404 `printf(format, weekday, month, day, hour, min);`

13405 For American usage, *format* could be a pointer to the following string:

13406 `"%s, %s %d, %d:%.2d\n"`

13407 This example would produce the following message:

13408 `Sunday, July 3, 10:02`

13409 For German usage, *format* could be a pointer to the following string:

13410 `"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"`

13411 This definition of *format* would produce the following message:

13412 `Sonntag, 3. Juli, 10:02`

13413 **Printing File Information**

13414 The following example prints information about the type, permissions, and number of links of a  
13415 specific file in a directory.

13416 The first two calls to *printf()* use data decoded from a previous *stat()* call. The user-defined  
13417 *strperm()* function shall return a string similar to the one at the beginning of the output for the  
13418 following command:

13419 `ls -l`

13420 The next call to *printf()* outputs the owner's name if it is found using *getpwuid()*; the *getpwuid()*  
13421 function shall return a **passwd** structure from which the name of the user is extracted. If the user  
13422 name is not found, the program instead prints out the numeric value of the user ID.

13423 The next call prints out the group name if it is found using *getgrgid()*; *getgrgid()* is very similar to  
13424 *getpwuid()* except that it shall return group information based on the group number. Once  
13425 again, if the group is not found, the program prints the numeric value of the group for the entry.

```

13426 The final call to printf() prints the size of the file.
13427 #include <stdio.h>
13428 #include <sys/types.h>
13429 #include <pwd.h>
13430 #include <grp.h>
13431 char *strperm (mode_t);
13432 ...
13433 struct stat statbuf;
13434 struct passwd *pwd;
13435 struct group *grp;
13436 ...
13437 printf("%10.10s", strperm (statbuf.st_mode));
13438 printf("%4d", statbuf.st_nlink);
13439 if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
13440 printf(" %-8.8s", pwd->pw_name);
13441 else
13442 printf(" %-8ld", (long) statbuf.st_uid);
13443 if ((grp = getgrgid(statbuf.st_gid)) != NULL)
13444 printf(" %-8.8s", grp->gr_name);
13445 else
13446 printf(" %-8ld", (long) statbuf.st_gid);
13447 printf("%9jd", (intmax_t) statbuf.st_size);
13448 ...

```

#### 13449 **Printing a Localized Date String**

13450 The following example gets a localized date string. The *nl\_langinfo()* function shall return the  
 13451 localized date string, which specifies the order and layout of the date. The *strftime()* function  
 13452 takes this information and, using the **tm** structure for values, places the date and time  
 13453 information into *datestring*. The *printf()* function then outputs *datestring* and the name of the  
 13454 entry.

```

13455 #include <stdio.h>
13456 #include <time.h>
13457 #include <langinfo.h>
13458 ...
13459 struct dirent *dp;
13460 struct tm *tm;
13461 char datestring[256];
13462 ...
13463 strftime(datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
13464 printf(" %s %s\n", datestring, dp->d_name);
13465 ...

```

13466 **Printing Error Information**

13467 The following example uses *fprintf()* to write error information to standard error.

13468 In the first group of calls, the program tries to open the password lock file named **LOCKFILE**. If  
 13469 the file already exists, this is an error, as indicated by the **O\_EXCL** flag on the *open()* function. If  
 13470 the call fails, the program assumes that someone else is updating the password file, and the  
 13471 program exits.

13472 The next group of calls saves a new password file as the current password file by creating a link  
 13473 between **LOCKFILE** and the new password file **PASSWDFILE**.

```

13474 #include <sys/types.h>
13475 #include <sys/stat.h>
13476 #include <fcntl.h>
13477 #include <stdio.h>
13478 #include <stdlib.h>
13479 #include <unistd.h>
13480 #include <string.h>
13481 #include <errno.h>

13482 #define LOCKFILE "/etc/ptmp"
13483 #define PASSWDFILE "/etc/passwd"
13484 ...
13485 int pfd;
13486 ...
13487 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
13488 S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
13489 {
13490 fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
13491 exit(1);
13492 }
13493 ...
13494 if (link(LOCKFILE, PASSWDFILE) == -1) {
13495 fprintf(stderr, "Link error: %s\n", strerror(errno));
13496 exit(1);
13497 }
13498 ...

```

13499 **Printing Usage Information**

13500 The following example checks to make sure the program has the necessary arguments, and uses  
 13501 *fprintf()* to print usage information if the expected number of arguments is not present.

```

13502 #include <stdio.h>
13503 #include <stdlib.h>
13504 ...
13505 char *Options = "hdbt1";
13506 ...
13507 if (argc < 2) {
13508 fprintf(stderr, "Usage: %s -%s <file\n", argv[0], Options); exit(1);
13509 }
13510 ...

```

13511 **Formatting a Decimal String**

13512 The following example prints a key and data pair on *stdout*. Note use of the '\*' (asterisk) in the  
 13513 format string; this ensures the correct number of decimal places for the element based on the  
 13514 number of elements requested.

```
13515 #include <stdio.h>
13516 ...
13517 long i;
13518 char *keyst;
13519 int elementlen, len;
13520 ...
13521 while (len < elementlen) {
13522 ...
13523 printf("%s Element%0*ld\n", keyst, elementlen, i);
13524 ...
13525 }
```

13526 **Creating a Filename**

13527 The following example creates a filename using information from a previous *getpwnam()*  
 13528 function that returned the HOME directory of the user.

```
13529 #include <stdio.h>
13530 #include <sys/types.h>
13531 #include <unistd.h>
13532 ...
13533 char filename[PATH_MAX+1];
13534 struct passwd *pw;
13535 ...
13536 sprintf(filename, "%s/%d.out", pw->pw_dir, getpid());
13537 ...
```

13538 **Reporting an Event**

13539 The following example loops until an event has timed out. The *pause()* function waits forever  
 13540 unless it receives a signal. The *fprintf()* statement should never occur due to the possible return  
 13541 values of *pause()*.

```
13542 #include <stdio.h>
13543 #include <unistd.h>
13544 #include <string.h>
13545 #include <errno.h>
13546 ...
13547 while (!event_complete) {
13548 ...
13549 if (pause() != -1 || errno != EINTR)
13550 fprintf(stderr, "pause: unknown error: %s\n", strerror(errno));
13551 }
13552 ...
```

13553 **Printing Monetary Information**

13554 The following example uses *strfmon()* to convert a number and store it as a formatted monetary  
 13555 string named *convbuf*. If the first number is printed, the program prints the format and the  
 13556 description; otherwise, it just prints the number.

```

13557 #include <monetary.h>
13558 #include <stdio.h>
13559 ...
13560 struct tblfmt {
13561 char *format;
13562 char *description;
13563 };
13564 struct tblfmt table[] = {
13565 { "%n", "default formatting" },
13566 { "%11n", "right align within an 11 character field" },
13567 { "%#5n", "aligned columns for values up to 99999" },
13568 { "%=*#5n", "specify a fill character" },
13569 { "%=0#5n", "fill characters do not use grouping" },
13570 { "%^#5n", "disable the grouping separator" },
13571 { "%^#5.0n", "round off to whole units" },
13572 { "%^#5.4n", "increase the precision" },
13573 { "%(#5n", "use an alternative pos/neg style" },
13574 { "%!(#5n", "disable the currency symbol" },
13575 };
13576 ...
13577 float input[3];
13578 int i, j;
13579 char convbuf[100];
13580 ...
13581 strfmon(convbuf, sizeof(convbuf), table[i].format, input[j]);
13582 if (j == 0) {
13583 printf("%s%s%s\n", table[i].format,
13584 convbuf, table[i].description);
13585 }
13586 else {
13587 printf("%s\n", convbuf);
13588 }
13589 ...

```

13590 **Printing Wide Characters**

13591 The following example prints a series of wide characters. Suppose that "L'@'" expands to three  
 13592 bytes:

```

13593 wchar_t wz [3] = L"@@"; // Zero-terminated
13594 wchar_t wn [3] = L"@@"; // Unterminated
13595 fprintf (stdout, "%ls", wz); // Outputs 6 bytes
13596 fprintf (stdout, "%ls", wn); // Undefined because wn has no terminator
13597 fprintf (stdout, "%4ls", wz); // Outputs 3 bytes
13598 fprintf (stdout, "%4ls", wn); // Outputs 3 bytes; no terminator needed
13599 fprintf (stdout, "%9ls", wz); // Outputs 6 bytes

```

```

13600 fprintf (stdout,"%9ls", wn); // Outputs 9 bytes; no terminator needed
13601 fprintf (stdout,"%10ls", wz); // Outputs 6 bytes
13602 fprintf (stdout,"%10ls", wn); // Undefined because wn has no terminator

```

13603 In the last line of the example, after processing three characters, nine bytes have been output.  
 13604 The fourth character must then be examined to determine whether it converts to one byte or  
 13605 more. If it converts to more than one byte, the output is only nine bytes. Since there is no fourth  
 13606 character in the array, the behavior is undefined.

#### 13607 APPLICATION USAGE

13608 If the application calling *fprintf()* has any objects of type **wint\_t** or **wchar\_t**, it must also include  
 13609 the **<wchar.h>** header to have these objects defined.

#### 13610 RATIONALE

13611 None.

#### 13612 FUTURE DIRECTIONS

13613 None.

#### 13614 SEE ALSO

13615 *fputc()*, *fscanf()*, *setlocale()*, *strfmon()*, *wcrtomb()*, the Base Definitions volume of  
 13616 IEEE Std 1003.1-2001, Chapter 7, Locale, **<stdio.h>**, **<wchar.h>**

#### 13617 CHANGE HISTORY

13618 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 13619 Issue 5

13620 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the **l** (ell) qualifier can  
 13621 now be used with **c** and **s** conversion specifiers.

13622 The *snprintf()* function is new in Issue 5.

#### 13623 Issue 6

13624 Extensions beyond the ISO C standard are marked.

13625 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

13626 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

13627 • The prototypes for *fprintf()*, *printf()*, *snprintf()*, and *sprintf()* are updated, and the XSI  
 13628 shading is removed from *snprintf()*.

13629 • The description of *snprintf()* is aligned with the ISO C standard. Note that this supersedes  
 13630 the *snprintf()* description in The Open Group Base Resolution bwg98-006, which changed the  
 13631 behavior from Issue 5.

13632 • The DESCRIPTION is updated.

13633 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion  
 13634 specification” consistently.

13635 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

13636 An example of printing wide characters is added.

13637 **NAME**

13638 fputc — put a byte on a stream

13639 **SYNOPSIS**

13640 #include &lt;stdio.h&gt;

13641 int fputc(int *c*, FILE \**stream*);13642 **DESCRIPTION**

13643 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 13644 conflict between the requirements described here and the ISO C standard is unintentional. This  
 13645 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13646 The *fputc()* function shall write the byte specified by *c* (converted to an **unsigned char**) to the  
 13647 output stream pointed to by *stream*, at the position indicated by the associated file-position  
 13648 indicator for the stream (if defined), and shall advance the indicator appropriately. If the file  
 13649 cannot support positioning requests, or if the stream was opened with append mode, the byte  
 13650 shall be appended to the output stream.

13651 CX The *st\_ctime* and *st\_mtime* fields of the file shall be marked for update between the successful  
 13652 execution of *fputc()* and the next successful completion of a call to *fflush()* or *fclose()* on the same  
 13653 stream or a call to *exit()* or *abort()*.

13654 **RETURN VALUE**

13655 Upon successful completion, *fputc()* shall return the value it has written. Otherwise, it shall  
 13656 CX return EOF, the error indicator for the stream shall be set, and *errno* shall be set to indicate the  
 13657 error.

13658 **ERRORS**

13659 The *fputc()* function shall fail if either the *stream* is unbuffered or the *stream*'s buffer needs to be  
 13660 flushed, and:

13661 CX [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor underlying *stream* and the  
 13662 process would be delayed in the write operation.

13663 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for  
 13664 writing.

13665 CX [EFBIG] An attempt was made to write to a file that exceeds the maximum file size.

13666 XSI [EFBIG] An attempt was made to write to a file that exceeds the process' file size limit.

13667 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the  
 13668 offset maximum.

13669 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data  
 13670 was transferred.

13671 CX [EIO] A physical I/O error has occurred, or the process is a member of a  
 13672 background process group attempting to write to its controlling terminal,  
 13673 TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the  
 13674 process group of the process is orphaned. This error may also be returned  
 13675 under implementation-defined conditions.

13676 CX [ENOSPC] There was no free space remaining on the device containing the file.

13677 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by  
 13678 any process. A SIGPIPE signal shall also be sent to the thread.

13679 The *fputc()* function may fail if:

13680 CX [ENOMEM] Insufficient storage space is available.

13681 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
13682 capabilities of the device.

13683 **EXAMPLES**

13684 None.

13685 **APPLICATION USAGE**

13686 None.

13687 **RATIONALE**

13688 None.

13689 **FUTURE DIRECTIONS**

13690 None.

13691 **SEE ALSO**

13692 *ferror()*, *fopen()*, *getrlimit()*, *putc()*, *puts()*, *setbuf()*, *ulimit()*, the Base Definitions volume of  
13693 IEEE Std 1003.1-2001, <stdio.h>

13694 **CHANGE HISTORY**

13695 First released in Issue 1. Derived from Issue 1 of the SVID.

13696 **Issue 5**

13697 Large File Summit extensions are added.

13698 **Issue 6**

13699 Extensions beyond the ISO C standard are marked.

13700 The following new requirements on POSIX implementations derive from alignment with the  
13701 Single UNIX Specification:

- 13702
- The [EIO] and [EFBIG] mandatory error conditions are added.
  - The [ENOMEM] and [ENXIO] optional error conditions are added.
- 13703

13704 **NAME**

13705 fputs — put a string on a stream

13706 **SYNOPSIS**

13707 #include &lt;stdio.h&gt;

13708 int fputs(const char \*restrict *s*, FILE \*restrict *stream*);13709 **DESCRIPTION**

13710 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 13711 conflict between the requirements described here and the ISO C standard is unintentional. This  
 13712 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13713 The *fputs()* function shall write the null-terminated string pointed to by *s* to the stream pointed  
 13714 to by *stream*. The terminating null byte shall not be written.

13715 CX The *st\_ctime* and *st\_mtime* fields of the file shall be marked for update between the successful  
 13716 execution of *fputs()* and the next successful completion of a call to *fflush()* or *fclose()* on the same  
 13717 stream or a call to *exit()* or *abort()*.

13718 **RETURN VALUE**

13719 Upon successful completion, *fputs()* shall return a non-negative number. Otherwise, it shall  
 13720 CX return EOF, set an error indicator for the stream, and set *errno* to indicate the error.

13721 **ERRORS**13722 Refer to *fputc()*.13723 **EXAMPLES**13724 **Printing to Standard Output**

13725 The following example gets the current time, converts it to a string using *localtime()* and  
 13726 *asctime()*, and prints it to standard output using *fputs()*. It then prints the number of minutes to  
 13727 an event for which it is waiting.

```

13728 #include <time.h>
13729 #include <stdio.h>
13730 ...
13731 time_t now;
13732 int minutes_to_event;
13733 ...
13734 time(&now);
13735 printf("The time is ");
13736 fputs(asctime(localtime(&now)), stdout);
13737 printf("There are still %d minutes to the event.\n",
13738 minutes_to_event);
13739 ...

```

13740 **APPLICATION USAGE**13741 The *puts()* function appends a <newline> while *fputs()* does not.13742 **RATIONALE**

13743 None.

13744 **FUTURE DIRECTIONS**

13745 None.

13746 **SEE ALSO**

13747 *fopen()*, *putc()*, *puts()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

13748 **CHANGE HISTORY**

13749 First released in Issue 1. Derived from Issue 1 of the SVID.

13750 **Issue 6**

13751 Extensions beyond the ISO C standard are marked.

13752 The *fputs()* prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

## 13753 NAME

13754 fputcw — put a wide-character code on a stream

## 13755 SYNOPSIS

13756 #include &lt;stdio.h&gt;

13757 #include &lt;wchar.h&gt;

13758 wint\_t fputcw(wchar\_t wc, FILE \*stream);

## 13759 DESCRIPTION

13760 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 13761 conflict between the requirements described here and the ISO C standard is unintentional. This  
 13762 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13763 The *fputcw()* function shall write the character corresponding to the wide-character code *wc* to  
 13764 the output stream pointed to by *stream*, at the position indicated by the associated file-position  
 13765 indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot  
 13766 support positioning requests, or if the stream was opened with append mode, the character is  
 13767 appended to the output stream. If an error occurs while writing the character, the shift state of  
 13768 the output file is left in an undefined state.

13769 CX The *st\_ctime* and *st\_mtime* fields of the file shall be marked for update between the successful  
 13770 execution of *fputcw()* and the next successful completion of a call to *fflush()* or *fclose()* on the  
 13771 same stream or a call to *exit()* or *abort()*.

## 13772 RETURN VALUE

13773 Upon successful completion, *fputcw()* shall return *wc*. Otherwise, it shall return WEOF, the error  
 13774 CX indicator for the stream shall be set, and *errno* shall be set to indicate the error.

## 13775 ERRORS

13776 The *fputcw()* function shall fail if either the stream is unbuffered or data in the *stream*'s buffer  
 13777 needs to be written, and:

13778 CX [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor underlying *stream* and the  
 13779 process would be delayed in the write operation.

13780 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for  
 13781 writing.

13782 CX [EFBIG] An attempt was made to write to a file that exceeds the maximum file size or  
 13783 the process' file size limit.

13784 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the  
 13785 offset maximum associated with the corresponding stream.

13786 [EILSEQ] The wide-character code *wc* does not correspond to a valid character.

13787 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data  
 13788 was transferred.

13789 CX [EIO] A physical I/O error has occurred, or the process is a member of a  
 13790 background process group attempting to write to its controlling terminal,  
 13791 TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the  
 13792 process group of the process is orphaned. This error may also be returned  
 13793 under implementation-defined conditions.

13794 CX [ENOSPC] There was no free space remaining on the device containing the file.

13795 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by  
 13796 any process. A SIGPIPE signal shall also be sent to the thread.

13797 The *fputc()* function may fail if:

13798 CX [ENOMEM] Insufficient storage space is available.

13799 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
13800 capabilities of the device.

#### 13801 EXAMPLES

13802 None.

#### 13803 APPLICATION USAGE

13804 None.

#### 13805 RATIONALE

13806 None.

#### 13807 FUTURE DIRECTIONS

13808 None.

#### 13809 SEE ALSO

13810 *ferror()*, *fopen()*, *setbuf()*, *ulimit()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
13811 `<stdio.h>`, `<wchar.h>`

#### 13812 CHANGE HISTORY

13813 First released in Issue 4. Derived from the MSE working draft.

#### 13814 Issue 5

13815 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*  
13816 is changed from `wint_t` to `wchar_t`.

13817 The Optional Header (OH) marking is removed from `<stdio.h>`.

13818 Large File Summit extensions are added.

#### 13819 Issue 6

13820 Extensions beyond the ISO C standard are marked.

13821 The following new requirements on POSIX implementations derive from alignment with the  
13822 Single UNIX Specification:

- 13823
- The [EFBIG] and [EIO] mandatory error conditions are added.
  - The [ENOMEM] and [ENXIO] optional error conditions are added.
- 13824

13825 **NAME**

13826 fputws — put a wide-character string on a stream

13827 **SYNOPSIS**

13828 #include &lt;stdio.h&gt;

13829 #include &lt;wchar.h&gt;

13830 int fputws(const wchar\_t \*restrict ws, FILE \*restrict stream);

13831 **DESCRIPTION**

13832 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
13833 conflict between the requirements described here and the ISO C standard is unintentional. This  
13834 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13835 The *fputws()* function shall write a character string corresponding to the (null-terminated)  
13836 wide-character string pointed to by *ws* to the stream pointed to by *stream*. No character  
13837 corresponding to the terminating null wide-character code shall be written.

13838 CX The *st\_ctime* and *st\_mtime* fields of the file shall be marked for update between the successful  
13839 execution of *fputws()* and the next successful completion of a call to *flush()* or *fclose()* on the  
13840 same stream or a call to *exit()* or *abort()*.

13841 **RETURN VALUE**

13842 Upon successful completion, *fputws()* shall return a non-negative number. Otherwise, it shall  
13843 CX return -1, set an error indicator for the stream, and set *errno* to indicate the error.

13844 **ERRORS**13845 Refer to *fputwc()*.13846 **EXAMPLES**

13847 None.

13848 **APPLICATION USAGE**13849 The *fputws()* function does not append a <newline>.13850 **RATIONALE**

13851 None.

13852 **FUTURE DIRECTIONS**

13853 None.

13854 **SEE ALSO**13855 *fopen()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>, <wchar.h>13856 **CHANGE HISTORY**

13857 First released in Issue 4. Derived from the MSE working draft.

13858 **Issue 5**

13859 The Optional Header (OH) marking is removed from &lt;stdio.h&gt;.

13860 **Issue 6**

13861 Extensions beyond the ISO C standard are marked.

13862 The *fputws()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

## 13863 NAME

13864 fread — binary input

## 13865 SYNOPSIS

13866 #include &lt;stdio.h&gt;

```
13867 size_t fread(void *restrict ptr, size_t size, size_t nitems,
13868 FILE *restrict stream);
```

## 13869 DESCRIPTION

13870 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 13871 conflict between the requirements described here and the ISO C standard is unintentional. This  
 13872 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13873 The *fread()* function shall read into the array pointed to by *ptr* up to *nitems* elements whose size  
 13874 is specified by *size* in bytes, from the stream pointed to by *stream*. For each object, *size* calls shall  
 13875 be made to the *fgetc()* function and the results stored, in the order read, in an array of **unsigned**  
 13876 **char** exactly overlaying the object. The file position indicator for the stream (if defined) shall be  
 13877 advanced by the number of bytes successfully read. If an error occurs, the resulting value of the  
 13878 file position indicator for the stream is unspecified. If a partial element is read, its value is  
 13879 unspecified.

13880 CX The *fread()* function may mark the *st\_atime* field of the file associated with *stream* for update. The  
 13881 *st\_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,  
 13882 *fgetwc()*, *fgetws()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()* using *stream* that returns  
 13883 data not supplied by a prior call to *ungetc()* or *ungetwc()*.

## 13884 RETURN VALUE

13885 Upon successful completion, *fread()* shall return the number of elements successfully read which  
 13886 is less than *nitems* only if a read error or end-of-file is encountered. If *size* or *nitems* is 0, *fread()*  
 13887 shall return 0 and the contents of the array and the state of the stream remain unchanged.  
 13888 CX Otherwise, if a read error occurs, the error indicator for the stream shall be set, and *errno* shall be  
 13889 set to indicate the error.

## 13890 ERRORS

13891 Refer to *fgetc()*.

## 13892 EXAMPLES

13893 **Reading from a Stream**13894 The following example reads a single element from the *fp* stream into the array pointed to by *buf*.

```
13895 #include <stdio.h>
13896 ...
13897 size_t bytes_read;
13898 char buf[100];
13899 FILE *fp;
13900 ...
13901 bytes_read = fread(buf, sizeof(buf), 1, fp);
13902 ...
```

## 13903 APPLICATION USAGE

13904 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an  
 13905 end-of-file condition.

13906 Because of possible differences in element length and byte ordering, files written using *fwrite()*  
 13907 are application-dependent, and possibly cannot be read using *fread()* by a different application

13908 or by the same application on a different processor.

13909 **RATIONALE**

13910 None.

13911 **FUTURE DIRECTIONS**

13912 None.

13913 **SEE ALSO**

13914 *feof()*, *ferror()*, *fgetc()*, *fopen()*, *getc()*, *gets()*, *scanf()*, the Base Definitions volume of  
13915 IEEE Std 1003.1-2001, <**stdio.h**>

13916 **CHANGE HISTORY**

13917 First released in Issue 1. Derived from Issue 1 of the SVID.

13918 **Issue 6**

13919 Extensions beyond the ISO C standard are marked.

13920 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 13921
- The *fread()* prototype is updated.
- 13922
- The DESCRIPTION is updated to describe how the bytes from a call to *fgetc()* are stored.

13923 **NAME**

13924 free — free allocated memory

13925 **SYNOPSIS**

13926 #include &lt;stdlib.h&gt;

13927 void free(void \*ptr);

13928 **DESCRIPTION**

13929 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
13930 conflict between the requirements described here and the ISO C standard is unintentional. This  
13931 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

13932 The *free()* function shall cause the space pointed to by *ptr* to be deallocated; that is, made  
13933 available for further allocation. If *ptr* is a null pointer, no action shall occur. Otherwise, if the  
13934 ADV argument does not match a pointer earlier returned by the *calloc()*, *malloc()*, *posix\_memalign()*,  
13935 XSI *realloc()*, or *strdup()* function, or if the space has been deallocated by a call to *free()* or *realloc()*,  
13936 the behavior is undefined.

13937 Any use of a pointer that refers to freed space results in undefined behavior.

13938 **RETURN VALUE**13939 The *free()* function shall not return a value.13940 **ERRORS**

13941 No errors are defined.

13942 **EXAMPLES**

13943 None.

13944 **APPLICATION USAGE**

13945 There is now no requirement for the implementation to support the inclusion of &lt;malloc.h&gt;.

13946 **RATIONALE**

13947 None.

13948 **FUTURE DIRECTIONS**

13949 None.

13950 **SEE ALSO**13951 *calloc()*, *malloc()*, *realloc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>13952 **CHANGE HISTORY**

13953 First released in Issue 1. Derived from Issue 1 of the SVID.

13954 **Issue 6**13955 Reference to the *valloc()* function is removed.

## 13956 NAME

13957 freeaddrinfo, getaddrinfo — get address information

## 13958 SYNOPSIS

```

13959 #include <sys/socket.h>
13960 #include <netdb.h>

13961 void freeaddrinfo(struct addrinfo *ai);
13962 int getaddrinfo(const char *restrict nodename,
13963 const char *restrict servname,
13964 const struct addrinfo *restrict hints,
13965 struct addrinfo **restrict res);

```

## 13966 DESCRIPTION

13967 The *freeaddrinfo()* function shall free one or more **addrinfo** structures returned by *getaddrinfo()*,  
 13968 along with any additional storage associated with those structures. If the *ai\_next* field of the  
 13969 structure is not null, the entire list of structures shall be freed. The *freeaddrinfo()* function shall  
 13970 support the freeing of arbitrary sublists of an **addrinfo** list originally returned by *getaddrinfo()*.

13971 The *getaddrinfo()* function shall translate the name of a service location (for example, a host  
 13972 name) and/or a service name and shall return a set of socket addresses and associated  
 13973 information to be used in creating a socket with which to address the specified service.

13974 **Note:** In many cases it is implemented by the Domain Name System, as documented in RFC 1034,  
 13975 RFC 1035, and RFC 1886.

13976 The *freeaddrinfo()* and *getaddrinfo()* functions shall be thread-safe.

13977 The *nodename* and *servname* arguments are either null pointers or pointers to null-terminated  
 13978 strings. One or both of these two arguments shall be supplied by the application as a non-null  
 13979 pointer.

13980 The format of a valid name depends on the address family or families. If a specific family is not  
 13981 given and the name could be interpreted as valid within multiple supported families, the  
 13982 implementation shall attempt to resolve the name in all supported families and, in absence of  
 13983 errors, one or more results shall be returned.

13984 If the *nodename* argument is not null, it can be a descriptive name or can be an address string. If  
 13985 IP6 the specified address family is AF\_INET, AF\_INET6, or AF\_UNSPEC, valid descriptive names  
 13986 include host names. If the specified address family is AF\_INET or AF\_UNSPEC, address strings  
 13987 using Internet standard dot notation as specified in *inet\_addr()* are valid.

13988 IP6 If the specified address family is AF\_INET6 or AF\_UNSPEC, standard IPv6 text forms described  
 13989 in *inet\_ntop()* are valid.

13990 If *nodename* is not null, the requested service location is named by *nodename*; otherwise, the  
 13991 requested service location is local to the caller.

13992 If *servname* is null, the call shall return network-level addresses for the specified *nodename*. If  
 13993 *servname* is not null, it is a null-terminated character string identifying the requested service. This  
 13994 can be either a descriptive name or a numeric representation suitable for use with the address  
 13995 IP6 family or families. If the specified address family is AF\_INET, AF\_INET6, or AF\_UNSPEC, the  
 13996 service can be specified as a string specifying a decimal port number.

13997 If the *hints* argument is not null, it refers to a structure containing input values that may direct  
 13998 the operation by providing options and by limiting the returned information to a specific socket  
 13999 type, address family, and/or protocol. In this *hints* structure every member other than *ai\_flags*,  
 14000 *ai\_family*, *ai\_socktype*, and *ai\_protocol* shall be set to zero or a null pointer. A value of  
 14001 AF\_UNSPEC for *ai\_family* means that the caller shall accept any address family. A value of zero

14002 for *ai\_socktype* means that the caller shall accept any socket type. A value of zero for *ai\_protocol*  
 14003 means that the caller shall accept any protocol. If *hints* is a null pointer, the behavior shall be as if  
 14004 it referred to a structure containing the value zero for the *ai\_flags*, *ai\_socktype*, and *ai\_protocol*  
 14005 fields, and AF\_UNSPEC for the *ai\_family* field.

14006 The *ai\_flags* field to which the *hints* parameter points shall be set to zero or be the bitwise-  
 14007 inclusive OR of one or more of the values AI\_PASSIVE, AI\_CANONNAME,  
 14008 AI\_NUMERICHOST, AI\_NUMERICSERV, AI\_V4MAPPED, AI\_ALL, and AI\_ADDRCONFIG.

14009 If the AI\_PASSIVE flag is specified, the returned address information shall be suitable for use in  
 14010 binding a socket for accepting incoming connections for the specified service. In this case, if the  
 14011 *nodename* argument is null, then the IP address portion of the socket address structure shall be  
 14012 set to INADDR\_ANY for an IPv4 address or IN6ADDR\_ANY\_INIT for an IPv6 address. If the  
 14013 AI\_PASSIVE flag is not specified, the returned address information shall be suitable for a call to  
 14014 *connect()* (for a connection-mode protocol) or for a call to *connect()*, *sendto()*, or *sendmsg()* (for a  
 14015 connectionless protocol). In this case, if the *nodename* argument is null, then the IP address  
 14016 portion of the socket address structure shall be set to the loopback address. The AI\_PASSIVE  
 14017 flag shall be ignored if the *nodename* argument is not null.

14018 If the AI\_CANONNAME flag is specified and the *nodename* argument is not null, the function  
 14019 shall attempt to determine the canonical name corresponding to *nodename* (for example, if  
 14020 *nodename* is an alias or shorthand notation for a complete name).

14021 **Note:** Since different implementations use different conceptual models, the terms “canonical name”  
 14022 and “alias” cannot be precisely defined for the general case. However, Domain Name System  
 14023 implementations are expected to interpret them as they are used in RFC 1034.

14024 A numeric host address string is not a “name”, and thus does not have a “canonical name”  
 14025 form; no address to host name translation is performed. See below for handling of the case  
 14026 where a canonical name cannot be obtained.

14027 If the AI\_NUMERICHOST flag is specified, then a non-null *nodename* string supplied shall be a  
 14028 numeric host address string. Otherwise, an [EAI\_NONAME] error is returned. This flag shall  
 14029 prevent any type of name resolution service (for example, the DNS) from being invoked.

14030 If the AI\_NUMERICSERV flag is specified, then a non-null *servname* string supplied shall be a  
 14031 numeric port string. Otherwise, an [EAI\_NONAME] error shall be returned. This flag shall  
 14032 prevent any type of name resolution service (for example, NIS+) from being invoked.

14033 IP6 If the AI\_V4MAPPED flag is specified along with an *ai\_family* of AF\_INET6, then *getaddrinfo()*  
 14034 shall return IPv4-mapped IPv6 addresses on finding no matching IPv6 addresses (*ai\_addrlen*  
 14035 shall be 16). The AI\_V4MAPPED flag shall be ignored unless *ai\_family* equals AF\_INET6. If the  
 14036 AI\_ALL flag is used with the AI\_V4MAPPED flag, then *getaddrinfo()* shall return all matching  
 14037 IPv6 and IPv4 addresses. The AI\_ALL flag without the AI\_V4MAPPED flag is ignored.

14038 If the AI\_ADDRCONFIG flag is specified, IPv4 addresses shall be returned only if an IPv4  
 14039 IP6 address is configured on the local system, and IPv6 addresses shall be returned only if an IPv6  
 14040 address is configured on the local system.

14041 The *ai\_socktype* field to which argument *hints* points specifies the socket type for the service, as  
 14042 defined in *socket()*. If a specific socket type is not given (for example, a value of zero) and the  
 14043 service name could be interpreted as valid with multiple supported socket types, the  
 14044 implementation shall attempt to resolve the service name for all supported socket types and, in  
 14045 the absence of errors, all possible results shall be returned. A non-zero socket type value shall  
 14046 limit the returned information to values with the specified socket type.

14047 If the *ai\_family* field to which *hints* points has the value AF\_UNSPEC, addresses shall be  
 14048 returned for use with any address family that can be used with the specified *nodename* and/or  
 14049 *servname*. Otherwise, addresses shall be returned for use only with the specified address family.

14050 If *ai\_family* is not AF\_UNSPEC and *ai\_protocol* is not zero, then addresses are returned for use  
 14051 only with the specified address family and protocol; the value of *ai\_protocol* shall be interpreted  
 14052 as in a call to the *socket()* function with the corresponding values of *ai\_family* and *ai\_protocol*.

#### 14053 RETURN VALUE

14054 A zero return value for *getaddrinfo()* indicates successful completion; a non-zero return value  
 14055 indicates failure. The possible values for the failures are listed in the ERRORS section.

14056 Upon successful return of *getaddrinfo()*, the location to which *res* points shall refer to a linked list  
 14057 of **addrinfo** structures, each of which shall specify a socket address and information for use in  
 14058 creating a socket with which to use that socket address. The list shall include at least one  
 14059 **addrinfo** structure. The *ai\_next* field of each structure contains a pointer to the next structure on  
 14060 the list, or a null pointer if it is the last structure on the list. Each structure on the list shall  
 14061 include values for use with a call to the *socket()* function, and a socket address for use with the  
 14062 *connect()* function or, if the AI\_PASSIVE flag was specified, for use with the *bind()* function. The  
 14063 fields *ai\_family*, *ai\_socktype*, and *ai\_protocol* shall be usable as the arguments to the *socket()*  
 14064 function to create a socket suitable for use with the returned address. The fields *ai\_addr* and  
 14065 *ai\_addrlen* are usable as the arguments to the *connect()* or *bind()* functions with such a socket,  
 14066 according to the AI\_PASSIVE flag.

14067 If *nodename* is not null, and if requested by the AI\_CANONNAME flag, the *ai\_canonname* field of  
 14068 the first returned **addrinfo** structure shall point to a null-terminated string containing the  
 14069 canonical name corresponding to the input *nodename*; if the canonical name is not available, then  
 14070 *ai\_canonname* shall refer to the *nodename* argument or a string with the same contents. The  
 14071 contents of the *ai\_flags* field of the returned structures are undefined.

14072 All fields in socket address structures returned by *getaddrinfo()* that are not filled in through an  
 14073 explicit argument (for example, *sin6\_flowinfo*) shall be set to zero.

14074 **Note:** This makes it easier to compare socket address structures.

#### 14075 ERRORS

14076 The *getaddrinfo()* function shall fail and return the corresponding value if:

14077 [EAI\_AGAIN] The name could not be resolved at this time. Future attempts may succeed.

14078 [EAI\_BADFLAGS]

14079 The *flags* parameter had an invalid value.

14080 [EAI\_FAIL] A non-recoverable error occurred when attempting to resolve the name.

14081 [EAI\_FAMILY] The address family was not recognized.

14082 [EAI\_MEMORY] There was a memory allocation failure when trying to allocate storage for the  
 14083 return value.

14084 [EAI\_NONAME] The name does not resolve for the supplied parameters.

14085 Neither *nodename* nor *servname* were supplied. At least one of these shall be  
 14086 supplied.

14087 [EAI\_SERVICE] The service passed was not recognized for the specified socket type.

14088 [EAI\_SOCKTYPE]

14089 The intended socket type was not recognized.

14090 [EAI\_SYSTEM] A system error occurred; the error code can be found in *errno*.

14091 [EAI\_OVERFLOW]

14092 An argument buffer overflowed.

14093 **EXAMPLES**

14094       None.

14095 **APPLICATION USAGE**

14096       If the caller handles only TCP and not UDP, for example, then the *ai\_protocol* member of the *hints*  
14097       structure should be set to IPPROTO\_TCP when *getaddrinfo()* is called.

14098       If the caller handles only IPv4 and not IPv6, then the *ai\_family* member of the *hints* structure  
14099       should be set to AF\_INET when *getaddrinfo()* is called.

14100       The term “canonical name” is misleading; it is taken from the Domain Name System (RFC 2181).  
14101       It should be noted that the canonical name is a result of alias processing, and not necessarily a  
14102       unique attribute of a host, address, or set of addresses. See RFC 2181 for more discussion of this  
14103       in the Domain Name System context.

14104 **RATIONALE**

14105       None.

14106 **FUTURE DIRECTIONS**

14107       None.

14108 **SEE ALSO**

14109       *connect()*, *gai\_strerror()*, *gethostbyaddr()*, *getnameinfo()*, *getservbyname()*, *socket()*, the Base  
14110       Definitions volume of IEEE Std 1003.1-2001, <netdb.h>, <sys/socket.h>

14111 **CHANGE HISTORY**

14112       First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

14113       The **restrict** keyword is added to the *getaddrinfo()* prototype for alignment with the  
14114       ISO/IEC 9899:1999 standard.

14115       IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/19 is applied, adding three notes to the  
14116       DESCRIPTION and adding text to the APPLICATION USAGE related to the term “canonical  
14117       name”. A reference to RFC 2181 is also added to the Informative References.

14118       IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/20 is applied, making changes for  
14119       alignment with IPv6. These include the following:

- 14120       • Adding AI\_V4MAPPED, AI\_ALL, and AI\_ADDRCONFIG to the allowed values for the  
14121        *ai\_flags* field
- 14122       • Adding a description of AI\_ADDRCONFIG
- 14123       • Adding a description of the consequences of ignoring the AI\_PASSIVE flag.

## 14124 NAME

14125        freopen — open a stream

## 14126 SYNOPSIS

14127        #include &lt;stdio.h&gt;

14128        FILE \*freopen(const char \*restrict *filename*, const char \*restrict *mode*,  
14129                      FILE \*restrict *stream*);

## 14130 DESCRIPTION

14131 CX        The functionality described on this reference page is aligned with the ISO C standard. Any  
14132        conflict between the requirements described here and the ISO C standard is unintentional. This  
14133        volume of IEEE Std 1003.1-2001 defers to the ISO C standard.14134        The *freopen()* function shall first attempt to flush the stream and close any file descriptor  
14135        associated with *stream*. Failure to flush or close the file descriptor successfully shall be ignored.  
14136        The error and end-of-file indicators for the stream shall be cleared.14137        The *freopen()* function shall open the file whose pathname is the string pointed to by *filename* and  
14138        associate the stream pointed to by *stream* with it. The *mode* argument shall be used just as in  
14139        *fopen()*.

14140        The original stream shall be closed regardless of whether the subsequent open succeeds.

14141        If *filename* is a null pointer, the *freopen()* function shall attempt to change the mode of the stream  
14142        to that specified by *mode*, as if the name of the file currently associated with the stream had been  
14143        used. It is implementation-defined which changes of mode are permitted (if any), and under  
14144        what circumstances.14145 XSI       After a successful call to the *freopen()* function, the orientation of the stream shall be cleared, the  
14146        encoding rule shall be cleared, and the associated **mbstate\_t** object shall be set to describe an  
14147        initial conversion state.14148 CX        The largest value that can be represented correctly in an object of type **off\_t** shall be established  
14149        as the offset maximum in the open file description.

## 14150 RETURN VALUE

14151        Upon successful completion, *freopen()* shall return the value of *stream*. Otherwise, a null pointer  
14152 CX        shall be returned, and *errno* shall be set to indicate the error.

## 14153 ERRORS

14154        The *freopen()* function shall fail if:14155 CX        [EACCES]        Search permission is denied on a component of the path prefix, or the file  
14156        exists and the permissions specified by *mode* are denied, or the file does not  
14157        exist and write permission is denied for the parent directory of the file to be  
14158        created.14159 CX        [EINTR]        A signal was caught during *freopen()*.14160 CX        [EISDIR]        The named file is a directory and *mode* requires write access.14161 CX        [ELOOP]        A loop exists in symbolic links encountered during resolution of the *path*  
14162        argument.

14163 CX        [EMFILE]        {OPEN\_MAX} file descriptors are currently open in the calling process.

14164 CX        [ENAMETOOLONG]        The length of the *filename* argument exceeds {PATH\_MAX} or a pathname  
14165        component is longer than {NAME\_MAX}.  
14166

|                            |                |                                                                                                                                     |
|----------------------------|----------------|-------------------------------------------------------------------------------------------------------------------------------------|
| 14167 CX                   | [ENFILE]       | The maximum allowable number of files is currently open in the system.                                                              |
| 14168 CX<br>14169          | [ENOENT]       | A component of <i>filename</i> does not name an existing file or <i>filename</i> is an empty string.                                |
| 14170 CX<br>14171          | [ENOSPC]       | The directory or file system that would contain the new file cannot be expanded, the file does not exist, and it was to be created. |
| 14172 CX                   | [ENOTDIR]      | A component of the path prefix is not a directory.                                                                                  |
| 14173 CX<br>14174          | [ENXIO]        | The named file is a character special or block special file, and the device associated with this special file does not exist.       |
| 14175 CX<br>14176          | [EOVERFLOW]    | The named file is a regular file and the size of the file cannot be represented correctly in an object of type <code>off_t</code> . |
| 14177 CX<br>14178          | [EROFS]        | The named file resides on a read-only file system and <i>mode</i> requires write access.                                            |
| 14179                      |                | The <i>freopen()</i> function may fail if:                                                                                          |
| 14180 CX                   | [EINVAL]       | The value of the <i>mode</i> argument is not valid.                                                                                 |
| 14181 CX<br>14182          | [ELOOP]        | More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.                              |
| 14183 CX<br>14184<br>14185 | [ENAMETOOLONG] | Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.                             |
| 14186 CX                   | [ENOMEM]       | Insufficient storage space is available.                                                                                            |
| 14187 CX<br>14188          | [ENXIO]        | A request was made of a nonexistent device, or the request was outside the capabilities of the device.                              |
| 14189 CX<br>14190          | [ETXTBSY]      | The file is a pure procedure (shared text) file that is being executed and <i>mode</i> requires write access.                       |

#### 14191 EXAMPLES

##### 14192 Directing Standard Output to a File

14193 The following example logs all standard output to the `/tmp/logfile` file.

```

14194 #include <stdio.h>
14195 ...
14196 FILE *fp;
14197 ...
14198 fp = freopen ("/tmp/logfile", "a+", stdout);
14199 ...

```

#### 14200 APPLICATION USAGE

14201 The *freopen()* function is typically used to attach the preopened *streams* associated with *stdin*,  
 14202 *stdout*, and *stderr* to other files.

#### 14203 RATIONALE

14204 None.

14205 **FUTURE DIRECTIONS**

14206 None.

14207 **SEE ALSO**14208 *fclose()*, *fopen()*, *fdopen()*, *mbsinit()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
14209 **<stdio.h>**14210 **CHANGE HISTORY**

14211 First released in Issue 1. Derived from Issue 1 of the SVID.

14212 **Issue 5**14213 The DESCRIPTION is updated to indicate that the orientation of the stream is cleared and the  
14214 conversion state of the stream is set to an initial conversion state by a successful call to the  
14215 *freopen()* function.

14216 Large File Summit extensions are added.

14217 **Issue 6**

14218 Extensions beyond the ISO C standard are marked.

14219 The following new requirements on POSIX implementations derive from alignment with the  
14220 Single UNIX Specification:

- 14221 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file  
14222 description. This change is to support large files.
- 14223 • In the ERRORS section, the [E\_OVERFLOW] condition is added. This change is to support  
14224 large files.
- 14225 • The [ELOOP] mandatory error condition is added.
- 14226 • A second [ENAMETOOLONG] is added as an optional error condition.
- 14227 • The [EINVAL], [ENOMEM], [ENXIO], and [ETXTBSY] optional error conditions are added.

14228 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 14229 • The *freopen()* prototype is updated.
- 14230 • The DESCRIPTION is updated.

14231 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
14232 [ELOOP] error condition is added.

14233 The DESCRIPTION is updated regarding failure to close, changing the “file” to “file descriptor”.

14234 **NAME**

14235 frexp, frexpf, frexpl — extract mantissa and exponent from a double precision number

14236 **SYNOPSIS**

14237 #include <math.h>

14238 double frexp(double num, int \*exp);

14239 float frexpf(float num, int \*exp);

14240 long double frexpl(long double num, int \*exp);

14241 **DESCRIPTION**

14242 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
14243 conflict between the requirements described here and the ISO C standard is unintentional. This  
14244 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

14245 These functions shall break a floating-point number *num* into a normalized fraction and an  
14246 integral power of 2. The integer exponent shall be stored in the **int** object pointed to by *exp*.

14247 **RETURN VALUE**

14248 For finite arguments, these functions shall return the value *x*, such that *x* has a magnitude in the  
14249 interval  $[\frac{1}{2}, 1)$  or 0, and *num* equals *x* times 2 raised to the power *\*exp*.

14250 **MX** If *num* is NaN, a NaN shall be returned, and the value of *\*exp* is unspecified.

14251 If *num* is  $\pm 0$ ,  $\pm 0$  shall be returned, and the value of *\*exp* shall be 0.

14252 If *num* is  $\pm \text{Inf}$ , *num* shall be returned, and the value of *\*exp* is unspecified.

14253 **ERRORS**

14254 No errors are defined.

14255 **EXAMPLES**

14256 None.

14257 **APPLICATION USAGE**

14258 None.

14259 **RATIONALE**

14260 None.

14261 **FUTURE DIRECTIONS**

14262 None.

14263 **SEE ALSO**

14264 *isnan()*, *ldexp()*, *modf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <math.h>

14265 **CHANGE HISTORY**

14266 First released in Issue 1. Derived from Issue 1 of the SVID.

14267 **Issue 5**

14268 The DESCRIPTION is updated to indicate how an application should check for an error. This  
14269 text was previously published in the APPLICATION USAGE section.

14270 **Issue 6**

14271 The *frexpf()* and *frexpl()* functions are added for alignment with the ISO/IEC 9899:1999  
14272 standard.

14273 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
14274 revised to align with the ISO/IEC 9899:1999 standard.

14275 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
14276 marked.

## 14277 NAME

14278 fscanf, scanf, sscanf — convert formatted input

## 14279 SYNOPSIS

14280 #include &lt;stdio.h&gt;

14281 int fscanf(FILE \*restrict *stream*, const char \*restrict *format*, ... );14282 int scanf(const char \*restrict *format*, ... );14283 int sscanf(const char \*restrict *s*, const char \*restrict *format*, ... );

## 14284 DESCRIPTION

14285 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 14286 conflict between the requirements described here and the ISO C standard is unintentional. This  
 14287 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

14288 The *fscanf()* function shall read from the named input *stream*. The *scanf()* function shall read  
 14289 from the standard input stream *stdin*. The *sscanf()* function shall read from the string *s*. Each  
 14290 function reads bytes, interprets them according to a format, and stores the results in its  
 14291 arguments. Each expects, as arguments, a control string *format* described below, and a set of  
 14292 *pointer* arguments indicating where the converted input should be stored. The result is  
 14293 undefined if there are insufficient arguments for the format. If the format is exhausted while  
 14294 arguments remain, the excess arguments shall be evaluated but otherwise ignored.

14295 XSI Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than  
 14296 to the next unused argument. In this case, the conversion specifier character % (see below) is  
 14297 replaced by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL\_ARGMAX}].  
 14298 This feature provides for the definition of format strings that select arguments in an order  
 14299 appropriate to specific languages. In format strings containing the "%n\$" form of conversion  
 14300 specifications, it is unspecified whether numbered arguments in the argument list can be  
 14301 referenced from the format string more than once.

14302 The *format* can contain either form of a conversion specification—that is, % or "%n\$"—but the  
 14303 two forms cannot be mixed within a single *format* string. The only exception to this is that %% or  
 14304 %\* can be mixed with the "%n\$" form. When numbered argument specifications are used,  
 14305 specifying the *N*th argument requires that all the leading arguments, from the first to the  
 14306 (*N*–1)th, are pointers.

14307 CX The *fscanf()* function in all its forms shall allow detection of a language-dependent radix  
 14308 character in the input string. The radix character is defined in the program's locale (category  
 14309 *LC\_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the  
 14310 radix character shall default to a period ( '.' ).

14311 The format is a character string, beginning and ending in its initial shift state, if any, composed  
 14312 of zero or more directives. Each directive is composed of one of the following: one or more  
 14313 white-space characters (<space>*s*, <tab>*s*, <newline>*s*, <vertical-tab>*s*, or <form-feed>*s*); an  
 14314 ordinary character (neither '%' nor a white-space character); or a conversion specification. Each  
 14315 XSI conversion specification is introduced by the character '%' or the character sequence "%n\$",  
 14316 after which the following appear in sequence:

- 14317 • An optional assignment-suppressing character '\*'.
- 14318 • An optional non-zero decimal integer that specifies the maximum field width.
- 14319 • An option length modifier that specifies the size of the receiving object.
- 14320 • A *conversion specifier* character that specifies the type of conversion to be applied. The valid  
 14321 conversion specifiers are described below.

14322 The *fscanf()* functions shall execute each directive of the format in turn. If a directive fails, as  
 14323 detailed below, the function shall return. Failures are described as input failures (due to the  
 14324 unavailability of input bytes) or matching failures (due to inappropriate input).

14325 A directive composed of one or more white-space characters shall be executed by reading input  
 14326 until no more valid input can be read, or up to the first byte which is not a white-space character,  
 14327 which remains unread.

14328 A directive that is an ordinary character shall be executed as follows: the next byte shall be read  
 14329 from the input and compared with the byte that comprises the directive; if the comparison  
 14330 shows that they are not equivalent, the directive shall fail, and the differing and subsequent  
 14331 bytes shall remain unread. Similarly, if end-of-file, an encoding error, or a read error prevents a  
 14332 character from being read, the directive shall fail.

14333 A directive that is a conversion specification defines a set of matching input sequences, as  
 14334 described below for each conversion character. A conversion specification shall be executed in  
 14335 the following steps.

14336 Input white-space characters (as specified by *isspace()*) shall be skipped, unless the conversion  
 14337 specification includes a `[`, `c`, `C`, or `n` conversion specifier.

14338 An item shall be read from the input, unless the conversion specification includes an `n`  
 14339 conversion specifier. An input item shall be defined as the longest sequence of input bytes (up to  
 14340 any specified maximum field width, which may be measured in characters or bytes dependent  
 14341 on the conversion specifier) which is an initial subsequence of a matching sequence. The first  
 14342 byte, if any, after the input item shall remain unread. If the length of the input item is 0, the  
 14343 execution of the conversion specification shall fail; this condition is a matching failure, unless  
 14344 end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is  
 14345 an input failure.

14346 Except in the case of a `%` conversion specifier, the input item (or, in the case of a `%n` conversion  
 14347 specification, the count of input bytes) shall be converted to a type appropriate to the conversion  
 14348 character. If the input item is not a matching sequence, the execution of the conversion  
 14349 specification fails; this condition is a matching failure. Unless assignment suppression was  
 14350 indicated by a `'*'`, the result of the conversion shall be placed in the object pointed to by the  
 14351 first argument following the *format* argument that has not already received a conversion result if  
 14352 XSI the conversion specification is introduced by `%`, or in the *n*th argument if introduced by the  
 14353 character sequence `"%n$"`. If this object does not have an appropriate type, or if the result of the  
 14354 conversion cannot be represented in the space provided, the behavior is undefined.

14355 The length modifiers and their meanings are:

14356 hh Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an  
 14357 argument with type pointer to **signed char** or **unsigned char**.

14358 h Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an  
 14359 argument with type pointer to **short** or **unsigned short**.

14360 l (ell) Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an  
 14361 argument with type pointer to **long** or **unsigned long**; that a following `a`, `A`, `e`, `E`, `f`, `F`, `g`,  
 14362 or `G` conversion specifier applies to an argument with type pointer to **double**; or that a  
 14363 following `c`, `s`, or `[` conversion specifier applies to an argument with type pointer to  
 14364 **wchar\_t**.

14365 ll (ell-ell)

14366 Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an  
 14367 argument with type pointer to **long long** or **unsigned long long**.

|       |            |                                                                                                                                                                                                                                                                                                                                                         |
|-------|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 14368 | j          | Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to <b>intmax_t</b> or <b>uintmax_t</b> .                                                                                                                                                                                                |
| 14369 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14370 | z          | Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to <b>size_t</b> or the corresponding signed integer type.                                                                                                                                                                              |
| 14371 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14372 | t          | Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to <b>ptrdiff_t</b> or the corresponding <b>unsigned</b> type.                                                                                                                                                                          |
| 14373 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14374 | L          | Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to an argument with type pointer to <b>long double</b> .                                                                                                                                                                                                              |
| 14375 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14376 |            | If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.                                                                                                                                                                                                                                    |
| 14377 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14378 |            | The following conversion specifiers are valid:                                                                                                                                                                                                                                                                                                          |
| 14379 | d          | Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <i>strtol()</i> with the value 10 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>int</b> .                                             |
| 14380 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14381 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14382 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14383 | i          | Matches an optionally signed integer, whose format is the same as expected for the subject sequence of <i>strtol()</i> with 0 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>int</b> .                                                                |
| 14384 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14385 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14386 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14387 | o          | Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of <i>strtol()</i> with the value 8 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>unsigned</b> .                                           |
| 14388 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14389 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14390 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14391 | u          | Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <i>strtol()</i> with the value 10 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>unsigned</b> .                                        |
| 14392 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14393 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14394 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14395 | x          | Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of <i>strtol()</i> with the value 16 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>unsigned</b> .                                    |
| 14396 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14397 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14398 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14399 | a, e, f, g |                                                                                                                                                                                                                                                                                                                                                         |
| 14400 |            | Matches an optionally signed floating-point number, infinity, or NaN, whose format is the same as expected for the subject sequence of <i>strtod()</i> . In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>float</b> .                                                                 |
| 14401 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14402 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14403 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14404 |            | If the <i>fprintf()</i> family of functions generates character string representations for infinity and NaN (a symbolic entity encoded in floating-point format) to support IEEE Std 754-1985, the <i>fscanf()</i> family of functions shall recognize them as input.                                                                                   |
| 14405 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14406 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14407 | s          | Matches a sequence of bytes that are not white-space characters. The application shall ensure that the corresponding argument is a pointer to the initial byte of an array of <b>char</b> , <b>signed char</b> , or <b>unsigned char</b> large enough to accept the sequence and a terminating null character code, which shall be added automatically. |
| 14408 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14409 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14410 |            |                                                                                                                                                                                                                                                                                                                                                         |
| 14411 |            | If an l (ell) qualifier is present, the input is a sequence of characters that begins in the initial shift state. Each character shall be converted to a wide character as if by a call to                                                                                                                                                              |
| 14412 |            |                                                                                                                                                                                                                                                                                                                                                         |

|       |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 14413 |   | the <i>mbrtowc()</i> function, with the conversion state described by an <b>mbstate_t</b> object initialized to zero before the first character is converted. The application shall ensure that the corresponding argument is a pointer to an array of <b>wchar_t</b> large enough to accept the sequence and the terminating null wide character, which shall be added automatically.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 14414 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14415 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14416 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14417 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14418 | [ | Matches a non-empty sequence of bytes from a set of expected bytes (the <i>scanset</i> ). The normal skip over white-space characters shall be suppressed in this case. The application shall ensure that the corresponding argument is a pointer to the initial byte of an array of <b>char</b> , <b>signed char</b> , or <b>unsigned char</b> large enough to accept the sequence and a terminating null byte, which shall be added automatically.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 14419 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14420 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14421 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14422 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14423 |   | If an <b>l</b> (ell) qualifier is present, the input is a sequence of characters that begins in the initial shift state. Each character in the sequence shall be converted to a wide character as if by a call to the <i>mbrtowc()</i> function, with the conversion state described by an <b>mbstate_t</b> object initialized to zero before the first character is converted. The application shall ensure that the corresponding argument is a pointer to an array of <b>wchar_t</b> large enough to accept the sequence and the terminating null wide character, which shall be added automatically.                                                                                                                                                                                                                                                                                                                                                   |
| 14424 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14425 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14426 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14427 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14428 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14429 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14430 |   | The conversion specification includes all subsequent bytes in the <i>format</i> string up to and including the matching right square bracket (']'). The bytes between the square brackets (the <i>scanlist</i> ) comprise the scanset, unless the byte after the left square bracket is a circumflex (^), in which case the scanset contains all bytes that do not appear in the scanlist between the circumflex and the right square bracket. If the conversion specification begins with "[ ]" or "[^]", the right square bracket is included in the scanlist and the next right square bracket is the matching right square bracket that ends the conversion specification; otherwise, the first right square bracket is the one that ends the conversion specification. If a '-' is in the scanlist and is not the first character, nor the second where the first character is a '^', nor the last character, the behavior is implementation-defined. |
| 14431 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14432 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14433 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14434 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14435 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14436 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14437 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14438 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14439 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14440 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14441 | c | Matches a sequence of bytes of the number specified by the field width (1 if no field width is present in the conversion specification). The application shall ensure that the corresponding argument is a pointer to the initial byte of an array of <b>char</b> , <b>signed char</b> , or <b>unsigned char</b> large enough to accept the sequence. No null byte is added. The normal skip over white-space characters shall be suppressed in this case.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 14442 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14443 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14444 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14445 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14446 |   | If an <b>l</b> (ell) qualifier is present, the input shall be a sequence of characters that begins in the initial shift state. Each character in the sequence is converted to a wide character as if by a call to the <i>mbrtowc()</i> function, with the conversion state described by an <b>mbstate_t</b> object initialized to zero before the first character is converted. The application shall ensure that the corresponding argument is a pointer to an array of <b>wchar_t</b> large enough to accept the resulting sequence of wide characters. No null wide character is added.                                                                                                                                                                                                                                                                                                                                                                 |
| 14447 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14448 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14449 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14450 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14451 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14452 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14453 | p | Matches an implementation-defined set of sequences, which shall be the same as the set of sequences that is produced by the %p conversion specification of the corresponding <i>fprintf()</i> functions. The application shall ensure that the corresponding argument is a pointer to a pointer to <b>void</b> . The interpretation of the input item is implementation-defined. If the input item is a value converted earlier during the same program execution, the pointer that results shall compare equal to that value; otherwise, the behavior of the %p conversion specification is undefined.                                                                                                                                                                                                                                                                                                                                                    |
| 14454 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14455 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14456 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14457 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14458 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14459 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 14460 | n | No input is consumed. The application shall ensure that the corresponding argument is a pointer to the integer into which shall be written the number of bytes read from the                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 14461 |   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

14462 input so far by this call to the *fscanf()* functions. Execution of a `%n` conversion  
 14463 specification shall not increment the assignment count returned at the completion of  
 14464 execution of the function. No argument shall be converted, but one shall be consumed.  
 14465 If the conversion specification includes an assignment-suppressing character or a field  
 14466 width, the behavior is undefined.

14467 XSI **C** Equivalent to `lc`.

14468 XSI **S** Equivalent to `ls`.

14469 `%` Matches a single `'%'` character; no conversion or assignment occurs. The complete  
 14470 conversion specification shall be `%%`.

14471 If a conversion specification is invalid, the behavior is undefined.

14472 The conversion specifiers `A`, `E`, `F`, `G`, and `X` are also valid and shall be equivalent to `a`, `e`, `f`, `g`, and  
 14473 `x`, respectively.

14474 If end-of-file is encountered during input, conversion shall be terminated. If end-of-file occurs  
 14475 before any bytes matching the current conversion specification (except for `%n`) have been read  
 14476 (other than leading white-space characters, where permitted), execution of the current  
 14477 conversion specification shall terminate with an input failure. Otherwise, unless execution of the  
 14478 current conversion specification is terminated with a matching failure, execution of the  
 14479 following conversion specification (if any) shall be terminated with an input failure.

14480 Reaching the end of the string in *sscanf()* shall be equivalent to encountering end-of-file for  
 14481 *fscanf()*.

14482 If conversion terminates on a conflicting input, the offending input is left unread in the input.  
 14483 Any trailing white space (including `<newline>`s) shall be left unread unless matched by a  
 14484 conversion specification. The success of literal matches and suppressed assignments is only  
 14485 directly determinable via the `%n` conversion specification.

14486 CX The *fscanf()* and *scanf()* functions may mark the *st\_atime* field of the file associated with *stream*  
 14487 for update. The *st\_atime* field shall be marked for update by the first successful execution of  
 14488 *fgetc()*, *fgets()*, *fread()*, *getc()*, *getchar()*, *gets()*, *fscanf()*, or *scanf()* using *stream* that returns data  
 14489 not supplied by a prior call to *ungetc()*.

#### 14490 RETURN VALUE

14491 Upon successful completion, these functions shall return the number of successfully matched  
 14492 and assigned input items; this number can be zero in the event of an early matching failure. If  
 14493 the input ends before the first matching failure or conversion, EOF shall be returned. If a read  
 14494 CX error occurs, the error indicator for the stream is set, EOF shall be returned, and *errno* shall be set  
 14495 to indicate the error.

#### 14496 ERRORS

14497 For the conditions under which the *fscanf()* functions fail and may fail, refer to *fgetc()* or  
 14498 *fgetwc()*.

14499 In addition, *fscanf()* may fail if:

14500 XSI **[EILSEQ]** Input byte sequence does not form a valid character.

14501 XSI **[EINVAL]** There are insufficient arguments.

14502 **EXAMPLES**

14503 The call:

```
14504 int i, n; float x; char name[50];
14505 n = scanf("%d%f%s", &i, &x, name);
```

14506 with the input line:

14507 25 54.32E-1 Hamster

14508 assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string  
14509 "Hamster".

14510 The call:

```
14511 int i; float x; char name[50];
14512 (void) scanf("%2d%f*d %[0123456789]", &i, &x, name);
```

14513 with input:

14514 56789 0123 56a72

14515 assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string "56\0" in *name*. The next call to  
14516 *getchar()* shall return the character 'a'.

14517 **Reading Data into an Array**

14518 The following call uses *fscanf()* to read three floating-point numbers from standard input into  
14519 the *input* array.

```
14520 float input[3]; fscanf (stdin, "%f %f %f", input, input+1, input+2);
```

14521 **APPLICATION USAGE**

14522 If the application calling *fscanf()* has any objects of type **wint\_t** or **wchar\_t**, it must also include  
14523 the **<wchar.h>** header to have these objects defined.

14524 **RATIONALE**

14525 This function is aligned with the ISO/IEC 9899:1999 standard, and in doing so a few "obvious"  
14526 things were not included. Specifically, the set of characters allowed in a scanset is limited to  
14527 single-byte characters. In other similar places, multi-byte characters have been permitted, but  
14528 for alignment with the ISO/IEC 9899:1999 standard, it has not been done here. Applications  
14529 needing this could use the corresponding wide-character functions to achieve the desired  
14530 results.

14531 **FUTURE DIRECTIONS**

14532 None.

14533 **SEE ALSO**

14534 *getc()*, *printf()*, *setlocale()*, *strtod()*, *strtol()*, *strtoul()*, *wcrtomb()*, the Base Definitions volume of  
14535 IEEE Std 1003.1-2001, Chapter 7, Locale, **<langinfo.h>**, **<stdio.h>**, **<wchar.h>**

14536 **CHANGE HISTORY**

14537 First released in Issue 1. Derived from Issue 1 of the SVID.

14538 **Issue 5**

14539 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the l (ell) qualifier is  
14540 now defined for the c, s, and [ conversion specifiers.

14541 The DESCRIPTION is updated to indicate that if infinity and NaN can be generated by the  
14542 *fprintf()* family of functions, then they are recognized by the *fscanf()* family.

14543 **Issue 6**

14544 The Open Group Corrigenda U021/7 and U028/10 are applied. These correct several  
14545 occurrences of “characters” in the text which have been replaced with the term “bytes”.

14546 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

14547 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

14548 • The prototypes for *fscanf()*, *scanf()*, and *sscanf()* are updated.

14549 • The DESCRIPTION is updated.

14550 • The *hh*, *ll*, *j*, *t*, and *z* length modifiers are added.

14551 • The *a*, *A*, and *F* conversion characters are added.

14552 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion  
14553 specification” consistently.

14554 **NAME**

14555        fseek, fseeko — reposition a file-position indicator in a stream

14556 **SYNOPSIS**

14557        #include <stdio.h>

14558        int fseek(FILE \*stream, long offset, int whence);

14559 CX     int fseeko(FILE \*stream, off\_t offset, int whence);

14560

14561 **DESCRIPTION**

14562 CX     The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

14565        The *fseek()* function shall set the file-position indicator for the stream pointed to by *stream*. If a read or write error occurs, the error indicator for the stream shall be set and *fseek()* fails.

14567        The new position, measured in bytes from the beginning of the file, shall be obtained by adding *offset* to the position specified by *whence*. The specified point is the beginning of the file for SEEK\_SET, the current value of the file-position indicator for SEEK\_CUR, or end-of-file for SEEK\_END.

14571        If the stream is to be used with wide-character input/output functions, the application shall ensure that *offset* is either 0 or a value returned by an earlier call to *ftell()* on the same stream and *whence* is SEEK\_SET.

14574        A successful call to *fseek()* shall clear the end-of-file indicator for the stream and undo any effects of *ungetc()* and *ungetwc()* on the same stream. After an *fseek()* call, the next operation on an update stream may be either input or output.

14577 CX     If the most recent operation, other than *ftell()*, on a given stream is *fflush()*, the file offset in the underlying open file description shall be adjusted to reflect the location specified by *fseek()*.

14579        The *fseek()* function shall allow the file-position indicator to be set beyond the end of existing data in the file. If data is later written at this point, subsequent reads of data in the gap shall return bytes with the value 0 until data is actually written into the gap.

14582        The behavior of *fseek()* on devices which are incapable of seeking is implementation-defined. The value of the file offset associated with such a device is undefined.

14584        If the stream is writable and buffered data had not been written to the underlying file, *fseek()* shall cause the unwritten data to be written to the file and shall mark the *st\_ctime* and *st\_mtime* fields of the file for update.

14587        In a locale with state-dependent encoding, whether *fseek()* restores the stream's shift state is implementation-defined.

14589        The *fseeko()* function shall be equivalent to the *fseek()* function except that the *offset* argument is of type **off\_t**.

14591 **RETURN VALUE**

14592 CX     The *fseek()* and *fseeko()* functions shall return 0 if they succeed.

14593 CX     Otherwise, they shall return -1 and set *errno* to indicate the error.

14594 **ERRORS**

14595 CX     The *fseek()* and *fseeko()* functions shall fail if, either the *stream* is unbuffered or the *stream*'s buffer needed to be flushed, and the call to *fseek()* or *fseeko()* causes an underlying *lseek()* or *write()* to be invoked, and:

|                                              |             |                                                                                                                                                                                                                                                                                                                                                                 |
|----------------------------------------------|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 14598 CX<br>14599                            | [EAGAIN]    | The O_NONBLOCK flag is set for the file descriptor and the process would be delayed in the write operation.                                                                                                                                                                                                                                                     |
| 14600 CX<br>14601                            | [EBADF]     | The file descriptor underlying the stream file is not open for writing or the stream's buffer needed to be flushed and the file is not open.                                                                                                                                                                                                                    |
| 14602 CX                                     | [EFBIG]     | An attempt was made to write a file that exceeds the maximum file size.                                                                                                                                                                                                                                                                                         |
| 14603 XSI                                    | [EFBIG]     | An attempt was made to write a file that exceeds the process' file size limit.                                                                                                                                                                                                                                                                                  |
| 14604 CX<br>14605                            | [EFBIG]     | The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.                                                                                                                                                                                                                           |
| 14606 CX<br>14607                            | [EINTR]     | The write operation was terminated due to the receipt of a signal, and no data was transferred.                                                                                                                                                                                                                                                                 |
| 14608 CX<br>14609                            | [EINVAL]    | The <i>whence</i> argument is invalid. The resulting file-position indicator would be set to a negative value.                                                                                                                                                                                                                                                  |
| 14610 CX<br>14611<br>14612<br>14613<br>14614 | [EIO]       | A physical I/O error has occurred, or the process is a member of a background process group attempting to perform a <i>write()</i> to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions. |
| 14615 CX                                     | [ENOSPC]    | There was no free space remaining on the device containing the file.                                                                                                                                                                                                                                                                                            |
| 14616 CX<br>14617                            | [ENXIO]     | A request was made of a nonexistent device, or the request was outside the capabilities of the device.                                                                                                                                                                                                                                                          |
| 14618 CX<br>14619                            | [EOVERFLOW] | For <i>fseek()</i> , the resulting file offset would be a value which cannot be represented correctly in an object of type <b>long</b> .                                                                                                                                                                                                                        |
| 14620 CX<br>14621                            | [EOVERFLOW] | For <i>fseeko()</i> , the resulting file offset would be a value which cannot be represented correctly in an object of type <b>off_t</b> .                                                                                                                                                                                                                      |
| 14622 CX<br>14623                            | [EPIPE]     | An attempt was made to write to a pipe or FIFO that is not open for reading by any process; a SIGPIPE signal shall also be sent to the thread.                                                                                                                                                                                                                  |
| 14624 CX                                     | [ESPIPE]    | The file descriptor underlying <i>stream</i> is associated with a pipe or FIFO.                                                                                                                                                                                                                                                                                 |

**14625 EXAMPLES**

14626 None.

**14627 APPLICATION USAGE**

14628 None.

**14629 RATIONALE**

14630 None.

**14631 FUTURE DIRECTIONS**

14632 None.

**14633 SEE ALSO**

14634 *fopen()*, *fsetpos()*, *ftell()*, *getrlimit()*, *lseek()*, *rewind()*, *ulimit()*, *ungetc()*, *write()*, the Base  
 14635 Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

**14636 CHANGE HISTORY**

14637 First released in Issue 1. Derived from Issue 1 of the SVID.

14638 **Issue 5**

14639 Normative text previously in the APPLICATION USAGE section is moved to the  
14640 DESCRIPTION.

14641 Large File Summit extensions are added.

14642 **Issue 6**

14643 Extensions beyond the ISO C standard are marked.

14644 The following new requirements on POSIX implementations derive from alignment with the  
14645 Single UNIX Specification:

- 14646 • The *fseeko()* function is added.
- 14647 • The [EFBIG], [EOVERFLOW], and [ENXIO] mandatory error conditions are added.

14648 The following change is incorporated for alignment with the FIPS requirements:

- 14649 • The [EINTR] error is no longer an indication that the implementation does not report partial  
14650 transfers.

14651 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

14652 The DESCRIPTION is updated to explicitly state that *fseek()* sets the file-position indicator, and  
14653 then on error the error indicator is set and *fseek()* fails. This is for alignment with the  
14654 ISO/IEC 9899:1999 standard.

14655 **NAME**

14656           fsetpos — set current file position

14657 **SYNOPSIS**

14658           #include &lt;stdio.h&gt;

14659           int fsetpos(FILE \*stream, const fpos\_t \*pos);

14660 **DESCRIPTION**

14661 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
 14662       conflict between the requirements described here and the ISO C standard is unintentional. This  
 14663       volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

14664       The *fsetpos()* function shall set the file position and state indicators for the stream pointed to by  
 14665       *stream* according to the value of the object pointed to by *pos*, which the application shall ensure  
 14666       is a value obtained from an earlier call to *fgetpos()* on the same stream. If a read or write error  
 14667       occurs, the error indicator for the stream shall be set and *fsetpos()* fails.

14668       A successful call to the *fsetpos()* function shall clear the end-of-file indicator for the stream and  
 14669       undo any effects of *ungetc()* on the same stream. After an *fsetpos()* call, the next operation on an  
 14670       update stream may be either input or output.

14671 CX       The behavior of *fsetpos()* on devices which are incapable of seeking is implementation-defined.  
 14672       The value of the file offset associated with such a device is undefined.

14673 **RETURN VALUE**

14674       The *fsetpos()* function shall return 0 if it succeeds; otherwise, it shall return a non-zero value and  
 14675       set *errno* to indicate the error.

14676 **ERRORS**

14677 CX       The *fsetpos()* function shall fail if, either the *stream* is unbuffered or the *stream*'s buffer needed to  
 14678       be flushed, and the call to *fsetpos()* causes an underlying *lseek()* or *write()* to be invoked, and:

14679 CX       [EAGAIN]       The O\_NONBLOCK flag is set for the file descriptor and the process would be  
 14680       delayed in the write operation.

14681 CX       [EBADF]       The file descriptor underlying the stream file is not open for writing or the  
 14682       stream's buffer needed to be flushed and the file is not open.

14683 CX       [EFBIG]       An attempt was made to write a file that exceeds the maximum file size.

14684 XSI       [EFBIG]       An attempt was made to write a file that exceeds the process' file size limit.

14685 CX       [EFBIG]       The file is a regular file and an attempt was made to write at or beyond the  
 14686       offset maximum associated with the corresponding stream.

14687 CX       [EINTR]       The write operation was terminated due to the receipt of a signal, and no data  
 14688       was transferred.

14689 CX       [EIO]        A physical I/O error has occurred, or the process is a member of a  
 14690       background process group attempting to perform a *write()* to its controlling  
 14691       terminal, TOSTOP is set, the process is neither ignoring nor blocking  
 14692       SIGTTOU, and the process group of the process is orphaned. This error may  
 14693       also be returned under implementation-defined conditions.

14694 CX       [ENOSPC]       There was no free space remaining on the device containing the file.

14695 CX       [ENXIO]       A request was made of a nonexistent device, or the request was outside the  
 14696       capabilities of the device.

- 14697 CX [EPIPE] The file descriptor underlying *stream* is associated with a pipe or FIFO.
- 14698 CX [EPIPE] An attempt was made to write to a pipe or FIFO that is not open for reading  
14699 by any process; a SIGPIPE signal shall also be sent to the thread.
- 14700 **EXAMPLES**
- 14701 None.
- 14702 **APPLICATION USAGE**
- 14703 None.
- 14704 **RATIONALE**
- 14705 None.
- 14706 **FUTURE DIRECTIONS**
- 14707 None.
- 14708 **SEE ALSO**
- 14709 *fopen()*, *ftell()*, *lseek()*, *rewind()*, *ungetc()*, *write()*, the Base Definitions volume of  
14710 IEEE Std 1003.1-2001, <stdio.h>
- 14711 **CHANGE HISTORY**
- 14712 First released in Issue 4. Derived from the ISO C standard.
- 14713 **Issue 6**
- 14714 Extensions beyond the ISO C standard are marked.
- 14715 An additional [ESPIPE] error condition is added for sockets.
- 14716 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.
- 14717 The DESCRIPTION is updated to clarify that the error indicator is set for the stream on a read or  
14718 write error. This is for alignment with the ISO/IEC 9899:1999 standard.
- 14719 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/21 is applied, deleting an erroneous  
14720 [EINVAL] error case from the ERRORS section. |

14721 **NAME**

14722        fstat — get file status

14723 **SYNOPSIS**

14724        #include &lt;sys/stat.h&gt;

14725        int fstat(int *fildev*, struct stat \**buf*);14726 **DESCRIPTION**14727        The *fstat()* function shall obtain information about an open file associated with the file descriptor *fildev*, and shall write it to the area pointed to by *buf*.14729 SHM        If *fildev* references a shared memory object, the implementation shall update in the **stat** structure pointed to by the *buf* argument only the *st\_uid*, *st\_gid*, *st\_size*, and *st\_mode* fields, and only the S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits need be valid. The implementation may update other fields and flags.14733 TYM        If *fildev* references a typed memory object, the implementation shall update in the **stat** structure pointed to by the *buf* argument only the *st\_uid*, *st\_gid*, *st\_size*, and *st\_mode* fields, and only the S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits need be valid. The implementation may update other fields and flags.14737        The *buf* argument is a pointer to a **stat** structure, as defined in <sys/stat.h>, into which information is placed concerning the file.14739        The structure members *st\_mode*, *st\_ino*, *st\_dev*, *st\_uid*, *st\_gid*, *st\_atime*, *st\_ctime*, and *st\_mtime* shall have meaningful values for all other file types defined in this volume of IEEE Std 1003.1-2001. The value of the member *st\_nlink* shall be set to the number of links to the file.14743        An implementation that provides additional or alternative file access control mechanisms may, under implementation-defined conditions, cause *fstat()* to fail.14745        The *fstat()* function shall update any time-related fields as described in the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.7, File Times Update, before writing into the **stat** structure.14748 **RETURN VALUE**14749        Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.14751 **ERRORS**14752        The *fstat()* function shall fail if:14753        [EBADF]        The *fildev* argument is not a valid file descriptor.

14754        [EIO]         An I/O error occurred while reading from the file system.

14755        [EOVERFLOW]   The file size in bytes or the number of blocks allocated to the file or the file serial number cannot be represented correctly in the structure pointed to by *buf*.14758        The *fstat()* function may fail if:14759        [EOVERFLOW]   One of the values is too large to store into the structure pointed to by the *buf* argument.

14761 **EXAMPLES**14762 **Obtaining File Status Information**

14763 The following example shows how to obtain file status information for a file named  
 14764 `/home/cnd/mod1`. The structure variable `buffer` is defined for the `stat` structure. The  
 14765 `/home/cnd/mod1` file is opened with read/write privileges and is passed to the open file  
 14766 descriptor `fildes`.

```
14767 #include <sys/types.h>
14768 #include <sys/stat.h>
14769 #include <fcntl.h>

14770 struct stat buffer;
14771 int status;
14772 ...
14773 fildes = open("/home/cnd/mod1", O_RDWR);
14774 status = fstat(fildes, &buffer);
```

14775 **APPLICATION USAGE**

14776 None.

14777 **RATIONALE**

14778 None.

14779 **FUTURE DIRECTIONS**

14780 None.

14781 **SEE ALSO**

14782 `lstat()`, `stat()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<sys/stat.h>`, `<sys/types.h>`

14783 **CHANGE HISTORY**

14784 First released in Issue 1. Derived from Issue 1 of the SVID.

14785 **Issue 5**

14786 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

14787 Large File Summit extensions are added.

14788 **Issue 6**

14789 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

14790 The following new requirements on POSIX implementations derive from alignment with the  
 14791 Single UNIX Specification:

- 14792 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
 14793 required for conforming implementations of previous POSIX specifications, it was not  
 14794 required for UNIX applications.
- 14795 • The [EIO] mandatory error condition is added.
- 14796 • The [EOVERFLOW] mandatory error condition is added. This change is to support large  
 14797 files.
- 14798 • The [EOVERFLOW] optional error condition is added.

14799 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
 14800 shared memory object semantics apply to typed memory objects.

14801 **NAME**

14802 fstatvfs, statvfs — get file system information

14803 **SYNOPSIS**

14804 XSI #include &lt;sys/statvfs.h&gt;

14805 int fstatvfs(int *fildev*, struct statvfs \**buf*);14806 int statvfs(const char \*restrict *path*, struct statvfs \*restrict *buf*);

14807

14808 **DESCRIPTION**14809 The *fstatvfs()* function shall obtain information about the file system containing the file  
14810 referenced by *fildev*.14811 The *statvfs()* function shall obtain information about the file system containing the file named by  
14812 *path*.14813 For both functions, the *buf* argument is a pointer to a **statvfs** structure that shall be filled. Read,  
14814 write, or execute permission of the named file is not required.14815 The following flags can be returned in the *f\_flag* member:

14816 ST\_RDONLY Read-only file system.

14817 ST\_NOSUID Setuid/setgid bits ignored by *exec*.14818 It is unspecified whether all members of the **statvfs** structure have meaningful values on all file  
14819 systems.14820 **RETURN VALUE**14821 Upon successful completion, *statvfs()* shall return 0. Otherwise, it shall return  $-1$  and set *errno* to  
14822 indicate the error.14823 **ERRORS**14824 The *fstatvfs()* and *statvfs()* functions shall fail if:

14825 [EIO] An I/O error occurred while reading the file system.

14826 [EINTR] A signal was caught during execution of the function.

14827 [EOVERFLOW] One of the values to be returned cannot be represented correctly in the  
14828 structure pointed to by *buf*.14829 The *fstatvfs()* function shall fail if:14830 [EBADF] The *fildev* argument is not an open file descriptor.14831 The *statvfs()* function shall fail if:

14832 [EACCES] Search permission is denied on a component of the path prefix.

14833 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
14834 argument.

14835 [ENAMETOOLONG]

14836 The length of a pathname exceeds {PATH\_MAX} or a pathname component is  
14837 longer than {NAME\_MAX}.14838 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.14839 [ENOTDIR] A component of the path prefix of *path* is not a directory.

14840 The *statvfs()* function may fail if:

14841 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
14842 resolution of the *path* argument.

14843 [ENAMETOOLONG]

14844 Pathname resolution of a symbolic link produced an intermediate result  
14845 whose length exceeds {PATH\_MAX}.

## 14846 EXAMPLES

### 14847 Obtaining File System Information Using *fstatvfs()*

14848 The following example shows how to obtain file system information for the file system upon  
14849 which the file named */home/cnd/mod1* resides, using the *fstatvfs()* function. The  
14850 */home/cnd/mod1* file is opened with read/write privileges and the open file descriptor is passed  
14851 to the *fstatvfs()* function.

```
14852 #include <statvfs.h>
14853 #include <fcntl.h>

14854 struct statvfs buffer;
14855 int status;
14856 ...
14857 fildes = open("/home/cnd/mod1", O_RDWR);
14858 status = fstatvfs(fildes, &buffer);
```

### 14859 Obtaining File System Information Using *statvfs()*

14860 The following example shows how to obtain file system information for the file system upon  
14861 which the file named */home/cnd/mod1* resides, using the *statvfs()* function.

```
14862 #include <statvfs.h>

14863 struct statvfs buffer;
14864 int status;
14865 ...
14866 status = statvfs("/home/cnd/mod1", &buffer);
```

## 14867 APPLICATION USAGE

14868 None.

## 14869 RATIONALE

14870 None.

## 14871 FUTURE DIRECTIONS

14872 None.

## 14873 SEE ALSO

14874 *chmod()*, *chown()*, *creat()*, *dup()*, *exec*, *fcntl()*, *link()*, *mknod()*, *open()*, *pipe()*, *read()*, *time()*,  
14875 *unlink()*, *utime()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, <*sys/statvfs.h*>

## 14876 CHANGE HISTORY

14877 First released in Issue 4, Version 2.

## 14878 Issue 5

14879 Moved from X/OPEN UNIX extension to BASE.

14880 Large File Summit extensions are added.

14881 **Issue 6**

- 14882 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.
- 14883 The **restrict** keyword is added to the *statvfs()* prototype for alignment with the  
14884 ISO/IEC 9899:1999 standard.
- 14885 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
14886 [ELOOP] error condition is added.

14887 **NAME**

14888 fsync — synchronize changes to a file

14889 **SYNOPSIS**

```
14890 FSC #include <unistd.h>
```

```
14891 int fsync(int fildev);
```

14892

14893 **DESCRIPTION**

14894 The *fsync()* function shall request that all data for the open file descriptor named by *fildev* is to be  
14895 transferred to the storage device associated with the file described by *fildev* in an  
14896 implementation-defined manner. The *fsync()* function shall not return until the system has  
14897 completed that action or until an error is detected.

14898 SIO If `_POSIX_SYNCHRONIZED_IO` is defined, the *fsync()* function shall force all currently queued  
14899 I/O operations associated with the file indicated by file descriptor *fildev* to the synchronized I/O  
14900 completion state. All I/O operations shall be completed as defined for synchronized I/O file  
14901 integrity completion.

14902 **RETURN VALUE**

14903 Upon successful completion, *fsync()* shall return 0. Otherwise, `-1` shall be returned and *errno* set  
14904 to indicate the error. If the *fsync()* function fails, outstanding I/O operations are not guaranteed  
14905 to have been completed.

14906 **ERRORS**

14907 The *fsync()* function shall fail if:

14908 [EBADF] The *fildev* argument is not a valid descriptor.

14909 [EINTR] The *fsync()* function was interrupted by a signal.

14910 [EINVAL] The *fildev* argument does not refer to a file on which this operation is possible.

14911 [EIO] An I/O error occurred while reading from or writing to the file system.

14912 In the event that any of the queued I/O operations fail, *fsync()* shall return the error conditions  
14913 defined for *read()* and *write()*.

14914 **EXAMPLES**

14915 None.

14916 **APPLICATION USAGE**

14917 The *fsync()* function should be used by programs which require modifications to a file to be  
14918 completed before continuing; for example, a program which contains a simple transaction  
14919 facility might use it to ensure that all modifications to a file or files caused by a transaction are  
14920 recorded.

14921 **RATIONALE**

14922 The *fsync()* function is intended to force a physical write of data from the buffer cache, and to  
14923 assure that after a system crash or other failure that all data up to the time of the *fsync()* call is  
14924 recorded on the disk. Since the concepts of “buffer cache”, “system crash”, “physical write”, and  
14925 “non-volatile storage” are not defined here, the wording has to be more abstract.

14926 If `_POSIX_SYNCHRONIZED_IO` is not defined, the wording relies heavily on the conformance  
14927 document to tell the user what can be expected from the system. It is explicitly intended that a  
14928 null implementation is permitted. This could be valid in the case where the system cannot assure  
14929 non-volatile storage under any circumstances or when the system is highly fault-tolerant and the  
14930 functionality is not required. In the middle ground between these extremes, *fsync()* might or  
14931 might not actually cause data to be written where it is safe from a power failure. The

14932 conformance document should identify at least that one configuration exists (and how to obtain  
14933 that configuration) where this can be assured for at least some files that the user can select to use  
14934 for critical data. It is not intended that an exhaustive list is required, but rather sufficient  
14935 information is provided so that if critical data needs to be saved, the user can determine how the  
14936 system is to be configured to allow the data to be written to non-volatile storage.

14937 It is reasonable to assert that the key aspects of *fsync()* are unreasonable to test in a test suite.  
14938 That does not make the function any less valuable, just more difficult to test. A formal  
14939 conformance test should probably force a system crash (power shutdown) during the test for  
14940 this condition, but it needs to be done in such a way that automated testing does not require this  
14941 to be done except when a formal record of the results is being made. It would also not be  
14942 unreasonable to omit testing for *fsync()*, allowing it to be treated as a quality-of-implementation  
14943 issue.

#### 14944 **FUTURE DIRECTIONS**

14945 None.

#### 14946 **SEE ALSO**

14947 *sync()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

#### 14948 **CHANGE HISTORY**

14949 First released in Issue 3.

#### 14950 **Issue 5**

14951 Aligned with *fsync()* in the POSIX Realtime Extension. Specifically, the DESCRIPTION and  
14952 RETURN VALUE sections are much expanded, and the ERRORS section is updated to indicate  
14953 that *fsync()* can return the error conditions defined for *read()* and *write()*.

#### 14954 **Issue 6**

14955 This function is marked as part of the File Synchronization option.

14956 The following new requirements on POSIX implementations derive from alignment with the  
14957 Single UNIX Specification:

- 14958 • The [EINVAL] and [EIO] mandatory error conditions are added.

14959 **NAME**

14960 ftell, ftello — return a file offset in a stream

14961 **SYNOPSIS**

14962 #include <stdio.h>

14963 long ftell(FILE \*stream);

14964 CX off\_t ftello(FILE \*stream);

14965

14966 **DESCRIPTION**

14967 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
14968 conflict between the requirements described here and the ISO C standard is unintentional. This  
14969 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

14970 The *ftell()* function shall obtain the current value of the file-position indicator for the stream  
14971 pointed to by *stream*.

14972 CX The *ftello()* function shall be equivalent to *ftell()*, except that the return value is of type **off\_t**.

14973 **RETURN VALUE**

14974 CX Upon successful completion, *ftell()* and *ftello()* shall return the current value of the file-position  
14975 indicator for the stream measured in bytes from the beginning of the file.

14976 CX Otherwise, *ftell()* and *ftello()* shall return  $-1$ , cast to **long** and **off\_t** respectively, and set *errno* to  
14977 indicate the error.

14978 **ERRORS**

14979 CX The *ftell()* and *ftello()* functions shall fail if:

14980 CX [EBADF] The file descriptor underlying *stream* is not an open file descriptor.

14981 CX [EOVERFLOW] For *ftell()*, the current file offset cannot be represented correctly in an object of  
14982 type **long**.

14983 CX [EOVERFLOW] For *ftello()*, the current file offset cannot be represented correctly in an object  
14984 of type **off\_t**.

14985 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe or FIFO.

14986 The *ftell()* function may fail if:

14987 CX [ESPIPE] The file descriptor underlying *stream* is associated with a socket.

14988 **EXAMPLES**

14989 None.

14990 **APPLICATION USAGE**

14991 None.

14992 **RATIONALE**

14993 None.

14994 **FUTURE DIRECTIONS**

14995 None.

14996 **SEE ALSO**

14997 *fgetpos()*, *fopen()*, *fseek()*, *lseek()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

14998 **CHANGE HISTORY**

14999 First released in Issue 1. Derived from Issue 1 of the SVID.

15000 **Issue 5**

15001 Large File Summit extensions are added.

15002 **Issue 6**

15003 Extensions beyond the ISO C standard are marked.

15004 The following new requirements on POSIX implementations derive from alignment with the  
15005 Single UNIX Specification:

- 15006 • The *ftello()* function is added.
- 15007 • The [Eoverflow] error conditions are added.

15008 An additional [ESPIPE] error condition is added for sockets.

15009 **NAME**15010 ftime — get date and time (**LEGACY**)15011 **SYNOPSIS**

15012 xSI #include &lt;sys/timeb.h&gt;

15013 int ftime(struct timeb \*tp);

15014

15015 **DESCRIPTION**

15016 The *ftime()* function shall set the *time* and *millitm* members of the **timeb** structure pointed to by  
 15017 *tp* to contain the seconds and milliseconds portions, respectively, of the current time in seconds  
 15018 since the Epoch. The contents of the *timezone* and *dstflag* members of *tp* after a call to *ftime()* are  
 15019 unspecified.

15020 The system clock need not have millisecond granularity. Depending on any granularity  
 15021 (particularly a granularity of one) renders code non-portable.

15022 **RETURN VALUE**15023 Upon successful completion, the *ftime()* function shall return 0; otherwise, -1 shall be returned.15024 **ERRORS**

15025 No errors are defined.

15026 **EXAMPLES**15027 **Getting the Current Time and Date**

15028 The following example shows how to get the current system time values using the *ftime()*  
 15029 function. The **timeb** structure pointed to by *tp* is filled with the current system time values for  
 15030 *time* and *millitm*.

15031 #include &lt;sys/timeb.h&gt;

15032 struct timeb tp;

15033 int status;

15034 ...

15035 status = ftime(&amp;tp);

15036 **APPLICATION USAGE**

15037 For applications portability, the *time()* function should be used to determine the current time  
 15038 instead of *ftime()*. Realtime applications should use *clock\_gettime()* to determine the current  
 15039 time instead of *ftime()*.

15040 **RATIONALE**

15041 None.

15042 **FUTURE DIRECTIONS**

15043 This function may be withdrawn in a future version.

15044 **SEE ALSO**

15045 *clock\_getres()*, *ctime()*, *gettimeofday()*, *time()*, the Base Definitions volume of  
 15046 IEEE Std 1003.1-2001, <sys/timeb.h>

15047 **CHANGE HISTORY**

15048 First released in Issue 4, Version 2.

15049 **Issue 5**

15050 Moved from X/OPEN UNIX extension to BASE.

15051 Normative text previously in the APPLICATION USAGE section is moved to the  
15052 DESCRIPTION.

15053 **Issue 6**

15054 This function is marked LEGACY.

15055 The DESCRIPTION is updated to refer to “seconds since the Epoch” rather than “seconds since  
15056 00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time*  
15057 functions.

## 15058 NAME

15059 ftok — generate an IPC key

## 15060 SYNOPSIS

15061 xSI #include &lt;sys/ipc.h&gt;

15062 key\_t ftok(const char \*path, int id);

15063

## 15064 DESCRIPTION

15065 The *ftok()* function shall return a key based on *path* and *id* that is usable in subsequent calls to  
 15066 *msgget()*, *semget()*, and *shmget()*. The application shall ensure that the *path* argument is the  
 15067 pathname of an existing file that the process is able to *stat()*.

15068 The *ftok()* function shall return the same key value for all paths that name the same file, when  
 15069 called with the same *id* value, and return different key values when called with different *id*  
 15070 values or with paths that name different files existing on the same file system at the same time. It  
 15071 is unspecified whether *ftok()* shall return the same key value when called again after the file  
 15072 named by *path* is removed and recreated with the same name.

15073 Only the low-order 8-bits of *id* are significant. The behavior of *ftok()* is unspecified if these bits  
 15074 are 0.

## 15075 RETURN VALUE

15076 Upon successful completion, *ftok()* shall return a key. Otherwise, *ftok()* shall return (**key\_t**)-1  
 15077 and set *errno* to indicate the error.

## 15078 ERRORS

15079 The *ftok()* function shall fail if:

15080 [EACCES] Search permission is denied for a component of the path prefix.

15081 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 15082 argument.

15083 [ENAMETOOLONG]  
 15084 The length of the *path* argument exceeds {PATH\_MAX} or a pathname  
 15085 component is longer than {NAME\_MAX}.

15086 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

15087 [ENOTDIR] A component of the path prefix is not a directory.

15088 The *ftok()* function may fail if:

15089 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 15090 resolution of the *path* argument.

15091 [ENAMETOOLONG]  
 15092 Pathname resolution of a symbolic link produced an intermediate result  
 15093 whose length exceeds {PATH\_MAX}.

15094 **EXAMPLES**15095 **Getting an IPC Key**

15096 The following example gets a unique key that can be used by the IPC functions *semget()*,  
 15097 *msgget()*, and *shmget()*. The key returned by *ftok()* for this example is based on the ID value *S*  
 15098 and the pathname */tmp*.

```
15099 #include <sys/ipc.h>
15100 ...
15101 key_t key;
15102 char *path = "/tmp";
15103 int id = 'S';
15104 key = ftok(path, id);
```

15105 **Saving an IPC Key**

15106 The following example gets a unique key based on the pathname */tmp* and the ID value *a*. It  
 15107 also assigns the value of the resulting key to the *semkey* variable so that it will be available to a  
 15108 later call to *semget()*, *msgget()*, or *shmget()*.

```
15109 #include <sys/ipc.h>
15110 ...
15111 key_t semkey;
15112 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
15113 perror("IPC error: ftok"); exit(1);
15114 }
```

15115 **APPLICATION USAGE**

15116 For maximum portability, *id* should be a single-byte character.

15117 **RATIONALE**

15118 None.

15119 **FUTURE DIRECTIONS**

15120 None.

15121 **SEE ALSO**

15122 *msgget()*, *semget()*, *shmget()*, the Base Definitions volume of IEEE Std 1003.1-2001, *<sys/ipc.h>*

15123 **CHANGE HISTORY**

15124 First released in Issue 4, Version 2.

15125 **Issue 5**

15126 Moved from X/OPEN UNIX extension to BASE.

15127 **Issue 6**

15128 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

15129 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
 15130 [ELOOP] error condition is added.

15131 **NAME**

15132 ftruncate — truncate a file to a specified length

15133 **SYNOPSIS**

15134 #include &lt;unistd.h&gt;

15135 int ftruncate(int *fd*, off\_t *length*);15136 **DESCRIPTION**15137 If *fd* is not a valid file descriptor open for writing, the *ftruncate()* function shall fail.

15138 If *fd* refers to a regular file, the *ftruncate()* function shall cause the size of the file to be truncated to *length*. If the size of the file previously exceeded *length*, the extra data shall no longer be available to reads on the file. If the file previously was smaller than this size, *ftruncate()* shall either increase the size of the file or fail. XSI-conformant systems shall increase the size of the file. If the file size is increased, the extended area shall appear as if it were zero-filled. The value of the seek pointer shall not be modified by a call to *ftruncate()*.

15144 Upon successful completion, if *fd* refers to a regular file, the *ftruncate()* function shall mark for update the *st\_ctime* and *st\_mtime* fields of the file and the S\_ISUID and S\_ISGID bits of the file mode may be cleared. If the *ftruncate()* function is unsuccessful, the file is unaffected.

15147 XSI If the request would cause the file size to exceed the soft file size limit for the process, the request shall fail and the implementation shall generate the SIGXFSZ signal for the thread.

15149 If *fd* refers to a directory, *ftruncate()* shall fail.15150 If *fd* refers to any other file type, except a shared memory object, the result is unspecified.

15151 SHM If *fd* refers to a shared memory object, *ftruncate()* shall set the size of the shared memory object to *length*.

15153 MF|SHM If the effect of *ftruncate()* is to decrease the size of a shared memory object or memory mapped file and whole pages beyond the new end were previously mapped, then the whole pages beyond the new end shall be discarded.

15156 MPR If the Memory Protection option is supported, references to discarded pages shall result in the generation of a SIGBUS signal; otherwise, the result of such references is undefined.

15158 MF|SHM If the effect of *ftruncate()* is to increase the size of a shared memory object, it is unspecified whether the contents of any mapped pages between the old end-of-file and the new are flushed to the underlying object.

15161 **RETURN VALUE**

15162 Upon successful completion, *ftruncate()* shall return 0; otherwise, -1 shall be returned and *errno* set to indicate the error.

15164 **ERRORS**15165 The *ftruncate()* function shall fail if:

15166 [EINTR] A signal was caught during execution.

15167 [EINVAL] The *length* argument was less than 0.

15168 [EFBIG] or [EINVAL]

15169 The *length* argument was greater than the maximum file size.

15170 XSI [EFBIG] The file is a regular file and *length* is greater than the offset maximum established in the open file description associated with *fd*.

15172 [EIO] An I/O error occurred while reading from or writing to a file system.

- 15173 [EBADF] or [EINVAL]  
 15174 The *fildest* argument is not a file descriptor open for writing.
- 15175 [EINVAL]  
 15176 The *fildest* argument references a file that was opened without write permission.
- 15177 [EROFS] The named file resides on a read-only file system.

**15178 EXAMPLES**

15179 None.

**15180 APPLICATION USAGE**

15181 None.

**15182 RATIONALE**

15183 The *ftruncate()* function is part of IEEE Std 1003.1-2001 as it was deemed to be more useful than  
 15184 *truncate()*. The *truncate()* function is provided as an XSI extension.

**15185 FUTURE DIRECTIONS**

15186 None.

**15187 SEE ALSO**

15188 *open()*, *truncate()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

**15189 CHANGE HISTORY**

15190 First released in Issue 4, Version 2.

**15191 Issue 5**

15192 Moved from X/OPEN UNIX extension to BASE and aligned with *ftruncate()* in the POSIX  
 15193 Realtime Extension. Specifically, the DESCRIPTION is extensively reworded and [EROFS] is  
 15194 added to the list of mandatory errors that can be returned by *ftruncate()*.

15195 Large File Summit extensions are added.

**15196 Issue 6**

15197 The *truncate()* function is split out into a separate reference page.

15198 The following new requirements on POSIX implementations derive from alignment with the  
 15199 Single UNIX Specification:

- 15200 • The DESCRIPTION is changed to indicate that if the file size is changed, and if the file is a  
 15201 regular file, the S\_ISUID and S\_ISGID bits in the file mode may be cleared.

15202 The following changes were made to align with the IEEE P1003.1a draft standard:

- 15203 • The DESCRIPTION text is updated.

15204 XSI-conformant systems are required to increase the size of the file if the file was previously  
 15205 smaller than the size requested.

15206 **NAME**

15207       ftrylockfile — stdio locking functions

15208 **SYNOPSIS**

15209 TSF     #include <stdio.h>

15210       int ftrylockfile(FILE \*file);

15211

15212 **DESCRIPTION**

15213       Refer to *flockfile()*.

15214 **NAME**

15215 ftw — traverse (walk) a file tree

15216 **SYNOPSIS**15217 XSI 

```
#include <ftw.h>
```

```
15218 int ftw(const char *path, int (*fn)(const char *,
15219 const struct stat *ptr, int flag), int ndirs);
15220
```

15221 **DESCRIPTION**

15222 The *ftw()* function shall recursively descend the directory hierarchy rooted in *path*. For each  
 15223 object in the hierarchy, *ftw()* shall call the function pointed to by *fn*, passing it a pointer to a  
 15224 null-terminated character string containing the name of the object, a pointer to a **stat** structure  
 15225 containing information about the object, and an integer. Possible values of the integer, defined  
 15226 in the `<ftw.h>` header, are:

15227 FTW\_D For a directory.

15228 FTW\_DNR For a directory that cannot be read.

15229 FTW\_F For a file.

15230 FTW\_SL For a symbolic link (but see also FTW\_NS below).

15231 FTW\_NS For an object other than a symbolic link on which *stat()* could not successfully be  
 15232 executed. If the object is a symbolic link and *stat()* failed, it is unspecified whether  
 15233 *ftw()* passes FTW\_SL or FTW\_NS to the user-supplied function.

15234 If the integer is FTW\_DNR, descendants of that directory shall not be processed. If the integer is  
 15235 FTW\_NS, the **stat** structure contains undefined values. An example of an object that would  
 15236 cause FTW\_NS to be passed to the function pointed to by *fn* would be a file in a directory with  
 15237 read but without execute (search) permission.

15238 The *ftw()* function shall visit a directory before visiting any of its descendants.15239 The *ftw()* function shall use at most one file descriptor for each level in the tree.15240 The argument *ndirs* should be in the range [1,{OPEN\_MAX}].

15241 The tree traversal shall continue until either the tree is exhausted, an invocation of *fn* returns a  
 15242 non-zero value, or some error, other than [EACCES], is detected within *ftw()*.

15243 The *ndirs* argument shall specify the maximum number of directory streams or file descriptors  
 15244 or both available for use by *ftw()* while traversing the tree. When *ftw()* returns it shall close any  
 15245 directory streams and file descriptors it uses not counting any opened by the application-  
 15246 supplied *fn* function.

15247 The results are unspecified if the application-supplied *fn* function does not preserve the current  
 15248 working directory.

15249 The *ftw()* function need not be reentrant. A function that is not required to be reentrant is not  
 15250 required to be thread-safe.

15251 **RETURN VALUE**

15252 If the tree is exhausted, *ftw()* shall return 0. If the function pointed to by *fn* returns a non-zero  
 15253 value, *ftw()* shall stop its tree traversal and return whatever value was returned by the function  
 15254 pointed to by *fn*. If *ftw()* detects an error, it shall return `-1` and set *errno* to indicate the error.

15255 If *ftw()* encounters an error other than [EACCES] (see FTW\_DNR and FTW\_NS above), it shall  
 15256 return `-1` and set *errno* to indicate the error. The external variable *errno* may contain any error

15257 value that is possible when a directory is opened or when one of the *stat* functions is executed on  
15258 a directory or file.

### 15259 ERRORS

15260 The *ftw()* function shall fail if:

15261 [EACCES] Search permission is denied for any component of *path* or read permission is  
15262 denied for *path*.

15263 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
15264 argument.

15265 [ENAMETOOLONG]  
15266 The length of the *path* argument exceeds {PATH\_MAX} or a pathname  
15267 component is longer than {NAME\_MAX}.

15268 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

15269 [ENOTDIR] A component of *path* is not a directory.

15270 [EOVERFLOW] A field in the *stat* structure cannot be represented correctly in the current  
15271 programming environment for one or more files found in the file hierarchy.

15272 The *ftw()* function may fail if:

15273 [EINVAL] The value of the *ndirs* argument is invalid.

15274 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
15275 resolution of the *path* argument.

15276 [ENAMETOOLONG]  
15277 Pathname resolution of a symbolic link produced an intermediate result  
15278 whose length exceeds {PATH\_MAX}.

15279 In addition, if the function pointed to by *fn* encounters system errors, *errno* may be set  
15280 accordingly.

### 15281 EXAMPLES

#### 15282 Walking a Directory Structure

15283 The following example walks the current directory structure, calling the *fn* function for every  
15284 directory entry, using at most 10 file descriptors:

```
15285 #include <ftw.h>
15286 ...
15287 if (ftw(".", fn, 10) != 0) {
15288 perror("ftw"); exit(2);
15289 }
```

### 15290 APPLICATION USAGE

15291 The *ftw()* function may allocate dynamic storage during its operation. If *ftw()* is forcibly  
15292 terminated, such as by *longjmp()* or *siglongjmp()* being executed by the function pointed to by *fn*  
15293 or an interrupt routine, *ftw()* does not have a chance to free that storage, so it remains  
15294 permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has  
15295 occurred, and arrange to have the function pointed to by *fn* return a non-zero value at its next  
15296 invocation.

15297 **RATIONALE**

15298           None.

15299 **FUTURE DIRECTIONS**

15300           None.

15301 **SEE ALSO**

15302           *longjmp()*, *lstat()*, *malloc()*, *nftw()*, *opendir()*, *siglongjmp()*, *stat()*, the Base Definitions volume of  
15303           IEEE Std 1003.1-2001, <ftw.h>, <sys/stat.h>

15304 **CHANGE HISTORY**

15305           First released in Issue 1. Derived from Issue 1 of the SVID.

15306 **Issue 5**

15307           UX codings in the DESCRIPTION, RETURN VALUE, and ERRORS sections are changed to EX.

15308 **Issue 6**

15309           The ERRORS section is updated as follows:

- 15310           • The wording of the mandatory [ELOOP] error condition is updated.
- 15311           • A second optional [ELOOP] error condition is added.
- 15312           • The [EOVERFLOW] mandatory error condition is added.

15313           Text is added to the DESCRIPTION to say that the *ftw()* function need not be reentrant and that  
15314           the results are unspecified if the application-supplied *fn* function does not preserve the current  
15315           working directory.

15316 **NAME**

15317 funlockfile — stdio locking functions

15318 **SYNOPSIS**

15319 TSF #include <stdio.h>

15320 void funlockfile(FILE \*file);

15321

15322 **DESCRIPTION**

15323 Refer to *flockfile()*.

15324 **NAME**

15325           fwide — set stream orientation

15326 **SYNOPSIS**

15327           #include &lt;stdio.h&gt;

15328           #include &lt;wchar.h&gt;

15329           int fwide(FILE \**stream*, int *mode*);15330 **DESCRIPTION**

15331 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 15332 conflict between the requirements described here and the ISO C standard is unintentional. This  
 15333 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

15334       The *fwide()* function shall determine the orientation of the stream pointed to by *stream*. If *mode* is  
 15335 greater than zero, the function first attempts to make the stream wide-oriented. If *mode* is less  
 15336 than zero, the function first attempts to make the stream byte-oriented. Otherwise, *mode* is zero  
 15337 and the function does not alter the orientation of the stream.

15338       If the orientation of the stream has already been determined, *fwide()* shall not change it.

15339 **CX**       Since no return value is reserved to indicate an error, an application wishing to check for error  
 15340 situations should set *errno* to 0, then call *fwide()*, then check *errno*, and if it is non-zero, assume  
 15341 an error has occurred.

15342 **RETURN VALUE**

15343       The *fwide()* function shall return a value greater than zero if, after the call, the stream has wide-  
 15344 orientation, a value less than zero if the stream has byte-orientation, or zero if the stream has no  
 15345 orientation.

15346 **ERRORS**

15347       The *fwide()* function may fail if:

15348 **CX**       [EBADF]       The *stream* argument is not a valid stream.

15349 **EXAMPLES**

15350       None.

15351 **APPLICATION USAGE**

15352       A call to *fwide()* with *mode* set to zero can be used to determine the current orientation of a  
 15353 stream.

15354 **RATIONALE**

15355       None.

15356 **FUTURE DIRECTIONS**

15357       None.

15358 **SEE ALSO**

15359       The Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>

15360 **CHANGE HISTORY**

15361       First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
 15362 (E).

15363 **Issue 6**

15364       Extensions beyond the ISO C standard are marked.

## 15365 NAME

15366 fwprintf, swprintf, wprintf — print formatted wide-character output

## 15367 SYNOPSIS

15368 #include &lt;stdio.h&gt;

15369 #include &lt;wchar.h&gt;

15370 int fwprintf(FILE \*restrict *stream*, const wchar\_t \*restrict *format*, ...);15371 int swprintf(wchar\_t \*restrict *ws*, size\_t *n*,15372 const wchar\_t \*restrict *format*, ...);15373 int wprintf(const wchar\_t \*restrict *format*, ...);

## 15374 DESCRIPTION

15375 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 15376 conflict between the requirements described here and the ISO C standard is unintentional. This  
 15377 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

15378 The *fwprintf()* function shall place output on the named output *stream*. The *wprintf()* function  
 15379 shall place output on the standard output stream *stdout*. The *swprintf()* function shall place  
 15380 output followed by the null wide character in consecutive wide characters starting at *\*ws*; no  
 15381 more than *n* wide characters shall be written, including a terminating null wide character, which  
 15382 is always added (unless *n* is zero).

15383 Each of these functions shall convert, format, and print its arguments under control of the *format*  
 15384 wide-character string. The *format* is composed of zero or more directives: *ordinary wide-*  
 15385 *characters*, which are simply copied to the output stream, and *conversion specifications*, each of  
 15386 which results in the fetching of zero or more arguments. The results are undefined if there are  
 15387 insufficient arguments for the *format*. If the *format* is exhausted while arguments remain, the  
 15388 excess arguments are evaluated but are otherwise ignored.

15389 XSI Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than  
 15390 to the next unused argument. In this case, the conversion specifier wide character % (see below)  
 15391 is replaced by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL\_ARGMAX}],  
 15392 giving the position of the argument in the argument list. This feature provides for the definition  
 15393 of *format* wide-character strings that select arguments in an order appropriate to specific  
 15394 languages (see the EXAMPLES section).

15395 The *format* can contain either numbered argument specifications (that is, "%n\$" and "\*m\$"), or  
 15396 unnumbered argument conversion specifications (that is, % and \*), but not both. The only  
 15397 exception to this is that %% can be mixed with the "%n\$" form. The results of mixing numbered  
 15398 and unnumbered argument specifications in a *format* wide-character string are undefined. When  
 15399 numbered argument specifications are used, specifying the *N*th argument requires that all the  
 15400 leading arguments, from the first to the (*N*-1)th, are specified in the *format* wide-character string.

15401 In *format* wide-character strings containing the "%n\$" form of conversion specification,  
 15402 numbered arguments in the argument list can be referenced from the *format* wide-character  
 15403 string as many times as required.

15404 In *format* wide-character strings containing the % form of conversion specification, each  
 15405 argument in the argument list shall be used exactly once.

15406 CX All forms of the *fwprintf()* function allow for the insertion of a locale-dependent radix character  
 15407 in the output string, output as a wide-character value. The radix character is defined in the  
 15408 program's locale (category *LC\_NUMERIC*). In the POSIX locale, or in a locale where the radix  
 15409 character is not defined, the radix character shall default to a period ( '.' ).

15410 XSI Each conversion specification is introduced by the '%' wide character or by the wide-character  
 15411 sequence "%n\$", after which the following appear in sequence:

- 15412           • Zero or more *flags* (in any order), which modify the meaning of the conversion specification.
- 15413           • An optional minimum *field width*. If the converted value has fewer wide characters than the  
15414           field width, it shall be padded with spaces by default on the left; it shall be padded on the  
15415           right, if the left-adjustment flag ('-'), described below, is given to the field width. The field  
15416           width takes the form of an asterisk ('\*'), described below, or a decimal integer.
- 15417           • An optional *precision* that gives the minimum number of digits to appear for the d, i, o, u, x,  
15418           and X conversion specifiers; the number of digits to appear after the radix character for the a,  
15419           A, e, E, f, and F conversion specifiers; the maximum number of significant digits for the g  
15420           and G conversion specifiers; or the maximum number of wide characters to be printed from a  
15421           string in the s conversion specifiers. The precision takes the form of a period ('.') followed  
15422           either by an asterisk ('\*'), described below, or an optional decimal digit string, where a null  
15423           digit string is treated as 0. If a precision appears with any other conversion wide character,  
15424           the behavior is undefined.
- 15425           • An optional length modifier that specifies the size of the argument.
- 15426           • A *conversion specifier* wide character that indicates the type of conversion to be applied.
- 15427           A field width, or precision, or both, may be indicated by an asterisk ('\*'). In this case an  
15428           argument of type **int** supplies the field width or precision. Applications shall ensure that  
15429           arguments specifying field width, or precision, or both appear in that order before the argument,  
15430           if any, to be converted. A negative field width is taken as a '-' flag followed by a positive field  
15431           width. A negative precision is taken as if the precision were omitted. In *format* wide-character  
15432           strings containing the "%n\$" form of a conversion specification, a field width or precision may  
15433           be indicated by the sequence "\*m\$", where *m* is a decimal integer in the range  
15434           [1,{NL\_ARGMAX}] giving the position in the argument list (after the *format* argument) of an  
15435           integer argument containing the field width or precision, for example:
- 15436           

```
wprintf(L"%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```
- 15437           The flag wide characters and their meanings are:
- 15438           '           The integer portion of the result of a decimal conversion (%i, %d, %u, %f, %F, %g, or %G)  
15439           shall be formatted with thousands' grouping wide characters. For other conversions,  
15440           the behavior is undefined. The numeric grouping wide character is used.
- 15441           -           The result of the conversion shall be left-justified within the field. The conversion shall  
15442           be right-justified if this flag is not specified.
- 15443           +           The result of a signed conversion shall always begin with a sign ('+' or '-'). The  
15444           conversion shall begin with a sign only when a negative value is converted if this flag is  
15445           not specified.
- 15446           <space>       If the first wide character of a signed conversion is not a sign, or if a signed conversion  
15447           results in no wide characters, a <space> shall be prefixed to the result. This means that  
15448           if the <space> and '+' flags both appear, the <space> flag shall be ignored.
- 15449           #           Specifies that the value is to be converted to an alternative form. For o conversion, it  
15450           increases the precision (if necessary) to force the first digit of the result to be 0. For x or  
15451           X conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For a, A, e,  
15452           E, f, F, g, and G conversion specifiers, the result shall always contain a radix character,  
15453           even if no digits follow it. Without this flag, a radix character appears in the result of  
15454           these conversions only if a digit follows it. For g and G conversion specifiers, trailing  
15455           zeros shall *not* be removed from the result as they normally are. For other conversion  
15456           specifiers, the behavior is undefined.

15457 0 For `d`, `i`, `o`, `u`, `x`, `X`, `a`, `A`, `e`, `E`, `f`, `F`, `g`, and `G` conversion specifiers, leading zeros  
 15458 (following any indication of sign or base) are used to pad to the field width; no space  
 15459 padding is performed. If the `'0'` and `'-'` flags both appear, the `'0'` flag shall be  
 15460 ignored. For `d`, `i`, `o`, `u`, `x`, and `X` conversion specifiers, if a precision is specified, the `'0'`  
 15461 flag shall be ignored. If the `'0'` and `' '` flags both appear, the grouping wide  
 15462 characters are inserted before zero padding. For other conversions, the behavior is  
 15463 undefined.

15464 The length modifiers and their meanings are:

15465 `hh` Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **signed char**  
 15466 or **unsigned char** argument (the argument will have been promoted according to the  
 15467 integer promotions, but its value shall be converted to **signed char** or **unsigned char**  
 15468 before printing); or that a following `n` conversion specifier applies to a pointer to a  
 15469 **signed char** argument.

15470 `h` Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **short** or  
 15471 **unsigned short** argument (the argument will have been promoted according to the  
 15472 integer promotions, but its value shall be converted to **short** or **unsigned short** before  
 15473 printing); or that a following `n` conversion specifier applies to a pointer to a **short**  
 15474 argument.

15475 `l` (`ell`) Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **long** or  
 15476 **unsigned long** argument; that a following `n` conversion specifier applies to a pointer to  
 15477 a **long** argument; that a following `c` conversion specifier applies to a **wint\_t** argument;  
 15478 that a following `s` conversion specifier applies to a pointer to a **wchar\_t** argument; or  
 15479 has no effect on a following `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G` conversion specifier.

15480 `ll` (`ell-ell`)

15481 Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **long long** or  
 15482 **unsigned long long** argument; or that a following `n` conversion specifier applies to a  
 15483 pointer to a **long long** argument.

15484 `j` Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to an **intmax\_t**  
 15485 or **uintmax\_t** argument; or that a following `n` conversion specifier applies to a pointer  
 15486 to an **intmax\_t** argument.

15487 `z` Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **size\_t** or the  
 15488 corresponding signed integer type argument; or that a following `n` conversion specifier  
 15489 applies to a pointer to a signed integer type corresponding to a **size\_t** argument.

15490 `t` Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **ptrdiff\_t** or  
 15491 the corresponding **unsigned** type argument; or that a following `n` conversion specifier  
 15492 applies to a pointer to a **ptrdiff\_t** argument.

15493 `L` Specifies that a following `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G` conversion specifier applies to a **long**  
 15494 **double** argument.

15495 If a length modifier appears with any conversion specifier other than as specified above, the  
 15496 behavior is undefined.

15497 The conversion specifiers and their meanings are:

15498 `d`, `i` The **int** argument shall be converted to a signed decimal in the style "`[-] dddd`". The  
 15499 precision specifies the minimum number of digits to appear; if the value being  
 15500 converted can be represented in fewer digits, it shall be expanded with leading zeros.  
 15501 The default precision shall be 1. The result of converting zero with an explicit precision  
 15502 of zero shall be no wide characters.

- 15503           o       The **unsigned** argument shall be converted to unsigned octal format in the style  
15504                    "ddd". The precision specifies the minimum number of digits to appear; if the value  
15505                    being converted can be represented in fewer digits, it shall be expanded with leading  
15506                    zeros. The default precision shall be 1. The result of converting zero with an explicit  
15507                    precision of zero shall be no wide characters.
- 15508           u       The **unsigned** argument shall be converted to unsigned decimal format in the style  
15509                    "ddd". The precision specifies the minimum number of digits to appear; if the value  
15510                    being converted can be represented in fewer digits, it shall be expanded with leading  
15511                    zeros. The default precision shall be 1. The result of converting zero with an explicit  
15512                    precision of zero shall be no wide characters.
- 15513           x       The **unsigned** argument shall be converted to unsigned hexadecimal format in the style  
15514                    "ddd"; the letters "abcdef" are used. The precision specifies the minimum number  
15515                    of digits to appear; if the value being converted can be represented in fewer digits, it  
15516                    shall be expanded with leading zeros. The default precision shall be 1. The result of  
15517                    converting zero with an explicit precision of zero shall be no wide characters.
- 15518           X       Equivalent to the x conversion specifier, except that letters "ABCDEF" are used instead  
15519                    of "abcdef".
- 15520           f, F     The **double** argument shall be converted to decimal notation in the style  
15521                    "[-]ddd.ddd", where the number of digits after the radix character shall be equal to  
15522                    the precision specification. If the precision is missing, it shall be taken as 6; if the  
15523                    precision is explicitly zero and no '#' flag is present, no radix character shall appear. If  
15524                    a radix character appears, at least one digit shall appear before it. The value shall be  
15525                    rounded in an implementation-defined manner to the appropriate number of digits.
- 15526                    A **double** argument representing an infinity shall be converted in one of the styles  
15527                    "[-]inf" or "[-]infinity"; which style is implementation-defined. A **double**  
15528                    argument representing a NaN shall be converted in one of the styles "[-]nan" or  
15529                    "[-]nan(*n-char-sequence*)"; which style, and the meaning of any *n-char-sequence*,  
15530                    is implementation-defined. The F conversion specifier produces "INF", "INFINITY",  
15531                    or "NAN" instead of "inf", "infinity", or "nan", respectively.
- 15532           e, E     The **double** argument shall be converted in the style "[-]d.ddde±dd", where there  
15533                    shall be one digit before the radix character (which is non-zero if the argument is non-  
15534                    zero) and the number of digits after it shall be equal to the precision; if the precision is  
15535                    missing, it shall be taken as 6; if the precision is zero and no '#' flag is present, no  
15536                    radix character shall appear. The value shall be rounded in an implementation-defined  
15537                    manner to the appropriate number of digits. The E conversion wide character shall  
15538                    produce a number with 'E' instead of 'e' introducing the exponent. The exponent  
15539                    shall always contain at least two digits. If the value is zero, the exponent shall be zero.
- 15540                    A **double** argument representing an infinity or NaN shall be converted in the style of  
15541                    an f or F conversion specifier.
- 15542           g, G     The **double** argument shall be converted in the style f or e (or in the style F or E in the  
15543                    case of a G conversion specifier), with the precision specifying the number of significant  
15544                    digits. If an explicit precision is zero, it shall be taken as 1. The style used depends on  
15545                    the value converted; style e (or E) shall be used only if the exponent resulting from  
15546                    such a conversion is less than -4 or greater than or equal to the precision. Trailing zeros  
15547                    shall be removed from the fractional portion of the result; a radix character shall appear  
15548                    only if it is followed by a digit.
- 15549                    A **double** argument representing an infinity or NaN shall be converted in the style of  
15550                    an f or F conversion specifier.

|       |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 15551 | a, A | A <b>double</b> argument representing a floating-point number shall be converted in the style "[−] 0xh.hhhhp±d", where there shall be one hexadecimal digit (which is non-zero if the argument is a normalized floating-point number and is otherwise unspecified) before the decimal-point wide character and the number of hexadecimal digits after it shall be equal to the precision; if the precision is missing and FLT_RADIX is a power of 2, then the precision shall be sufficient for an exact representation of the value; if the precision is missing and FLT_RADIX is not a power of 2, then the precision shall be sufficient to distinguish values of type <b>double</b> , except that trailing zeros may be omitted; if the precision is zero and the '#' flag is not specified, no decimal-point wide character shall appear. The letters "abcdef" are used for a conversion and the letters "ABCDEF" for A conversion. The A conversion specifier produces a number with 'X' and 'P' instead of 'x' and 'p'. The exponent shall always contain at least one digit, and only as many more digits as necessary to represent the decimal exponent of 2. If the value is zero, the exponent shall be zero. |
| 15552 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15553 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15554 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15555 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15556 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15557 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15558 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15559 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15560 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15561 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15562 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15563 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15564 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15565 |      | A <b>double</b> argument representing an infinity or NaN shall be converted in the style of an f or F conversion specifier.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 15566 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15567 | c    | If no l (ell) qualifier is present, the <b>int</b> argument shall be converted to a wide character as if by calling the <i>btowc()</i> function and the resulting wide character shall be written. Otherwise, the <b>wint_t</b> argument shall be converted to <b>wchar_t</b> , and written.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 15568 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15569 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15570 | s    | If no l (ell) qualifier is present, the application shall ensure that the argument is a pointer to a character array containing a character sequence beginning in the initial shift state. Characters from the array shall be converted as if by repeated calls to the <i>mbrtowc()</i> function, with the conversion state described by an <b>mbstate_t</b> object initialized to zero before the first character is converted, and written up to (but not including) the terminating null wide character. If the precision is specified, no more than that many wide characters shall be written. If the precision is not specified, or is greater than the size of the array, the application shall ensure that the array contains a null wide character.                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 15571 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15572 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15573 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15574 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15575 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15576 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15577 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15578 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15579 |      | If an l (ell) qualifier is present, the application shall ensure that the argument is a pointer to an array of type <b>wchar_t</b> . Wide characters from the array shall be written up to (but not including) a terminating null wide character. If no precision is specified, or is greater than the size of the array, the application shall ensure that the array contains a null wide character. If a precision is specified, no more than that many wide characters shall be written.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 15580 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15581 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15582 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15583 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15584 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15585 | p    | The application shall ensure that the argument is a pointer to <b>void</b> . The value of the pointer shall be converted to a sequence of printable wide characters in an implementation-defined manner.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 15586 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15587 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15588 | n    | The application shall ensure that the argument is a pointer to an integer into which is written the number of wide characters written to the output so far by this call to one of the <i>fwprintf()</i> functions. No argument shall be converted, but one shall be consumed. If the conversion specification includes any flags, a field width, or a precision, the behavior is undefined.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 15589 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15590 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15591 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15592 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 15593 | XSI  | C Equivalent to lc.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 15594 | XSI  | S Equivalent to ls.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 15595 | %    | Output a '%' wide character; no argument shall be converted. The entire conversion specification shall be %%.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 15596 |      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

15597 If a conversion specification does not match one of the above forms, the behavior is undefined.  
 15598 In no case does a nonexistent or small field width cause truncation of a field; if the result of a  
 15599 conversion is wider than the field width, the field shall be expanded to contain the conversion  
 15600 result. Characters generated by *fwprintf()* and *wprintf()* shall be printed as if *fputwc()* had been  
 15601 called.

15602 For *a* and *A* conversions, if *FLT\_RADIX* is not a power of 2 and the result is not exactly  
 15603 representable in the given precision, the result should be one of the two adjacent numbers in  
 15604 hexadecimal floating style with the given precision, with the extra stipulation that the error  
 15605 should have a correct sign for the current rounding direction.

15606 For *e*, *E*, *f*, *F*, *g*, and *G* conversion specifiers, if the number of significant decimal digits is at most  
 15607 *DECIMAL\_DIG*, then the result should be correctly rounded. If the number of significant  
 15608 decimal digits is more than *DECIMAL\_DIG* but the source value is exactly representable with  
 15609 *DECIMAL\_DIG* digits, then the result should be an exact representation with trailing zeros.  
 15610 Otherwise, the source value is bounded by two adjacent decimal strings  $L < U$ , both having  
 15611 *DECIMAL\_DIG* significant digits; the value of the resultant decimal string  $D$  should satisfy  $L \leq$   
 15612  $D \leq U$ , with the extra stipulation that the error should have a correct sign for the current  
 15613 rounding direction.

15614 CX The *st\_ctime* and *st\_mtime* fields of the file shall be marked for update between the call to a  
 15615 successful execution of *fwprintf()* or *wprintf()* and the next successful completion of a call to  
 15616 *fflush()* or *fclose()* on the same stream, or a call to *exit()* or *abort()*.

#### 15617 RETURN VALUE

15618 Upon successful completion, these functions shall return the number of wide characters  
 15619 transmitted, excluding the terminating null wide character in the case of *swprintf()*, or a negative  
 15620 CX value if an output error was encountered, and set *errno* to indicate the error.

15621 If *n* or more wide characters were requested to be written, *swprintf()* shall return a negative  
 15622 CX value, and set *errno* to indicate the error.

#### 15623 ERRORS

15624 For the conditions under which *fwprintf()* and *wprintf()* fail and may fail, refer to *fputwc()*.

15625 In addition, all forms of *fwprintf()* may fail if:

15626 XSI [EILSEQ] A wide-character code that does not correspond to a valid character has been  
 15627 detected.

15628 XSI [EINVAL] There are insufficient arguments.

15629 In addition, *wprintf()* and *fwprintf()* may fail if:

15630 XSI [ENOMEM] Insufficient storage space is available.

#### 15631 EXAMPLES

15632 To print the language-independent date and time format, the following statement could be used:

```
15633 wprintf(format, weekday, month, day, hour, min);
```

15634 For American usage, *format* could be a pointer to the wide-character string:

```
15635 L"%s, %s %d, %d:%.2d\n"
```

15636 producing the message:

```
15637 Sunday, July 3, 10:02
```

15638 whereas for German usage, *format* could be a pointer to the wide-character string:

15639 L"%1\$s, %3\$d. %2\$s, %4\$d:%5\$.2d\n"

15640 producing the message:

15641 Sonntag, 3. Juli, 10:02

15642 **APPLICATION USAGE**

15643 None.

15644 **RATIONALE**

15645 None.

15646 **FUTURE DIRECTIONS**

15647 None.

15648 **SEE ALSO**

15649 *btowc()*, *fputwc()*, *fwscanf()*, *mbrtowc()*, *setlocale()*, the Base Definitions volume of  
15650 IEEE Std 1003.1-2001, Chapter 7, Locale, <stdio.h>, <wchar.h>

15651 **CHANGE HISTORY**

15652 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
15653 (E).

15654 **Issue 6**

15655 The Open Group Corrigendum U040/1 is applied to the RETURN VALUE section, describing  
15656 the case if *n* or more wide characters are requested to be written using *swprintf()*.

15657 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

15658 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 15659 • The prototypes for *fwprintf()*, *swprintf()*, and *wprintf()* are updated.
- 15660 • The DESCRIPTION is updated.
- 15661 • The hh, ll, j, t, and z length modifiers are added.
- 15662 • The a, A, and F conversion characters are added.
- 15663 • XSI shading is removed from the description of character string representations of infinity  
15664 and NaN floating-point values.

15665 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion  
15666 specification” consistently.

15667 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

15668 **NAME**

15669        fwrite — binary output

15670 **SYNOPSIS**

15671        #include &lt;stdio.h&gt;

15672        size\_t fwrite(const void \*restrict ptr, size\_t size, size\_t nitems,  
15673                      FILE \*restrict stream);15674 **DESCRIPTION**15675 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
15676 conflict between the requirements described here and the ISO C standard is unintentional. This  
15677 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.15678        The *fwrite()* function shall write, from the array pointed to by *ptr*, up to *nitems* elements whose  
15679 size is specified by *size*, to the stream pointed to by *stream*. For each object, *size* calls shall be  
15680 made to the *fputc()* function, taking the values (in order) from an array of **unsigned char** exactly  
15681 overlaying the object. The file-position indicator for the stream (if defined) shall be advanced by  
15682 the number of bytes successfully written. If an error occurs, the resulting value of the file-  
15683 position indicator for the stream is unspecified.15684 **CX**        The *st\_ctime* and *st\_mtime* fields of the file shall be marked for update between the successful  
15685 execution of *fwrite()* and the next successful completion of a call to *fflush()* or *fclose()* on the  
15686 same stream, or a call to *exit()* or *abort()*.15687 **RETURN VALUE**15688        The *fwrite()* function shall return the number of elements successfully written, which may be  
15689 less than *nitems* if a write error is encountered. If *size* or *nitems* is 0, *fwrite()* shall return 0 and the  
15690 state of the stream remains unchanged. Otherwise, if a write error occurs, the error indicator for  
15691 **CX**        the stream shall be set, and *errno* shall be set to indicate the error.15692 **ERRORS**15693        Refer to *fputc()*.15694 **EXAMPLES**

15695        None.

15696 **APPLICATION USAGE**15697        Because of possible differences in element length and byte ordering, files written using *fwrite()*  
15698 are application-dependent, and possibly cannot be read using *fread()* by a different application  
15699 or by the same application on a different processor.15700 **RATIONALE**

15701        None.

15702 **FUTURE DIRECTIONS**

15703        None.

15704 **SEE ALSO**15705        *ferror()*, *fopen()*, *printf()*, *putc()*, *puts()*, *write()*, the Base Definitions volume of  
15706 IEEE Std 1003.1-2001, <stdio.h>15707 **CHANGE HISTORY**

15708        First released in Issue 1. Derived from Issue 1 of the SVID.

15709 **Issue 6**

15710        Extensions beyond the ISO C standard are marked.

15711        The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

15712

- The *fwrite()* prototype is updated.

15713

- The DESCRIPTION is updated to clarify how the data is written out using *fputc()*.

## 15714 NAME

15715 fwscanf, swscanf, wscanf — convert formatted wide-character input

## 15716 SYNOPSIS

15717 #include &lt;stdio.h&gt;

15718 #include &lt;wchar.h&gt;

15719 int fwscanf(FILE \*restrict *stream*, const wchar\_t \*restrict *format*, ... );15720 int swscanf(const wchar\_t \*restrict *ws*,15721 const wchar\_t \*restrict *format*, ... );15722 int wscanf(const wchar\_t \*restrict *format*, ... );

## 15723 DESCRIPTION

15724 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 15725 conflict between the requirements described here and the ISO C standard is unintentional. This  
 15726 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

15727 The *fwscanf()* function shall read from the named input *stream*. The *wscanf()* function shall read  
 15728 from the standard input stream *stdin*. The *swscanf()* function shall read from the wide-character  
 15729 string *ws*. Each function reads wide characters, interprets them according to a format, and stores  
 15730 the results in its arguments. Each expects, as arguments, a control wide-character string *format*  
 15731 described below, and a set of *pointer* arguments indicating where the converted input should be  
 15732 stored. The result is undefined if there are insufficient arguments for the format. If the *format* is  
 15733 exhausted while arguments remain, the excess arguments are evaluated but are otherwise  
 15734 ignored.

15735 XSI Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than  
 15736 to the next unused argument. In this case, the conversion specifier wide character % (see below)  
 15737 is replaced by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL\_ARGMAX}].  
 15738 This feature provides for the definition of *format* wide-character strings that select arguments in  
 15739 an order appropriate to specific languages. In *format* wide-character strings containing the  
 15740 "%n\$" form of conversion specifications, it is unspecified whether numbered arguments in the  
 15741 argument list can be referenced from the *format* wide-character string more than once.

15742 The *format* can contain either form of a conversion specification—that is, % or "%n\$" — but the  
 15743 two forms cannot normally be mixed within a single *format* wide-character string. The only  
 15744 exception to this is that %% or %\* can be mixed with the "%n\$" form. When numbered  
 15745 argument specifications are used, specifying the *N*th argument requires that all the leading  
 15746 arguments, from the first to the (*N*–1)th, are pointers.

15747 CX The *fwscanf()* function in all its forms allows for detection of a language-dependent radix  
 15748 character in the input string, encoded as a wide-character value. The radix character is defined in  
 15749 the program's locale (category *LC\_NUMERIC*). In the POSIX locale, or in a locale where the  
 15750 radix character is not defined, the radix character shall default to a period ( '.' ).

15751 The *format* is a wide-character string composed of zero or more directives. Each directive is  
 15752 composed of one of the following: one or more white-space wide characters (<space>s, <tab>s,  
 15753 <newline>s, <vertical-tab>s, or <form-feed>s); an ordinary wide character (neither '%' nor a  
 15754 white-space character); or a conversion specification. Each conversion specification is introduced  
 15755 XSI by a '%' or the sequence "%n\$" after which the following appear in sequence:

- 15756 • An optional assignment-suppressing character ' \* '.
- 15757 • An optional non-zero decimal integer that specifies the maximum field width.
- 15758 • An optional length modifier that specifies the size of the receiving object.

- 15759           • A conversion specifier wide character that specifies the type of conversion to be applied. The  
15760           valid conversion specifiers are described below.
- 15761           The *fwscanf()* functions shall execute each directive of the format in turn. If a directive fails, as  
15762           detailed below, the function shall return. Failures are described as input failures (due to the  
15763           unavailability of input bytes) or matching failures (due to inappropriate input).
- 15764           A directive composed of one or more white-space wide characters is executed by reading input  
15765           until no more valid input can be read, or up to the first wide character which is not a white-  
15766           space wide character, which remains unread.
- 15767           A directive that is an ordinary wide character shall be executed as follows. The next wide  
15768           character is read from the input and compared with the wide character that comprises the  
15769           directive; if the comparison shows that they are not equivalent, the directive shall fail, and the  
15770           differing and subsequent wide characters remain unread. Similarly, if end-of-file, an encoding  
15771           error, or a read error prevents a wide character from being read, the directive shall fail.
- 15772           A directive that is a conversion specification defines a set of matching input sequences, as  
15773           described below for each conversion wide character. A conversion specification is executed in  
15774           the following steps.
- 15775           Input white-space wide characters (as specified by *isspace()*) shall be skipped, unless the  
15776           conversion specification includes a `['', c, or n` conversion specifier.
- 15777           An item shall be read from the input, unless the conversion specification includes an `n`  
15778           conversion specifier wide character. An input item is defined as the longest sequence of input  
15779           wide characters, not exceeding any specified field width, which is an initial subsequence of a  
15780           matching sequence. The first wide character, if any, after the input item shall remain unread. If  
15781           the length of the input item is zero, the execution of the conversion specification shall fail; this  
15782           condition is a matching failure, unless end-of-file, an encoding error, or a read error prevented  
15783           input from the stream, in which case it is an input failure.
- 15784           Except in the case of a `%` conversion specifier, the input item (or, in the case of a `%n` conversion  
15785           specification, the count of input wide characters) shall be converted to a type appropriate to the  
15786           conversion wide character. If the input item is not a matching sequence, the execution of the  
15787           conversion specification shall fail; this condition is a matching failure. Unless assignment  
15788           suppression was indicated by a `'*'`, the result of the conversion shall be placed in the object  
15789           pointed to by the first argument following the *format* argument that has not already received a  
15790 XSI           conversion result if the conversion specification is introduced by `%`, or in the *n*th argument if  
15791           introduced by the wide-character sequence `"%n$"`. If this object does not have an appropriate  
15792           type, or if the result of the conversion cannot be represented in the space provided, the behavior  
15793           is undefined.
- 15794           The length modifiers and their meanings are:
- 15795           hh       Specifies that a following `d, i, o, u, x, X, or n` conversion specifier applies to an  
15796           argument with type pointer to **signed char** or **unsigned char**.
- 15797           h        Specifies that a following `d, i, o, u, x, X, or n` conversion specifier applies to an  
15798           argument with type pointer to **short** or **unsigned short**.
- 15799           l (ell) Specifies that a following `d, i, o, u, x, X, or n` conversion specifier applies to an  
15800           argument with type pointer to **long** or **unsigned long**; that a following `a, A, e, E, f, F, g,`  
15801           or `G` conversion specifier applies to an argument with type pointer to **double**; or that a  
15802           following `c, s,` or `['` conversion specifier applies to an argument with type pointer to  
15803           **wchar\_t**.

|       |              |                                                                                                         |
|-------|--------------|---------------------------------------------------------------------------------------------------------|
| 15804 | ll (ell-ell) |                                                                                                         |
| 15805 |              | Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an                    |
| 15806 |              | argument with type pointer to <b>long long</b> or <b>unsigned long long</b> .                           |
| 15807 | j            | Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an                    |
| 15808 |              | argument with type pointer to <b>intmax_t</b> or <b>uintmax_t</b> .                                     |
| 15809 | z            | Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an                    |
| 15810 |              | argument with type pointer to <b>size_t</b> or the corresponding signed integer type.                   |
| 15811 | t            | Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an                    |
| 15812 |              | argument with type pointer to <b>ptrdiff_t</b> or the corresponding <b>unsigned</b> type.               |
| 15813 | L            | Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to an                 |
| 15814 |              | argument with type pointer to <b>long double</b> .                                                      |
| 15815 |              | If a length modifier appears with any conversion specifier other than as specified above, the           |
| 15816 |              | behavior is undefined.                                                                                  |
| 15817 |              | The following conversion specifier wide characters are valid:                                           |
| 15818 | d            | Matches an optionally signed decimal integer, whose format is the same as expected for                  |
| 15819 |              | the subject sequence of <i>wcstol()</i> with the value 10 for the <i>base</i> argument. In the absence  |
| 15820 |              | of a size modifier, the application shall ensure that the corresponding argument is a                   |
| 15821 |              | pointer to <b>int</b> .                                                                                 |
| 15822 | i            | Matches an optionally signed integer, whose format is the same as expected for the                      |
| 15823 |              | subject sequence of <i>wcstol()</i> with 0 for the <i>base</i> argument. In the absence of a size       |
| 15824 |              | modifier, the application shall ensure that the corresponding argument is a pointer to                  |
| 15825 |              | <b>int</b> .                                                                                            |
| 15826 | o            | Matches an optionally signed octal integer, whose format is the same as expected for                    |
| 15827 |              | the subject sequence of <i>wcstoul()</i> with the value 8 for the <i>base</i> argument. In the absence  |
| 15828 |              | of a size modifier, the application shall ensure that the corresponding argument is a                   |
| 15829 |              | pointer to <b>unsigned</b> .                                                                            |
| 15830 | u            | Matches an optionally signed decimal integer, whose format is the same as expected for                  |
| 15831 |              | the subject sequence of <i>wcstoul()</i> with the value 10 for the <i>base</i> argument. In the absence |
| 15832 |              | of a size modifier, the application shall ensure that the corresponding argument is a                   |
| 15833 |              | pointer to <b>unsigned</b> .                                                                            |
| 15834 | x            | Matches an optionally signed hexadecimal integer, whose format is the same as                           |
| 15835 |              | expected for the subject sequence of <i>wcstoul()</i> with the value 16 for the <i>base</i> argument.   |
| 15836 |              | In the absence of a size modifier, the application shall ensure that the corresponding                  |
| 15837 |              | argument is a pointer to <b>unsigned</b> .                                                              |
| 15838 | a, e, f, g   |                                                                                                         |
| 15839 |              | Matches an optionally signed floating-point number, infinity, or NaN whose format is                    |
| 15840 |              | the same as expected for the subject sequence of <i>wcstod()</i> . In the absence of a size             |
| 15841 |              | modifier, the application shall ensure that the corresponding argument is a pointer to                  |
| 15842 |              | <b>float</b> .                                                                                          |
| 15843 |              | If the <i>fwprintf()</i> family of functions generates character string representations for             |
| 15844 |              | infinity and NaN (a symbolic entity encoded in floating-point format) to support                        |
| 15845 |              | IEEE Std 754-1985, the <i>fwscanf()</i> family of functions shall recognize them as input.              |
| 15846 | s            | Matches a sequence of non white-space wide characters. If no l (ell) qualifier is present,              |
| 15847 |              | characters from the input field shall be converted as if by repeated calls to the                       |
| 15848 |              | <i>wcrtomb()</i> function, with the conversion state described by an <b>mbstate_t</b> object            |

15849 initialized to zero before the first wide character is converted. The application shall  
 15850 ensure that the corresponding argument is a pointer to a character array large enough  
 15851 to accept the sequence and the terminating null character, which shall be added  
 15852 automatically.

15853 Otherwise, the application shall ensure that the corresponding argument is a pointer to  
 15854 an array of **wchar\_t** large enough to accept the sequence and the terminating null wide  
 15855 character, which shall be added automatically.

15856 [ Matches a non-empty sequence of wide characters from a set of expected wide  
 15857 characters (the *scanset*). If no **l** (ell) qualifier is present, wide characters from the input  
 15858 field shall be converted as if by repeated calls to the *wcrtomb()* function, with the  
 15859 conversion state described by an **mbstate\_t** object initialized to zero before the first  
 15860 wide character is converted. The application shall ensure that the corresponding  
 15861 argument is a pointer to a character array large enough to accept the sequence and the  
 15862 terminating null character, which shall be added automatically.

15863 If an **l** (ell) qualifier is present, the application shall ensure that the corresponding  
 15864 argument is a pointer to an array of **wchar\_t** large enough to accept the sequence and  
 15865 the terminating null wide character, which shall be added automatically.

15866 The conversion specification includes all subsequent wide characters in the *format*  
 15867 string up to and including the matching right square bracket (']'). The wide  
 15868 characters between the square brackets (the *scanlist*) comprise the scanset, unless the  
 15869 wide character after the left square bracket is a circumflex ('^'), in which case the  
 15870 scanset contains all wide characters that do not appear in the scanlist between the  
 15871 circumflex and the right square bracket. If the conversion specification begins with  
 15872 "[ ]" or "[^]", the right square bracket is included in the scanlist and the next right  
 15873 square bracket is the matching right square bracket that ends the conversion  
 15874 specification; otherwise, the first right square bracket is the one that ends the  
 15875 conversion specification. If a '-' is in the scanlist and is not the first wide character,  
 15876 nor the second where the first wide character is a '^', nor the last wide character, the  
 15877 behavior is implementation-defined.

15878 c Matches a sequence of wide characters of exactly the number specified by the field  
 15879 width (1 if no field width is present in the conversion specification).

15880 If no **l** (ell) length modifier is present, characters from the input field shall be converted  
 15881 as if by repeated calls to the *wcrtomb()* function, with the conversion state described by  
 15882 an **mbstate\_t** object initialized to zero before the first wide character is converted. The  
 15883 corresponding argument shall be a pointer to the initial element of a character array  
 15884 large enough to accept the sequence. No null character is added.

15885 If an **l** (ell) length modifier is present, the corresponding argument shall be a pointer to  
 15886 the initial element of an array of **wchar\_t** large enough to accept the sequence. No null  
 15887 wide character is added.

15888 Otherwise, the application shall ensure that the corresponding argument is a pointer to  
 15889 an array of **wchar\_t** large enough to accept the sequence. No null wide character is  
 15890 added.

15891 p Matches an implementation-defined set of sequences, which shall be the same as the set  
 15892 of sequences that is produced by the %p conversion specification of the corresponding  
 15893 *fwprintf()* functions. The application shall ensure that the corresponding argument is a  
 15894 pointer to a pointer to **void**. The interpretation of the input item is implementation-  
 15895 defined. If the input item is a value converted earlier during the same program  
 15896 execution, the pointer that results shall compare equal to that value; otherwise, the

- 15897 behavior of the %p conversion is undefined.
- 15898 n No input is consumed. The application shall ensure that the corresponding argument is  
15899 a pointer to the integer into which is to be written the number of wide characters read  
15900 from the input so far by this call to the *fwscanf()* functions. Execution of a %n  
15901 conversion specification shall not increment the assignment count returned at the  
15902 completion of execution of the function. No argument shall be converted, but one shall  
15903 be consumed. If the conversion specification includes an assignment-suppressing wide  
15904 character or a field width, the behavior is undefined.
- 15905 XSI C Equivalent to `lc`.
- 15906 XSI S Equivalent to `ls`.
- 15907 % Matches a single '%' wide character; no conversion or assignment shall occur. The  
15908 complete conversion specification shall be %%.
- 15909 If a conversion specification is invalid, the behavior is undefined.
- 15910 The conversion specifiers A, E, F, G, and X are also valid and shall be equivalent to, respectively,  
15911 a, e, f, g, and x.
- 15912 If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before  
15913 any wide characters matching the current conversion specification (except for %n) have been  
15914 read (other than leading white-space, where permitted), execution of the current conversion  
15915 specification shall terminate with an input failure. Otherwise, unless execution of the current  
15916 conversion specification is terminated with a matching failure, execution of the following  
15917 conversion specification (if any) shall be terminated with an input failure.
- 15918 Reaching the end of the string in *swscanf()* shall be equivalent to encountering end-of-file for  
15919 *fwscanf()*.
- 15920 If conversion terminates on a conflicting input, the offending input shall be left unread in the  
15921 input. Any trailing white space (including <newline>) shall be left unread unless matched by a  
15922 conversion specification. The success of literal matches and suppressed assignments is only  
15923 directly determinable via the %n conversion specification.
- 15924 CX The *fwscanf()* and *wscanf()* functions may mark the *st\_atime* field of the file associated with  
15925 *stream* for update. The *st\_atime* field shall be marked for update by the first successful execution  
15926 of *fgetc()*, *fgetwc()*, *fgets()*, *fgetws()*, *fread()*, *getc()*, *getwc()*, *getchar()*, *getwchar()*, *gets()*, *fscanf()*,  
15927 or *fwscanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.
- 15928 **RETURN VALUE**
- 15929 Upon successful completion, these functions shall return the number of successfully matched  
15930 and assigned input items; this number can be zero in the event of an early matching failure. If  
15931 the input ends before the first matching failure or conversion, EOF shall be returned. If a read  
15932 CX error occurs, the error indicator for the stream is set, EOF shall be returned, and *errno* shall be set  
15933 to indicate the error.
- 15934 **ERRORS**
- 15935 For the conditions under which the *fwscanf()* functions shall fail and may fail, refer to *fgetwc()*.
- 15936 In addition, *fwscanf()* may fail if:
- 15937 XSI [EILSEQ] Input byte sequence does not form a valid character.
- 15938 XSI [EINVAL] There are insufficient arguments.

15939 **EXAMPLES**

15940 The call:

```
15941 int i, n; float x; char name[50];
15942 n = wscanf(L"%d%f%s", &i, &x, name);
```

15943 with the input line:

15944 25 54.32E-1 Hamster

15945 assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string  
15946 "Hamster".

15947 The call:

```
15948 int i; float x; char name[50];
15949 (void) wscanf(L"%2d%f*d %[0123456789]", &i, &x, name);
```

15950 with input:

15951 56789 0123 56a72

15952 assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string "56\0" in *name*. The next call to  
15953 *getchar()* shall return the character 'a'.

15954 **APPLICATION USAGE**

15955 In format strings containing the '%' form of conversion specifications, each argument in the  
15956 argument list is used exactly once.

15957 **RATIONALE**

15958 None.

15959 **FUTURE DIRECTIONS**

15960 None.

15961 **SEE ALSO**

15962 *getwc()*, *fwprintf()*, *setlocale()*, *wctod()*, *wctol()*, *wcstoul()*, *wcrtomb()*, the Base Definitions  
15963 volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <langinfo.h>, <stdio.h>, <wchar.h>

15964 **CHANGE HISTORY**

15965 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
15966 (E).

15967 **Issue 6**

15968 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

15969 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 15970 • The prototypes for *fwscanf()* and *swscanf()* are updated.
- 15971 • The DESCRIPTION is updated.
- 15972 • The hh, ll, j, t, and z length modifiers are added.
- 15973 • The a, A, and F conversion characters are added.

15974 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion  
15975 specification” consistently.

15976 **NAME**

15977        gai\_strerror — address and name information error description

15978 **SYNOPSIS**

15979        #include &lt;netdb.h&gt;

15980        const char \*gai\_strerror(int *ecode*);15981 **DESCRIPTION**15982        The *gai\_strerror()* function shall return a text string describing an error value for the *getaddrinfo()*  
15983        and *getnameinfo()* functions listed in the <netdb.h> header.15984        When the *ecode* argument is one of the following values listed in the <netdb.h> header:

15985           [EAI\_AGAIN]

15986           [EAI\_BADFLAGS]

15987           [EAI\_FAIL]

15988           [EAI\_FAMILY]

15989           [EAI\_MEMORY]

15990           [EAI\_NONAME]

15991           [EAI\_OVERFLOW]

15992           [EAI\_SERVICE]

15993           [EAI\_SOCKTYPE]

15994           [EAI\_SYSTEM]

15995        the function return value shall point to a string describing the error. If the argument is not one  
15996        of those values, the function shall return a pointer to a string whose contents indicate an  
15997        unknown error.15998 **RETURN VALUE**15999        Upon successful completion, *gai\_strerror()* shall return a pointer to an implementation-defined  
16000        string.16001 **ERRORS**

16002        No errors are defined.

16003 **EXAMPLES**

16004        None.

16005 **APPLICATION USAGE**

16006        None.

16007 **RATIONALE**

16008        None.

16009 **FUTURE DIRECTIONS**

16010        None.

16011 **SEE ALSO**16012        *getaddrinfo()*, the Base Definitions volume of IEEE Std 1003.1-2001, <netdb.h>16013 **CHANGE HISTORY**

16014        First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

16015        The Open Group Base Resolution bwg2001-009 is applied, which changes the return type from  
16016        **char \*** to **const char \***. This is for coordination with the IPnG Working Group.16017        IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/22 is applied, adding the  
16018        [EAI\_OVERFLOW] error code. |

16019 **NAME**

16020 gcvt — convert a floating-point number to a string (**LEGACY**)

16021 **SYNOPSIS**

16022 XSI #include <stdlib.h>

16023 char \*gcvt(double value, int ndigit, char \*buf);

16024

16025 **DESCRIPTION**

16026 Refer to *ecvt()*.

16027 **NAME**

16028           getaddrinfo — get address information

16029 **SYNOPSIS**

16030           #include &lt;sys/socket.h&gt;

16031           #include &lt;netdb.h&gt;

16032           int getaddrinfo(const char \*restrict *nodename*,16033                           const char \*restrict *servname*,16034                           const struct addrinfo \*restrict *hints*,16035                           struct addrinfo \*\*restrict *res*);16036 **DESCRIPTION**16037           Refer to *freeaddrinfo()*.

16038 **NAME**

16039           getc — get a byte from a stream

16040 **SYNOPSIS**

16041           #include <stdio.h>

16042           int getc(FILE \*stream);

16043 **DESCRIPTION**

16044 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
16045 conflict between the requirements described here and the ISO C standard is unintentional. This  
16046 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

16047       The *getc()* function shall be equivalent to *fgetc()*, except that if it is implemented as a macro it  
16048 may evaluate *stream* more than once, so the argument should never be an expression with side  
16049 effects.

16050 **RETURN VALUE**

16051       Refer to *fgetc()*.

16052 **ERRORS**

16053       Refer to *fgetc()*.

16054 **EXAMPLES**

16055       None.

16056 **APPLICATION USAGE**

16057       If the integer value returned by *getc()* is stored into a variable of type **char** and then compared  
16058 against the integer constant **EOF**, the comparison may never succeed, because sign-extension of  
16059 a variable of type **char** on widening to integer is implementation-defined.

16060       Since it may be implemented as a macro, *getc()* may treat incorrectly a *stream* argument with  
16061 side effects. In particular, *getc(\*f++)* does not necessarily work as expected. Therefore, use of this  
16062 function should be preceded by "#undef getc" in such situations; *fgetc()* could also be used.

16063 **RATIONALE**

16064       None.

16065 **FUTURE DIRECTIONS**

16066       None.

16067 **SEE ALSO**

16068       *fgetc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

16069 **CHANGE HISTORY**

16070       First released in Issue 1. Derived from Issue 1 of the SVID.

16071 **NAME**

16072        getc\_unlocked, getchar\_unlocked, putc\_unlocked, putchar\_unlocked — stdio with explicit client  
16073        locking

16074 **SYNOPSIS**

```
16075 TSF #include <stdio.h>

16076 int getc_unlocked(FILE *stream);
16077 int getchar_unlocked(void);
16078 int putc_unlocked(int c, FILE *stream);
16079 int putchar_unlocked(int c);
16080
```

16081 **DESCRIPTION**

16082        Versions of the functions *getc()*, *getchar()*, *putc()*, and *putchar()* respectively named  
16083        *getc\_unlocked()*, *getchar\_unlocked()*, *putc\_unlocked()*, and *putchar\_unlocked()* shall be provided  
16084        which are functionally equivalent to the original versions, with the exception that they are not  
16085        required to be implemented in a thread-safe manner. They may only safely be used within a  
16086        scope protected by *flockfile()* (or *ftrylockfile()*) and *funlockfile()*. These functions may safely be  
16087        used in a multi-threaded program if and only if they are called while the invoking thread owns  
16088        the (**FILE\***) object, as is the case after a successful call to the *flockfile()* or *ftrylockfile()* functions.

16089 **RETURN VALUE**

16090        See *getc()*, *getchar()*, *putc()*, and *putchar()*.

16091 **ERRORS**

16092        See *getc()*, *getchar()*, *putc()*, and *putchar()*.

16093 **EXAMPLES**

16094        None.

16095 **APPLICATION USAGE**

16096        Since they may be implemented as macros, *getc\_unlocked()* and *putc\_unlocked()* may treat  
16097        incorrectly a *stream* argument with side effects. In particular, *getc\_unlocked(\*f++)* and  
16098        *putc\_unlocked(\*f++)* do not necessarily work as expected. Therefore, use of these functions in  
16099        such situations should be preceded by the following statement as appropriate:

```
16100 #undef getc_unlocked
16101 #undef putc_unlocked
```

16102 **RATIONALE**

16103        Some I/O functions are typically implemented as macros for performance reasons (for example,  
16104        *putc()* and *getc()*). For safety, they need to be synchronized, but it is often too expensive to  
16105        synchronize on every character. Nevertheless, it was felt that the safety concerns were more  
16106        important; consequently, the *getc()*, *getchar()*, *putc()*, and *putchar()* functions are required to be  
16107        thread-safe. However, unlocked versions are also provided with names that clearly indicate the  
16108        unsafe nature of their operation but can be used to exploit their higher performance. These  
16109        unlocked versions can be safely used only within explicitly locked program regions, using  
16110        exported locking primitives. In particular, a sequence such as:

```
16111 flockfile(fileptr);
16112 putc_unlocked('1', fileptr);
16113 putc_unlocked('\n', fileptr);
16114 fprintf(fileptr, "Line 2\n");
16115 funlockfile(fileptr);
```

16116        is permissible, and results in the text sequence:

16117 1  
16118 Line 2  
16119 being printed without being interspersed with output from other threads.

16120 It would be wrong to have the standard names such as *getc()*, *putc()*, and so on, map to the  
16121 “faster, but unsafe” rather than the “slower, but safe” versions. In either case, you would still  
16122 want to inspect all uses of *getc()*, *putc()*, and so on, by hand when converting existing code.  
16123 Choosing the safe bindings as the default, at least, results in correct code and maintains the  
16124 “atomicity at the function” invariant. To do otherwise would introduce gratuitous  
16125 synchronization errors into converted code. Other routines that modify the *stdio* (**FILE** \*)  
16126 structures or buffers are also safely synchronized.

16127 Note that there is no need for functions of the form *getc\_locked()*, *putc\_locked()*, and so on, since  
16128 this is the functionality of *getc()*, *putc()*, *et al.* It would be inappropriate to use a feature test  
16129 macro to switch a macro definition of *getc()* between *getc\_locked()* and *getc\_unlocked()*, since the  
16130 ISO C standard requires an actual function to exist, a function whose behavior could not be  
16131 changed by the feature test macro. Also, providing both the *xxx\_locked()* and *xxx\_unlocked()*  
16132 forms leads to the confusion of whether the suffix describes the behavior of the function or the  
16133 circumstances under which it should be used.

16134 Three additional routines, *flockfile()*, *ftrylockfile()*, and *funlockfile()* (which may be macros), are  
16135 provided to allow the user to delineate a sequence of I/O statements that are executed  
16136 synchronously.

16137 The *ungetc()* function is infrequently called relative to the other functions/macros so no  
16138 unlocked variation is needed.

16139 **FUTURE DIRECTIONS**  
16140 None.

16141 **SEE ALSO**  
16142 *getc()*, *getchar()*, *putc()*, *putchar()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
16143 **<stdio.h>**

16144 **CHANGE HISTORY**  
16145 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

16146 **Issue 6**  
16147 These functions are marked as part of the Thread-Safe Functions option.

16148 The Open Group Corrigendum U030/2 is applied, adding APPLICATION USAGE describing  
16149 how applications should be written to avoid the case when the functions are implemented as  
16150 macros.

16151 **NAME**

16152            *getchar* — get a byte from a stdin stream

16153 **SYNOPSIS**

16154            #include <stdio.h>

16155            int *getchar*(void);

16156 **DESCRIPTION**

16157 *cx*        The functionality described on this reference page is aligned with the ISO C standard. Any  
16158            conflict between the requirements described here and the ISO C standard is unintentional. This  
16159            volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

16160            The *getchar()* function shall be equivalent to *getc(stdin)*.

16161 **RETURN VALUE**

16162            Refer to *fgetc()*.

16163 **ERRORS**

16164            Refer to *fgetc()*.

16165 **EXAMPLES**

16166            None.

16167 **APPLICATION USAGE**

16168            If the integer value returned by *getchar()* is stored into a variable of type **char** and then  
16169            compared against the integer constant EOF, the comparison may never succeed, because sign-  
16170            extension of a variable of type **char** on widening to integer is implementation-defined.

16171 **RATIONALE**

16172            None.

16173 **FUTURE DIRECTIONS**

16174            None.

16175 **SEE ALSO**

16176            *getc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**stdio.h**>

16177 **CHANGE HISTORY**

16178            First released in Issue 1. Derived from Issue 1 of the SVID.

16179 **NAME**

16180            getchar\_unlocked — stdio with explicit client locking

16181 **SYNOPSIS**

16182 TSF        #include <stdio.h>

16183            int getchar\_unlocked(void);

16184

16185 **DESCRIPTION**

16186            Refer to *getc\_unlocked()*.

16187 **NAME**

16188       getcontext, setcontext — get and set current user context

16189 **SYNOPSIS**

16190 xSI       #include &lt;ucontext.h&gt;

16191       int getcontext(ucontext\_t \*ucp);

16192       int setcontext(const ucontext\_t \*ucp);

16193

16194 **DESCRIPTION**

16195       The *getcontext()* function shall initialize the structure pointed to by *ucp* to the current user  
 16196       context of the calling thread. The **ucontext\_t** type that *ucp* points to defines the user context and  
 16197       includes the contents of the calling thread's machine registers, the signal mask, and the current  
 16198       execution stack.

16199       The *setcontext()* function shall restore the user context pointed to by *ucp*. A successful call to  
 16200       *setcontext()* shall not return; program execution resumes at the point specified by the *ucp*  
 16201       argument passed to *setcontext()*. The *ucp* argument should be created either by a prior call to  
 16202       *getcontext()* or *makecontext()*, or by being passed as an argument to a signal handler. If the *ucp*  
 16203       argument was created with *getcontext()*, program execution continues as if the corresponding  
 16204       call of *getcontext()* had just returned. If the *ucp* argument was created with *makecontext()*,  
 16205       program execution continues with the function passed to *makecontext()*. When that function  
 16206       returns, the thread shall continue as if after a call to *setcontext()* with the *ucp* argument that was  
 16207       input to *makecontext()*. If the *uc\_link* member of the **ucontext\_t** structure pointed to by the *ucp*  
 16208       argument is equal to 0, then this context is the main context, and the thread shall exit when this  
 16209       context returns. The effects of passing a *ucp* argument obtained from any other source are  
 16210       unspecified.

16211 **RETURN VALUE**

16212       Upon successful completion, *setcontext()* shall not return and *getcontext()* shall return 0;  
 16213       otherwise, a value of -1 shall be returned.

16214 **ERRORS**

16215       No errors are defined.

16216 **EXAMPLES**16217       Refer to *makecontext()*.16218 **APPLICATION USAGE**

16219       When a signal handler is executed, the current user context is saved and a new context is  
 16220       created. If the thread leaves the signal handler via *longjmp()*, then it is unspecified whether the  
 16221       context at the time of the corresponding *setjmp()* call is restored and thus whether future calls to  
 16222       *getcontext()* provide an accurate representation of the current context, since the context restored  
 16223       by *longjmp()* does not necessarily contain all the information that *setcontext()* requires. Signal  
 16224       handlers should use *siglongjmp()* or *setcontext()* instead.

16225       Conforming applications should not modify or access the *uc\_mcontext* member of **ucontext\_t**. A  
 16226       conforming application cannot assume that context includes any process-wide static data,  
 16227       possibly including *errno*. Users manipulating contexts should take care to handle these  
 16228       explicitly when required.

16229       Use of contexts to create alternate stacks is not defined by this volume of IEEE Std 1003.1-2001.

16230 **RATIONALE**

16231           None.

16232 **FUTURE DIRECTIONS**

16233           None.

16234 **SEE ALSO**

16235           *bsd\_signal()*, *makecontext()*, *setcontext()*, *setjmp()*, *sigaction()*, *sigaltstack()*, *siglongjmp()*,  
16236           *sigprocmask()*, *sigsetjmp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**ucontext.h**>

16237 **CHANGE HISTORY**

16238           First released in Issue 4, Version 2.

16239 **Issue 5**

16240           Moved from X/OPEN UNIX extension to BASE.

16241           The following sentence was removed from the DESCRIPTION: “If the *ucp* argument was passed  
16242           to a signal handler, program execution continues with the program instruction following the  
16243           instruction interrupted by the signal.”

16244 **NAME**

16245           getcwd — get the pathname of the current working directory

16246 **SYNOPSIS**

16247           #include &lt;unistd.h&gt;

16248           char \*getcwd(char \*buf, size\_t size);

16249 **DESCRIPTION**

16250           The *getcwd()* function shall place an absolute pathname of the current working directory in the  
 16251           array pointed to by *buf*, and return *buf*. The pathname copied to the array shall contain no  
 16252           components that are symbolic links. The *size* argument is the size in bytes of the character array  
 16253           pointed to by the *buf* argument. If *buf* is a null pointer, the behavior of *getcwd()* is unspecified.

16254 **RETURN VALUE**

16255           Upon successful completion, *getcwd()* shall return the *buf* argument. Otherwise, *getcwd()* shall  
 16256           return a null pointer and set *errno* to indicate the error. The contents of the array pointed to by  
 16257           *buf* are then undefined.

16258 **ERRORS**16259           The *getcwd()* function shall fail if:16260           [EINVAL]           The *size* argument is 0.

16261           [ERANGE]           The *size* argument is greater than 0, but is smaller than the length of the  
 16262           pathname +1.

16263           The *getcwd()* function may fail if:

16264           [EACCES]           Read or search permission was denied for a component of the pathname.

16265           [ENOMEM]           Insufficient storage space is available.

16266 **EXAMPLES**16267           **Determining the Absolute Pathname of the Current Working Directory**

16268           The following example returns a pointer to an array that holds the absolute pathname of the  
 16269           current working directory. The pointer is returned in the *ptr* variable, which points to the *buf*  
 16270           array where the pathname is stored.

16271           #include &lt;stdlib.h&gt;

16272           #include &lt;unistd.h&gt;

16273           ...

16274           long size;

16275           char \*buf;

16276           char \*ptr;

16277           size = pathconf(".", \_PC\_PATH\_MAX);

16278           if ((buf = (char \*)malloc((size\_t)size)) != NULL)

16279           ptr = getcwd(buf, (size\_t)size);

16280           ...

16281 **APPLICATION USAGE**

16282           None.

16283 **RATIONALE**

16284 Since the maximum pathname length is arbitrary unless {PATH\_MAX} is defined, an application  
16285 generally cannot supply a *buf* with *size* {{PATH\_MAX}+1}.

16286 Having *getcwd()* take no arguments and instead use the *malloc()* function to produce space for  
16287 the returned argument was considered. The advantage is that *getcwd()* knows how big the  
16288 working directory pathname is and can allocate an appropriate amount of space. But the  
16289 programmer would have to use the *free()* function to free the resulting object, or each use of  
16290 *getcwd()* would further reduce the available memory. Also, *malloc()* and *free()* are used nowhere  
16291 else in this volume of IEEE Std 1003.1-2001. Finally, *getcwd()* is taken from the SVID where it has  
16292 the two arguments used in this volume of IEEE Std 1003.1-2001.

16293 The older function *getwd()* was rejected for use in this context because it had only a buffer  
16294 argument and no *size* argument, and thus had no way to prevent overwriting the buffer, except  
16295 to depend on the programmer to provide a large enough buffer.

16296 On some implementations, if *buf* is a null pointer, *getcwd()* may obtain *size* bytes of memory  
16297 using *malloc()*. In this case, the pointer returned by *getcwd()* may be used as the argument in a  
16298 subsequent call to *free()*. Invoking *getcwd()* with *buf* as a null pointer is not recommended in  
16299 conforming applications.

16300 If a program is operating in a directory where some (grand)parent directory does not permit  
16301 reading, *getcwd()* may fail, as in most implementations it must read the directory to determine  
16302 the name of the file. This can occur if search, but not read, permission is granted in an  
16303 intermediate directory, or if the program is placed in that directory by some more privileged  
16304 process (for example, login). Including the [EACCES] error condition makes the reporting of the  
16305 error consistent and warns the application writer that *getcwd()* can fail for reasons beyond the  
16306 control of the application writer or user. Some implementations can avoid this occurrence (for  
16307 example, by implementing *getcwd()* using *pwd*, where *pwd* is a set-user-root process), thus the  
16308 error was made optional. Since this volume of IEEE Std 1003.1-2001 permits the addition of other  
16309 errors, this would be a common addition and yet one that applications could not be expected to  
16310 deal with without this addition.

16311 **FUTURE DIRECTIONS**

16312 None.

16313 **SEE ALSO**

16314 *malloc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <unistd.h>

16315 **CHANGE HISTORY**

16316 First released in Issue 1. Derived from Issue 1 of the SVID.

16317 **Issue 6**

16318 The following new requirements on POSIX implementations derive from alignment with the  
16319 Single UNIX Specification:

- 16320 • The [ENOMEM] optional error condition is added.

16321 **NAME**

16322 getdate — convert user format date and time

16323 **SYNOPSIS**16324 XSI 

```
#include <time.h>
```

16325 

```
struct tm *getdate(const char *string);
```

16326

16327 **DESCRIPTION**16328 The *getdate()* function shall convert a string representation of a date or time into a broken-down  
16329 time.16330 The external variable or macro *getdate\_err* is used by *getdate()* to return error values.16331 Templates are used to parse and interpret the input string. The templates are contained in a text  
16332 file identified by the environment variable *DATEMSK*. The *DATEMSK* variable should be set to  
16333 indicate the full pathname of the file that contains the templates. The first line in the template  
16334 that matches the input specification is used for interpretation and conversion into the internal  
16335 time format.

16336 The following conversion specifications shall be supported:

16337 %% Equivalent to %.

16338 %a Abbreviated weekday name.

16339 %A Full weekday name.

16340 %b Abbreviated month name.

16341 %B Full month name.

16342 %c Locale's appropriate date and time representation.

16343 %C Century number [00,99]; leading zeros are permitted but not required.

16344 %d Day of month [01,31]; the leading 0 is optional.

16345 %D Date as %m/%d/%y.

16346 %e Equivalent to %d.

16347 %h Abbreviated month name.

16348 %H Hour [00,23].

16349 %I Hour [01,12].

16350 %m Month number [01,12].

16351 %M Minute [00,59].

16352 %n Equivalent to &lt;newline&gt;.

16353 %p Locale's equivalent of either AM or PM.

16354 %r The locale's appropriate representation of time in AM and PM notation. In the POSIX  
16355 locale, this shall be equivalent to %I:%M:%S %p.

16356 %R Time as %H:%M.

16357 %S Seconds [00,60]. The range goes to 60 (rather than stopping at 59) to allow positive leap  
16358 seconds to be expressed. Since leap seconds cannot be predicted by any algorithm, leap  
16359 second data must come from some external source.

|       |    |                                                                                                                                                                                                                                                                                                                                            |
|-------|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 16360 | %t | Equivalent to <tab>.                                                                                                                                                                                                                                                                                                                       |
| 16361 | %T | Time as %H:%M:%S.                                                                                                                                                                                                                                                                                                                          |
| 16362 | %w | Weekday number (Sunday = [0,6]).                                                                                                                                                                                                                                                                                                           |
| 16363 | %x | Locale's appropriate date representation.                                                                                                                                                                                                                                                                                                  |
| 16364 | %X | Locale's appropriate time representation.                                                                                                                                                                                                                                                                                                  |
| 16365 | %y | Year within century. When a century is not otherwise specified, values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.                                                                                                                      |
| 16368 |    | <b>Note:</b> It is expected that in a future version of IEEE Std 1003.1-2001 the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)                                                                                                                           |
| 16369 |    |                                                                                                                                                                                                                                                                                                                                            |
| 16370 |    |                                                                                                                                                                                                                                                                                                                                            |
| 16371 | %Y | Year as "ccyy" (for example, 2001).                                                                                                                                                                                                                                                                                                        |
| 16372 | %Z | Timezone name or no characters if no timezone exists. If the timezone supplied by %Z is not the timezone that <i>getdate()</i> expects, an invalid input specification error shall result. The <i>getdate()</i> function calculates an expected timezone based on information supplied to the function (such as the hour, day, and month). |
| 16373 |    |                                                                                                                                                                                                                                                                                                                                            |
| 16374 |    |                                                                                                                                                                                                                                                                                                                                            |
| 16375 |    |                                                                                                                                                                                                                                                                                                                                            |
| 16376 |    | The match between the template and input specification performed by <i>getdate()</i> shall be case-insensitive.                                                                                                                                                                                                                            |
| 16377 |    |                                                                                                                                                                                                                                                                                                                                            |
| 16378 |    | The month and weekday names can consist of any combination of upper and lowercase letters. The process can request that the input date or time specification be in a specific language by setting the <i>LC_TIME</i> category (see <i>setlocale()</i> ).                                                                                   |
| 16379 |    |                                                                                                                                                                                                                                                                                                                                            |
| 16380 |    |                                                                                                                                                                                                                                                                                                                                            |
| 16381 |    | Leading zeros are not necessary for the descriptors that allow leading zeros. However, at most two digits are allowed for those descriptors, including leading zeros. Extra whitespace in either the template file or in <i>string</i> shall be ignored.                                                                                   |
| 16382 |    |                                                                                                                                                                                                                                                                                                                                            |
| 16383 |    |                                                                                                                                                                                                                                                                                                                                            |
| 16384 |    | The results are undefined if the conversion specifications %c, %x, and %X include unsupported conversion specifications.                                                                                                                                                                                                                   |
| 16385 |    |                                                                                                                                                                                                                                                                                                                                            |
| 16386 |    | The following rules apply for converting the input specification into the internal format:                                                                                                                                                                                                                                                 |
| 16387 |    | • If %Z is being scanned, then <i>getdate()</i> shall initialize the broken-down time to be the current time in the scanned timezone. Otherwise, it shall initialize the broken-down time based on the current local time as if <i>localtime()</i> had been called.                                                                        |
| 16388 |    |                                                                                                                                                                                                                                                                                                                                            |
| 16389 |    |                                                                                                                                                                                                                                                                                                                                            |
| 16390 |    | • If only the weekday is given, the day chosen shall be the day, starting with today and moving into the future, which first matches the named day.                                                                                                                                                                                        |
| 16391 |    |                                                                                                                                                                                                                                                                                                                                            |
| 16392 |    | • If only the month (and no year) is given, the month chosen shall be the month, starting with the current month and moving into the future, which first matches the named month. The first day of the month shall be assumed if no day is given.                                                                                          |
| 16393 |    |                                                                                                                                                                                                                                                                                                                                            |
| 16394 |    |                                                                                                                                                                                                                                                                                                                                            |
| 16395 |    | • If no hour, minute, and second are given, the current hour, minute, and second shall be assumed.                                                                                                                                                                                                                                         |
| 16396 |    |                                                                                                                                                                                                                                                                                                                                            |
| 16397 |    | • If no date is given, the hour chosen shall be the hour, starting with the current hour and moving into the future, which first matches the named hour.                                                                                                                                                                                   |
| 16398 |    |                                                                                                                                                                                                                                                                                                                                            |
| 16399 |    | If a conversion specification in the <i>DATMSK</i> file does not correspond to one of the conversion specifications above, the behavior is unspecified.                                                                                                                                                                                    |
| 16400 |    |                                                                                                                                                                                                                                                                                                                                            |
| 16401 |    | The <i>getdate()</i> function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.                                                                                                                                                                                                    |
| 16402 |    |                                                                                                                                                                                                                                                                                                                                            |

## 16403 RETURN VALUE

16404 Upon successful completion, `getdate()` shall return a pointer to a **struct tm**. Otherwise, it shall  
 16405 return a null pointer and set `getdate_err` to indicate the error.

## 16406 ERRORS

16407 The `getdate()` function shall fail in the following cases, setting `getdate_err` to the value shown in  
 16408 the list below. Any changes to `errno` are unspecified.

- 16409 1. The `DATEMSK` environment variable is null or undefined.
- 16410 2. The template file cannot be opened for reading.
- 16411 3. Failed to get file status information.
- 16412 4. The template file is not a regular file.
- 16413 5. An I/O error is encountered while reading the template file.
- 16414 6. Memory allocation failed (not enough memory available).
- 16415 7. There is no line in the template that matches the input.
- 16416 8. Invalid input specification. For example, February 31; or a time is specified that cannot be  
 16417 represented in a `time_t` (representing the time in seconds since the Epoch).

## 16418 EXAMPLES

- 16419 1. The following example shows the possible contents of a template:

```
16420 %m
16421 %A %B %d, %Y, %H:%M:%S
16422 %A
16423 %B
16424 %m/%d/%y %I %p
16425 %d, %m, %Y %H:%M
16426 at %A the %dst of %B in %Y
16427 run job at %I %p, %B %dnd
16428 %A den %d. %B %Y %H.%M Uhr
```

- 16429 2. The following are examples of valid input specifications for the template in Example 1:

```
16430 getdate("10/1/87 4 PM");
16431 getdate("Friday");
16432 getdate("Friday September 18, 1987, 10:30:30");
16433 getdate("24,9,1986 10:30");
16434 getdate("at monday the 1st of december in 1986");
16435 getdate("run job at 3 PM, december 2nd");
```

16436 If the `LC_TIME` category is set to a German locale that includes *freitag* as a weekday name  
 16437 and *oktober* as a month name, the following would be valid:

```
16438 getdate("freitag den 10. oktober 1986 10.30 Uhr");
```

- 16439 3. The following example shows how local date and time specification can be defined in the  
 16440 template:

16441  
16442  
16443  
16444  
16445  
16446

| Invocation                 | Line in Template |
|----------------------------|------------------|
| getdate("11/27/86")        | %m/%d/%y         |
| getdate("27.11.86")        | %d.%m.%y         |
| getdate("86-11-27")        | %y-%m-%d         |
| getdate("Friday 12:00:00") | %A %H:%M:%S      |

16447  
16448  
16449  
16450

4. The following examples help to illustrate the above rules assuming that the current date is Mon Sep 22 12:19:47 EDT 1986 and the *LC\_TIME* category is set to the default C locale:

16451  
16452  
16453  
16454  
16455  
16456  
16457  
16458  
16459  
16460  
16461  
16462  
16463  
16464

| Input        | Line in Template | Date                         |
|--------------|------------------|------------------------------|
| Mon          | %a               | Mon Sep 22 12:19:47 EDT 1986 |
| Sun          | %a               | Sun Sep 28 12:19:47 EDT 1986 |
| Fri          | %a               | Fri Sep 26 12:19:47 EDT 1986 |
| September    | %B               | Mon Sep 1 12:19:47 EDT 1986  |
| January      | %B               | Thu Jan 1 12:19:47 EST 1987  |
| December     | %B               | Mon Dec 1 12:19:47 EST 1986  |
| Sep Mon      | %b %a            | Mon Sep 1 12:19:47 EDT 1986  |
| Jan Fri      | %b %a            | Fri Jan 2 12:19:47 EST 1987  |
| Dec Mon      | %b %a            | Mon Dec 1 12:19:47 EST 1986  |
| Jan Wed 1989 | %b %a %Y         | Wed Jan 4 12:19:47 EST 1989  |
| Fri 9        | %a %H            | Fri Sep 26 09:00:00 EDT 1986 |
| Feb 10:30    | %b %H:%S         | Sun Feb 1 10:00:30 EST 1987  |
| 10:30        | %H:%M            | Tue Sep 23 10:30:00 EDT 1986 |
| 13:30        | %H:%M            | Mon Sep 22 13:30:00 EDT 1986 |

#### 16465 APPLICATION USAGE

16466 Although historical versions of *getdate()* did not require that **<time.h>** declare the external  
16467 variable *getdate\_err*, this volume of IEEE Std 1003.1-2001 does require it. The standard  
16468 developers encourage applications to remove declarations of *getdate\_err* and instead incorporate  
16469 the declaration by including **<time.h>**.

16470 Applications should use %Y (4-digit years) in preference to %y (2-digit years).

#### 16471 RATIONALE

16472 In standard locales, the conversion specifications %c, %x, and %X do not include unsupported  
16473 conversion specifiers and so the text regarding results being undefined is not a problem in that  
16474 case.

#### 16475 FUTURE DIRECTIONS

16476 None.

#### 16477 SEE ALSO

16478 *ctime()*, *localtime()*, *setlocale()*, *strftime()*, *times()*, the Base Definitions volume of  
16479 IEEE Std 1003.1-2001, **<time.h>**

#### 16480 CHANGE HISTORY

16481 First released in Issue 4, Version 2.

#### 16482 Issue 5

16483 Moved from X/OPEN UNIX extension to BASE.

16484 The last paragraph of the DESCRIPTION is added.

16485 The %C conversion specification is added, and the exact meaning of the %y conversion  
16486 specification is clarified in the DESCRIPTION.

- 16487 A note indicating that this function need not be reentrant is added to the DESCRIPTION.
- 16488 The %R conversion specification is changed to follow historical practice.
- 16489 **Issue 6**
- 16490 The DESCRIPTION is updated to refer to “seconds since the Epoch” rather than “seconds since  
16491 00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time*  
16492 functions.
- 16493 The description of %S is updated so that the valid range is [00,60] rather than [00,61].
- 16494 The DESCRIPTION is updated to refer to conversion specifications instead of field descriptors  
16495 for consistency with other functions.

16496 **NAME**

16497           getegid — get the effective group ID

16498 **SYNOPSIS**

16499           #include <unistd.h>

16500           gid\_t getegid(void);

16501 **DESCRIPTION**

16502           The *getegid()* function shall return the effective group ID of the calling process.

16503 **RETURN VALUE**

16504           The *getegid()* function shall always be successful and no return value is reserved to indicate an error.

16506 **ERRORS**

16507           No errors are defined.

16508 **EXAMPLES**

16509           None.

16510 **APPLICATION USAGE**

16511           None.

16512 **RATIONALE**

16513           None.

16514 **FUTURE DIRECTIONS**

16515           None.

16516 **SEE ALSO**

16517           *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>

16519 **CHANGE HISTORY**

16520           First released in Issue 1. Derived from Issue 1 of the SVID.

16521 **Issue 6**

16522           In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

16523           The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 16525           • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- 16526
- 16527

16528 **NAME**

16529            getenv — get value of an environment variable

16530 **SYNOPSIS**

16531            #include <stdlib.h>

16532            char \*getenv(const char \*name);

16533 **DESCRIPTION**

16534 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
16535 conflict between the requirements described here and the ISO C standard is unintentional. This  
16536 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

16537            The *getenv()* function shall search the environment of the calling process (see the Base  
16538 Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables) for the  
16539 environment variable *name* if it exists and return a pointer to the value of the environment  
16540 variable. If the specified environment variable cannot be found, a null pointer shall be returned.  
16541 The application shall ensure that it does not modify the string pointed to by the *getenv()*  
16542 function.

16543 **CX**        The string pointed to may be overwritten by a subsequent call to *getenv()*, *setenv()*, or *unsetenv()*,  
16544 but shall not be overwritten by a call to any other function in this volume of  
16545 IEEE Std 1003.1-2001.

16546 **CX**        If the application modifies *environ* or the pointers to which it points, the behavior of *getenv()* is  
16547 undefined.

16548            The *getenv()* function need not be reentrant. A function that is not required to be reentrant is not  
16549 required to be thread-safe.

16550 **RETURN VALUE**

16551            Upon successful completion, *getenv()* shall return a pointer to a string containing the *value* for  
16552 the specified *name*. If the specified *name* cannot be found in the environment of the calling  
16553 process, a null pointer shall be returned.

16554            The return value from *getenv()* may point to static data which may be overwritten by  
16555 **CX**        subsequent calls to *getenv()*, *setenv()*, or *unsetenv()*.

16556 **XSI**        On XSI-conformant systems, the return value from *getenv()* may point to static data which may  
16557 also be overwritten by subsequent calls to *putenv()*.

16558 **ERRORS**

16559            No errors are defined.

16560 **EXAMPLES**16561            **Getting the Value of an Environment Variable**

16562            The following example gets the value of the *HOME* environment variable.

```
16563 #include <stdlib.h>
16564 ...
16565 const char *name = "HOME";
16566 char *value;
16567 value = getenv(name);
```

16568 **APPLICATION USAGE**

16569 None.

16570 **RATIONALE**

16571 The *clearenv()* function was considered but rejected. The *putenv()* function has now been  
16572 included for alignment with the Single UNIX Specification.

16573 The *getenv()* function is inherently not reentrant because it returns a value pointing to static  
16574 data.

16575 Conforming applications are required not to modify *environ* directly, but to use only the  
16576 functions described here to manipulate the process environment as an abstract object. Thus, the  
16577 implementation of the environment access functions has complete control over the data  
16578 structure used to represent the environment (subject to the requirement that *environ* be  
16579 maintained as a list of strings with embedded equal signs for applications that wish to scan the  
16580 environment). This constraint allows the implementation to properly manage the memory it  
16581 allocates, either by using allocated storage for all variables (copying them on the first invocation  
16582 of *setenv()* or *unsetenv()*), or keeping track of which strings are currently in allocated space and  
16583 which are not, via a separate table or some other means. This enables the implementation to free  
16584 any allocated space used by strings (and perhaps the pointers to them) stored in *environ* when  
16585 *unsetenv()* is called. A C runtime start-up procedure (that which invokes *main()* and perhaps  
16586 initializes *environ*) can also initialize a flag indicating that none of the environment has yet been  
16587 copied to allocated storage, or that the separate table has not yet been initialized.

16588 In fact, for higher performance of *getenv()*, the implementation could also maintain a separate  
16589 copy of the environment in a data structure that could be searched much more quickly (such as  
16590 an indexed hash table, or a binary tree), and update both it and the linear list at *environ* when  
16591 *setenv()* or *unsetenv()* is invoked.

16592 Performance of *getenv()* can be important for applications which have large numbers of  
16593 environment variables. Typically, applications like this use the environment as a resource  
16594 database of user-configurable parameters. The fact that these variables are in the user's shell  
16595 environment usually means that any other program that uses environment variables (such as *ls*,  
16596 which attempts to use *COLUMNS*), or really almost any utility (*LANG*, *LC\_ALL*, and so on) is  
16597 similarly slowed down by the linear search through the variables.

16598 An implementation that maintains separate data structures, or even one that manages the  
16599 memory it consumes, is not currently required as it was thought it would reduce consensus  
16600 among implementors who do not want to change their historical implementations.

16601 The POSIX Threads Extension states that multi-threaded applications must not modify *environ*  
16602 directly, and that IEEE Std 1003.1-2001 is providing functions which such applications can use in  
16603 the future to manipulate the environment in a thread-safe manner. Thus, moving away from  
16604 application use of *environ* is desirable from that standpoint as well.

16605 **FUTURE DIRECTIONS**

16606 None.

16607 **SEE ALSO**

16608 *exec*, *putenv()*, *setenv()*, *unsetenv()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter  
16609 8, Environment Variables, <**stdlib.h**>

16610 **CHANGE HISTORY**

16611 First released in Issue 1. Derived from Issue 1 of the SVID.

16612 **Issue 5**

16613 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
16614 VALUE section.

16615 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

16616 **Issue 6**

16617 The following changes were made to align with the IEEE P1003.1a draft standard:

16618 • References added to the new *setenv()* and *unsetenv()* functions.

16619 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

16620 **NAME**

16621           geteuid — get the effective user ID

16622 **SYNOPSIS**

16623           #include <unistd.h>

16624           uid\_t geteuid(void);

16625 **DESCRIPTION**

16626           The *geteuid()* function shall return the effective user ID of the calling process.

16627 **RETURN VALUE**

16628           The *geteuid()* function shall always be successful and no return value is reserved to indicate an error.

16630 **ERRORS**

16631           No errors are defined.

16632 **EXAMPLES**

16633           None.

16634 **APPLICATION USAGE**

16635           None.

16636 **RATIONALE**

16637           None.

16638 **FUTURE DIRECTIONS**

16639           None.

16640 **SEE ALSO**

16641           *getegid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>

16643 **CHANGE HISTORY**

16644           First released in Issue 1. Derived from Issue 1 of the SVID.

16645 **Issue 6**

16646           In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

16647           The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 16649           • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
16650           required for conforming implementations of previous POSIX specifications, it was not  
16651           required for UNIX applications.

16652 **NAME**

16653           getgid — get the real group ID

16654 **SYNOPSIS**

16655           #include <unistd.h>

16656           gid\_t getgid(void);

16657 **DESCRIPTION**

16658           The *getgid()* function shall return the real group ID of the calling process.

16659 **RETURN VALUE**

16660           The *getgid()* function shall always be successful and no return value is reserved to indicate an error.

16662 **ERRORS**

16663           No errors are defined.

16664 **EXAMPLES**

16665           None.

16666 **APPLICATION USAGE**

16667           None.

16668 **RATIONALE**

16669           None.

16670 **FUTURE DIRECTIONS**

16671           None.

16672 **SEE ALSO**

16673           *getegid()*, *geteuid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>

16675 **CHANGE HISTORY**

16676           First released in Issue 1. Derived from Issue 1 of the SVID.

16677 **Issue 6**

16678           In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

16679           The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 16681
  - The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- 16682
- 16683

16684 **NAME**

16685           getgrent — get the group database entry

16686 **SYNOPSIS**

16687 xSI       #include <grp.h>

16688           struct group \*getgrent(void);

16689

16690 **DESCRIPTION**

16691           Refer to *endgrent()*.

## 16692 NAME

16693 getgrgid, getgrgid\_r — get group database entry for a group ID

## 16694 SYNOPSIS

16695 #include &lt;grp.h&gt;

16696 struct group \*getgrgid(gid\_t gid);

16697 TSF int getgrgid\_r(gid\_t gid, struct group \*grp, char \*buffer,

16698 size\_t bufsize, struct group \*\*result);

16699

## 16700 DESCRIPTION

16701 The *getgrgid()* function shall search the group database for an entry with a matching *gid*.16702 The *getgrgid()* function need not be reentrant. A function that is not required to be reentrant is  
16703 not required to be thread-safe.16704 TSF The *getgrgid\_r()* function shall update the **group** structure pointed to by *grp* and store a pointer  
16705 to that structure at the location pointed to by *result*. The structure shall contain an entry from  
16706 the group database with a matching *gid*. Storage referenced by the group structure is allocated  
16707 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. The  
16708 maximum size needed for this buffer can be determined with the `{_SC_GETGR_R_SIZE_MAX}`  
16709 *sysconf()* parameter. A NULL pointer shall be returned at the location pointed to by *result* on  
16710 error or if the requested entry is not found.

## 16711 RETURN VALUE

16712 Upon successful completion, *getgrgid()* shall return a pointer to a **struct group** with the structure  
16713 defined in `<grp.h>` with a matching entry if one is found. The *getgrgid()* function shall return a  
16714 null pointer if either the requested entry was not found, or an error occurred. On error, *errno*  
16715 shall be set to indicate the error.16716 The return value may point to a static area which is overwritten by a subsequent call to  
16717 *getgrent()*, *getgrgid()*, or *getgrnam()*.16718 TSF If successful, the *getgrgid\_r()* function shall return zero; otherwise, an error number shall be  
16719 returned to indicate the error.

## 16720 ERRORS

16721 The *getgrgid()* and *getgrgid\_r()* functions may fail if:

16722 [EIO] An I/O error has occurred.

16723 [EINTR] A signal was caught during *getgrgid()*.16724 [EMFILE] `{OPEN_MAX}` file descriptors are currently open in the calling process.

16725 [ENFILE] The maximum allowable number of files is currently open in the system.

16726 TSF The *getgrgid\_r()* function may fail if:16727 TSF [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to  
16728 be referenced by the resulting **group** structure.

16729 **EXAMPLES**16730 **Finding an Entry in the Group Database**

16731 The following example uses *getgrgid()* to search the group database for a group ID that was  
 16732 previously stored in a **stat** structure, then prints out the group name if it is found. If the group is  
 16733 not found, the program prints the numeric value of the group for the entry.

```
16734 #include <sys/types.h>
16735 #include <grp.h>
16736 #include <stdio.h>
16737 ...
16738 struct stat statbuf;
16739 struct group *grp;
16740 ...
16741 if ((grp = getgrgid(statbuf.st_gid)) != NULL)
16742 printf(" %-8.8s", grp->gr_name);
16743 else
16744 printf(" %-8d", statbuf.st_gid);
16745 ...
```

16746 **APPLICATION USAGE**

16747 Applications wishing to check for error situations should set *errno* to 0 before calling *getgrgid()*.  
 16748 If *errno* is set on return, an error occurred.

16749 The *getgrgid\_r()* function is thread-safe and shall return values in a user-supplied buffer instead  
 16750 of possibly using a static data area that may be overwritten by each call.

16751 **RATIONALE**

16752 None.

16753 **FUTURE DIRECTIONS**

16754 None.

16755 **SEE ALSO**

16756 *endgrent()*, *getgrnam()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<grp.h>**,  
 16757 **<limits.h>**, **<sys/types.h>**

16758 **CHANGE HISTORY**

16759 First released in Issue 1. Derived from System V Release 2.0.

16760 **Issue 5**

16761 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
 16762 VALUE section.

16763 The *getgrgid\_r()* function is included for alignment with the POSIX Threads Extension.

16764 A note indicating that the *getgrgid()* function need not be reentrant is added to the  
 16765 DESCRIPTION.

16766 **Issue 6**

16767 The *getgrgid\_r()* function is marked as part of the Thread-Safe Functions option.

16768 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION  
 16769 describing matching the *gid*.

16770 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

16771 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

- 16772 The following new requirements on POSIX implementations derive from alignment with the  
16773 Single UNIX Specification:
- 16774 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
16775 required for conforming implementations of previous POSIX specifications, it was not  
16776 required for UNIX applications.
  - 16777 • In the RETURN VALUE section, the requirement to set *errno* on error is added.
  - 16778 • The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.
- 16779 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
16780 its avoidance of possibly using a static data area.
- 16781 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the  
16782 buffer from *bufsize* characters to bytes.

## 16783 NAME

16784 getgrnam, getgrnam\_r — search group database for a name

## 16785 SYNOPSIS

16786 #include <grp.h>

16787 struct group \*getgrnam(const char \*name);

16788 TSF int getgrnam\_r(const char \*name, struct group \*grp, char \*buffer,

16789 size\_t bufsize, struct group \*\*result);

16790

## 16791 DESCRIPTION

16792 The *getgrnam()* function shall search the group database for an entry with a matching *name*.

16793 The *getgrnam()* function need not be reentrant. A function that is not required to be reentrant is  
16794 not required to be thread-safe.

16795 TSF The *getgrnam\_r()* function shall update the **group** structure pointed to by *grp* and store a pointer  
16796 to that structure at the location pointed to by *result*. The structure shall contain an entry from  
16797 the group database with a matching *gid* or *name*. Storage referenced by the **group** structure is  
16798 allocated from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. The  
16799 maximum size needed for this buffer can be determined with the `{_SC_GETGR_R_SIZE_MAX}`  
16800 *sysconf()* parameter. A NULL pointer is returned at the location pointed to by *result* on error or if  
16801 the requested entry is not found.

## 16802 RETURN VALUE

16803 The *getgrnam()* function shall return a pointer to a **struct group** with the structure defined in  
16804 `<grp.h>` with a matching entry if one is found. The *getgrnam()* function shall return a null  
16805 pointer if either the requested entry was not found, or an error occurred. On error, *errno* shall be  
16806 set to indicate the error.

16807 The return value may point to a static area which is overwritten by a subsequent call to  
16808 *getgrent()*, *getgrgid()*, or *getgrnam()*.

16809 TSF If successful, the *getgrnam\_r()* function shall return zero; otherwise, an error number shall be  
16810 returned to indicate the error.

## 16811 ERRORS

16812 The *getgrnam()* and *getgrnam\_r()* functions may fail if:

16813 [EIO] An I/O error has occurred.

16814 [EINTR] A signal was caught during *getgrnam()*.

16815 [EMFILE] `{OPEN_MAX}` file descriptors are currently open in the calling process.

16816 [ENFILE] The maximum allowable number of files is currently open in the system.

16817 The *getgrnam\_r()* function may fail if:

16818 TSF [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to  
16819 be referenced by the resulting **group** structure.

16820 **EXAMPLES**

16821 None.

16822 **APPLICATION USAGE**

16823 Applications wishing to check for error situations should set *errno* to 0 before calling *getgrnam()*.  
 16824 If *errno* is set on return, an error occurred.

16825 The *getgrnam\_r()* function is thread-safe and shall return values in a user-supplied buffer instead  
 16826 of possibly using a static data area that may be overwritten by each call.

16827 **RATIONALE**

16828 None.

16829 **FUTURE DIRECTIONS**

16830 None.

16831 **SEE ALSO**

16832 *endgrent()*, *getgrgid()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<grp.h>**, **<limits.h>**,  
 16833 **<sys/types.h>**

16834 **CHANGE HISTORY**

16835 First released in Issue 1. Derived from System V Release 2.0.

16836 **Issue 5**

16837 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
 16838 VALUE section.

16839 The *getgrnam\_r()* function is included for alignment with the POSIX Threads Extension.

16840 A note indicating that the *getgrnam()* function need not be reentrant is added to the  
 16841 DESCRIPTION.

16842 **Issue 6**16843 The *getgrnam\_r()* function is marked as part of the Thread-Safe Functions option.

16844 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

16845 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

16846 The following new requirements on POSIX implementations derive from alignment with the  
 16847 Single UNIX Specification:

16848 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was  
 16849 required for conforming implementations of previous POSIX specifications, it was not  
 16850 required for UNIX applications.

16851 • In the RETURN VALUE section, the requirement to set *errno* on error is added.

16852 • The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

16853 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
 16854 its avoidance of possibly using a static data area.

16855 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the  
 16856 buffer from *bufsize* characters to bytes.

16857 **NAME**

16858           getgroups — get supplementary group IDs

16859 **SYNOPSIS**

16860           #include &lt;unistd.h&gt;

16861           int getgroups(int *gidsetsize*, gid\_t *grouplist*[]);16862 **DESCRIPTION**

16863           The *getgroups()* function shall fill in the array *grouplist* with the current supplementary group  
 16864           IDs of the calling process. It is implementation-defined whether *getgroups()* also returns the  
 16865           effective group ID in the *grouplist* array.

16866           The *gidsetsize* argument specifies the number of elements in the array *grouplist*. The actual  
 16867           number of group IDs stored in the array shall be returned. The values of array entries with  
 16868           indices greater than or equal to the value returned are undefined.

16869           If *gidsetsize* is 0, *getgroups()* shall return the number of group IDs that it would otherwise return  
 16870           without modifying the array pointed to by *grouplist*.

16871           If the effective group ID of the process is returned with the supplementary group IDs, the value  
 16872           returned shall always be greater than or equal to one and less than or equal to the value of  
 16873           {NGROUPS\_MAX}+1.

16874 **RETURN VALUE**

16875           Upon successful completion, the number of supplementary group IDs shall be returned. A  
 16876           return value of -1 indicates failure and *errno* shall be set to indicate the error.

16877 **ERRORS**16878           The *getgroups()* function shall fail if:

16879           [EINVAL]           The *gidsetsize* argument is non-zero and less than the number of group IDs  
 16880           that would have been returned.

16881 **EXAMPLES**16882           **Getting the Supplementary Group IDs of the Calling Process**

16883           The following example places the current supplementary group IDs of the calling process into  
 16884           the *group* array.

```
16885 #include <sys/types.h>
16886 #include <unistd.h>
16887 ...
16888 gid_t *group;
16889 int nogroups;
16890 long ngroups_max;

16891 ngroups_max = sysconf(_SC_NGROUPS_MAX) + 1;
16892 group = (gid_t *)malloc(ngroups_max * sizeof(gid_t));

16893 ngroups = getgroups(ngroups_max, group);
```

16894 **APPLICATION USAGE**

16895           None.

16896 **RATIONALE**

16897           The related function *setgroups()* is a privileged operation and therefore is not covered by this  
 16898           volume of IEEE Std 1003.1-2001.

16899 As implied by the definition of supplementary groups, the effective group ID may appear in the  
16900 array returned by *getgroups()* or it may be returned only by *getegid()*. Duplication may exist, but  
16901 the application needs to call *getegid()* to be sure of getting all of the information. Various  
16902 implementation variations and administrative sequences cause the set of groups appearing in  
16903 the result of *getgroups()* to vary in order and as to whether the effective group ID is included,  
16904 even when the set of groups is the same (in the mathematical sense of “set”). (The history of a  
16905 process and its parents could affect the details of the result.)

16906 Application writers should note that {NGROUPS\_MAX} is not necessarily a constant on all  
16907 implementations.

#### 16908 **FUTURE DIRECTIONS**

16909 None.

#### 16910 **SEE ALSO**

16911 *getegid()*, *setgid()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>,  
16912 <unistd.h>

#### 16913 **CHANGE HISTORY**

16914 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

#### 16915 **Issue 5**

16916 Normative text previously in the APPLICATION USAGE section is moved to the  
16917 DESCRIPTION.

#### 16918 **Issue 6**

16919 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

16920 The following new requirements on POSIX implementations derive from alignment with the  
16921 Single UNIX Specification:

- 16922 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
16923 required for conforming implementations of previous POSIX specifications, it was not  
16924 required for UNIX applications.
- 16925 • A return value of 0 is not permitted, because {NGROUPS\_MAX} cannot be 0. This is a FIPS  
16926 requirement.

16927 The following changes were made to align with the IEEE P1003.1a draft standard:

- 16928 • An explanation is added that the effective group ID may be included in the supplementary  
16929 group list.

## 16930 NAME

16931 gethostbyaddr, gethostbyname — network host database functions

## 16932 SYNOPSIS

16933 #include <netdb.h>

```
16934 OB struct hostent *gethostbyaddr(const void *addr, socklen_t len,
16935 int type);
```

```
16936 struct hostent *gethostbyname(const char *name);
```

16937

## 16938 DESCRIPTION

16939 These functions shall retrieve information about hosts. This information is considered to be  
 16940 stored in a database that can be accessed sequentially or randomly. Implementation of this  
 16941 database is unspecified.

16942 **Note:** In many cases it is implemented by the Domain Name System, as documented in RFC 1034,  
 16943 RFC 1035, and RFC 1886.

16944 Entries shall be returned in **hostent** structures.

16945 The *gethostbyaddr()* function shall return an entry containing addresses of address family *type* for  
 16946 the host with address *addr*. The *len* argument contains the length of the address pointed to by  
 16947 *addr*. The *gethostbyaddr()* function need not be reentrant. A function that is not required to be  
 16948 reentrant is not required to be thread-safe.

16949 The *gethostbyname()* function shall return an entry containing addresses of address family  
 16950 AF\_INET for the host with name *name*. The *gethostbyname()* function need not be reentrant. A  
 16951 function that is not required to be reentrant is not required to be thread-safe.

16952 The *addr* argument of *gethostbyaddr()* shall be an **in\_addr** structure when *type* is AF\_INET. It  
 16953 contains a binary format (that is, not null-terminated) address in network byte order. The  
 16954 *gethostbyaddr()* function is not guaranteed to return addresses of address families other than  
 16955 AF\_INET, even when such addresses exist in the database.

16956 If *gethostbyaddr()* returns successfully, then the *h\_addrtype* field in the result shall be the same as  
 16957 the *type* argument that was passed to the function, and the *h\_addr\_list* field shall list a single  
 16958 address that is a copy of the *addr* argument that was passed to the function.

16959 The *name* argument of *gethostbyname()* shall be a node name; the behavior of *gethostbyname()*  
 16960 when passed a numeric address string is unspecified. For IPv4, a numeric address string shall be  
 16961 in the dotted-decimal notation described in *inet\_addr()*.

16962 If *name* is not a numeric address string and is an alias for a valid host name, then *gethostbyname()*  
 16963 shall return information about the host name to which the alias refers, and *name* shall be  
 16964 included in the list of aliases returned.

## 16965 RETURN VALUE

16966 Upon successful completion, these functions shall return a pointer to a **hostent** structure if the  
 16967 requested entry was found, and a null pointer if the end of the database was reached or the  
 16968 requested entry was not found.

16969 Upon unsuccessful completion, *gethostbyaddr()* and *gethostbyname()* shall set *h\_errno* to indicate  
 16970 the error.

## 16971 ERRORS

16972 These functions shall fail in the following cases. The *gethostbyaddr()* and *gethostbyname()*  
 16973 functions shall set *h\_errno* to the value shown in the list below. Any changes to *errno* are  
 16974 unspecified.

- 16975 [HOST\_NOT\_FOUND]  
16976 No such host is known.
- 16977 [NO\_DATA] The server recognized the request and the name, but no address is available.  
16978 Another type of request to the name server for the domain might return an  
16979 answer.
- 16980 [NO\_RECOVERY]  
16981 An unexpected server failure occurred which cannot be recovered.
- 16982 [TRY\_AGAIN] A temporary and possibly transient error occurred, such as a failure of a  
16983 server to respond.
- 16984 **EXAMPLES**  
16985 None.
- 16986 **APPLICATION USAGE**  
16987 The *gethostbyaddr()* and *gethostbyname()* functions may return pointers to static data, which may  
16988 be overwritten by subsequent calls to any of these functions.
- 16989 The *getaddrinfo()* and *getnameinfo()* functions are preferred over the *gethostbyaddr()* and  
16990 *gethostbyname()* functions.
- 16991 **RATIONALE**  
16992 None.
- 16993 **FUTURE DIRECTIONS**  
16994 The *gethostbyaddr()* and *gethostbyname()* functions may be withdrawn in a future version.
- 16995 **SEE ALSO**  
16996 *endhostent()*, *endservent()*, *gai\_strerror()*, *getaddrinfo()*, *h\_errno*, *inet\_addr()*, the Base Definitions  
16997 volume of IEEE Std 1003.1-2001, <**netdb.h**>
- 16998 **CHANGE HISTORY**  
16999 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

17000 **NAME**

17001           gethostent — network host database functions

17002 **SYNOPSIS**

17003           #include <netdb.h>

17004           struct hostent \*gethostent(void);

17005 **DESCRIPTION**

17006           Refer to *endhostent()*.

17007 **NAME**

17008           gethostid — get an identifier for the current host

17009 **SYNOPSIS**

17010 XSI       #include <unistd.h>

17011           long gethostid(void);

17012

17013 **DESCRIPTION**

17014           The *gethostid()* function shall retrieve a 32-bit identifier for the current host.

17015 **RETURN VALUE**

17016           Upon successful completion, *gethostid()* shall return an identifier for the current host.

17017 **ERRORS**

17018           No errors are defined.

17019 **EXAMPLES**

17020           None.

17021 **APPLICATION USAGE**

17022           This volume of IEEE Std 1003.1-2001 does not define the domain in which the return value is  
17023           unique.

17024 **RATIONALE**

17025           None.

17026 **FUTURE DIRECTIONS**

17027           None.

17028 **SEE ALSO**

17029           *random()*, the Base Definitions volume of IEEE Std 1003.1-2001, <unistd.h>

17030 **CHANGE HISTORY**

17031           First released in Issue 4, Version 2.

17032 **Issue 5**

17033           Moved from X/OPEN UNIX extension to BASE.

17034 **NAME**

17035         gethostname — get name of current host

17036 **SYNOPSIS**

17037         #include <unistd.h>

17038         int gethostname(char \*name, size\_t namelen);

17039 **DESCRIPTION**

17040         The *gethostname()* function shall return the standard host name for the current machine. The  
17041         *namelen* argument shall specify the size of the array pointed to by the *name* argument. The  
17042         returned name shall be null-terminated, except that if *namelen* is an insufficient length to hold  
17043         the host name, then the returned name shall be truncated and it is unspecified whether the  
17044         returned name is null-terminated.

17045         Host names are limited to {HOST\_NAME\_MAX} bytes.

17046 **RETURN VALUE**

17047         Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned.

17048 **ERRORS**

17049         No errors are defined.

17050 **EXAMPLES**

17051         None.

17052 **APPLICATION USAGE**

17053         None.

17054 **RATIONALE**

17055         None.

17056 **FUTURE DIRECTIONS**

17057         None.

17058 **SEE ALSO**

17059         *gethostid()*, *uname()*, the Base Definitions volume of IEEE Std 1003.1-2001, <unistd.h>

17060 **CHANGE HISTORY**

17061         First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

17062         The Open Group Base Resolution bwg2001-008 is applied, changing the *namelen* parameter from  
17063         **socklen\_t** to **size\_t**.

17064 **NAME**

17065       getitimer, setitimer — get and set value of interval timer

17066 **SYNOPSIS**

17067 XSI       #include &lt;sys/time.h&gt;

17068       int getitimer(int *which*, struct itimerval \**value*);17069       int setitimer(int *which*, const struct itimerval \*restrict *value*,17070           struct itimerval \*restrict *ovalue*);

17071

17072 **DESCRIPTION**

17073       The *getitimer()* function shall store the current value of the timer specified by *which* into the  
 17074       structure pointed to by *value*. The *setitimer()* function shall set the timer specified by *which* to  
 17075       the value specified in the structure pointed to by *value*, and if *ovalue* is not a null pointer, store  
 17076       the previous value of the timer in the structure pointed to by *ovalue*.

17077       A timer value is defined by the **itimerval** structure, specified in <sys/time.h>. If *it\_value* is non-  
 17078       zero, it shall indicate the time to the next timer expiration. If *it\_interval* is non-zero, it shall  
 17079       specify a value to be used in reloading *it\_value* when the timer expires. Setting *it\_value* to 0 shall  
 17080       disable a timer, regardless of the value of *it\_interval*. Setting *it\_interval* to 0 shall disable a timer  
 17081       after its next expiration (assuming *it\_value* is non-zero).

17082       Implementations may place limitations on the granularity of timer values. For each interval  
 17083       timer, if the requested timer value requires a finer granularity than the implementation supports,  
 17084       the actual timer value shall be rounded up to the next supported value.

17085       An XSI-conforming implementation provides each process with at least three interval timers,  
 17086       which are indicated by the *which* argument:

17087       **ITIMER\_REAL**       Decrements in real time. A SIGALRM signal is delivered when this timer  
 17088       expires.

17089       **ITIMER\_VIRTUAL**   Decrements in process virtual time. It runs only when the process is  
 17090       executing. A SIGVTALRM signal is delivered when it expires.

17091       **ITIMER\_PROF**       Decrements both in process virtual time and when the system is running  
 17092       on behalf of the process. It is designed to be used by interpreters in  
 17093       statistically profiling the execution of interpreted programs. Each time the  
 17094       **ITIMER\_PROF** timer expires, the SIGPROF signal is delivered.

17095       The interaction between *setitimer()* and any of *alarm()*, *sleep()*, or *usleep()* is unspecified.

17096 **RETURN VALUE**

17097       Upon successful completion, *getitimer()* or *setitimer()* shall return 0; otherwise, -1 shall be  
 17098       returned and *errno* set to indicate the error.

17099 **ERRORS**

17100       The *setitimer()* function shall fail if:

17101       [EINVAL]       The *value* argument is not in canonical form. (In canonical form, the number of  
 17102       microseconds is a non-negative integer less than 1 000 000 and the number of  
 17103       seconds is a non-negative integer.)

17104       The *getitimer()* and *setitimer()* functions may fail if:

17105       [EINVAL]       The *which* argument is not recognized.

17106 **EXAMPLES**

17107           None.

17108 **APPLICATION USAGE**

17109           None.

17110 **RATIONALE**

17111           None.

17112 **FUTURE DIRECTIONS**

17113           None.

17114 **SEE ALSO**

17115           *alarm()*, *sleep()*, *timer\_getoverrun()*, *ualarm()*, *usleep()*, the Base Definitions volume of  
17116           IEEE Std 1003.1-2001, <**signal.h**>, <**sys/time.h**>

17117 **CHANGE HISTORY**

17118           First released in Issue 4, Version 2.

17119 **Issue 5**

17120           Moved from X/OPEN UNIX extension to BASE.

17121 **Issue 6**

17122           The **restrict** keyword is added to the *setitimer()* prototype for alignment with the  
17123           ISO/IEC 9899:1999 standard.

17124 **NAME**

17125 getlogin, getlogin\_r — get login name

17126 **SYNOPSIS**

17127 #include &lt;unistd.h&gt;

17128 char \*getlogin(void);

17129 TSF int getlogin\_r(char \*name, size\_t namesize);

17130

17131 **DESCRIPTION**

17132 The *getlogin()* function shall return a pointer to a string containing the user name associated by  
 17133 the login activity with the controlling terminal of the current process. If *getlogin()* returns a non-  
 17134 null pointer, then that pointer points to the name that the user logged in under, even if there are  
 17135 several login names with the same user ID.

17136 The *getlogin()* function need not be reentrant. A function that is not required to be reentrant is  
 17137 not required to be thread-safe.

17138 TSF The *getlogin\_r()* function shall put the name associated by the login activity with the controlling  
 17139 terminal of the current process in the character array pointed to by *name*. The array is *namesize*  
 17140 characters long and should have space for the name and the terminating null character. The  
 17141 maximum size of the login name is {LOGIN\_NAME\_MAX}.

17142 If *getlogin\_r()* is successful, *name* points to the name the user used at login, even if there are  
 17143 several login names with the same user ID.

17144 **RETURN VALUE**

17145 Upon successful completion, *getlogin()* shall return a pointer to the login name or a null pointer  
 17146 if the user's login name cannot be found. Otherwise, it shall return a null pointer and set *errno* to  
 17147 indicate the error.

17148 The return value from *getlogin()* may point to static data whose content is overwritten by each  
 17149 call.

17150 TSF If successful, the *getlogin\_r()* function shall return zero; otherwise, an error number shall be  
 17151 returned to indicate the error.

17152 **ERRORS**17153 The *getlogin()* and *getlogin\_r()* functions may fail if:

17154 [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

17155 [ENFILE] The maximum allowable number of files is currently open in the system.

17156 [ENXIO] The calling process has no controlling terminal.

17157 The *getlogin\_r()* function may fail if:

17158 TSF [ERANGE] The value of *namesize* is smaller than the length of the string to be returned  
 17159 including the terminating null character.

## 17160 EXAMPLES

17161 **Getting the User Login Name**

17162 The following example calls the *getlogin()* function to obtain the name of the user associated  
 17163 with the calling process, and passes this information to the *getpwnam()* function to get the  
 17164 associated user database information.

```

17165 #include <unistd.h>
17166 #include <sys/types.h>
17167 #include <pwd.h>
17168 #include <stdio.h>
17169 ...
17170 char *lgn;
17171 struct passwd *pw;
17172 ...
17173 if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
17174 fprintf(stderr, "Get of user information failed.\n"); exit(1);
17175 }
```

17176 **APPLICATION USAGE**

17177 Three names associated with the current process can be determined: *getpwuid(geteuid())* shall  
 17178 return the name associated with the effective user ID of the process; *getlogin()* shall return the  
 17179 name associated with the current login activity; and *getpwuid(getuid())* shall return the name  
 17180 associated with the real user ID of the process.

17181 The *getlogin\_r()* function is thread-safe and returns values in a user-supplied buffer instead of  
 17182 possibly using a static data area that may be overwritten by each call.

17183 **RATIONALE**

17184 The *getlogin()* function returns a pointer to the user's login name. The same user ID may be  
 17185 shared by several login names. If it is desired to get the user database entry that is used during  
 17186 login, the result of *getlogin()* should be used to provide the argument to the *getpwnam()*  
 17187 function. (This might be used to determine the user's login shell, particularly where a single user  
 17188 has multiple login shells with distinct login names, but the same user ID.)

17189 The information provided by the *cuserid()* function, which was originally defined in the  
 17190 POSIX.1-1988 standard and subsequently removed, can be obtained by the following:

```
17191 getpwuid(geteuid())
```

17192 while the information provided by historical implementations of *cuserid()* can be obtained by:

```
17193 getpwuid(getuid())
```

17194 The thread-safe version of this function places the user name in a user-supplied buffer and  
 17195 returns a non-zero value if it fails. The non-thread-safe version may return the name in a static  
 17196 data area that may be overwritten by each call.

17197 **FUTURE DIRECTIONS**

17198 None.

17199 **SEE ALSO**

17200 *getpwnam()*, *getpwuid()*, *geteuid()*, *getuid()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
 17201 <limits.h>, <unistd.h>

17202 **CHANGE HISTORY**

17203 First released in Issue 1. Derived from System V Release 2.0.

17204 **Issue 5**

17205 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
17206 VALUE section.

17207 The *getlogin\_r()* function is included for alignment with the POSIX Threads Extension.

17208 A note indicating that the *getlogin()* function need not be reentrant is added to the  
17209 DESCRIPTION.

17210 **Issue 6**

17211 The *getlogin\_r()* function is marked as part of the Thread-Safe Functions option.

17212 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

17213 The following new requirements on POSIX implementations derive from alignment with the  
17214 Single UNIX Specification:

17215 • In the RETURN VALUE section, the requirement to set *errno* on error is added.

17216 • The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

17217 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
17218 its avoidance of possibly using a static data area.

## 17219 NAME

17220 getmsg, getpmsg — receive next message from a STREAMS file (STREAMS)

## 17221 SYNOPSIS

17222 XSR #include &lt;stropts.h&gt;

```

17223 int getmsg(int fildes, struct strbuf *restrict ctlptr,
17224 struct strbuf *restrict dataptr, int *restrict flagsp);
17225 int getpmsg(int fildes, struct strbuf *restrict ctlptr,
17226 struct strbuf *restrict dataptr, int *restrict bandp,
17227 int *restrict flagsp);
17228

```

## 17229 DESCRIPTION

17230 The *getmsg()* function shall retrieve the contents of a message located at the head of the  
 17231 STREAM head read queue associated with a STREAMS file and place the contents into one or  
 17232 more buffers. The message contains either a data part, a control part, or both. The data and  
 17233 control parts of the message shall be placed into separate buffers, as described below. The  
 17234 semantics of each part are defined by the originator of the message.

17235 The *getpmsg()* function shall be equivalent to *getmsg()*, except that it provides finer control over  
 17236 the priority of the messages received. Except where noted, all requirements on *getmsg()* also  
 17237 pertain to *getpmsg()*.

17238 The *fildes* argument specifies a file descriptor referencing a STREAMS-based file.

17239 The *ctlptr* and *dataptr* arguments each point to a **strbuf** structure, in which the *buf* member points  
 17240 to a buffer in which the data or control information is to be placed, and the *maxlen* member  
 17241 indicates the maximum number of bytes this buffer can hold. On return, the *len* member shall  
 17242 contain the number of bytes of data or control information actually received. The *len* member  
 17243 shall be set to 0 if there is a zero-length control or data part and *len* shall be set to -1 if no data or  
 17244 control information is present in the message.

17245 When *getmsg()* is called, *flagsp* should point to an integer that indicates the type of message the  
 17246 process is able to receive. This is described further below.

17247 The *ctlptr* argument is used to hold the control part of the message, and *dataptr* is used to hold  
 17248 the data part of the message. If *ctlptr* (or *dataptr*) is a null pointer or the *maxlen* member is -1, the  
 17249 control (or data) part of the message shall not be processed and shall be left on the STREAM  
 17250 head read queue, and if the *ctlptr* (or *dataptr*) is not a null pointer, *len* shall be set to -1. If the  
 17251 *maxlen* member is set to 0 and there is a zero-length control (or data) part, that zero-length part  
 17252 shall be removed from the read queue and *len* shall be set to 0. If the *maxlen* member is set to 0  
 17253 and there are more than 0 bytes of control (or data) information, that information shall be left on  
 17254 the read queue and *len* shall be set to 0. If the *maxlen* member in *ctlptr* (or *dataptr*) is less than the  
 17255 control (or data) part of the message, *maxlen* bytes shall be retrieved. In this case, the remainder  
 17256 of the message shall be left on the STREAM head read queue and a non-zero return value shall  
 17257 be provided.

17258 By default, *getmsg()* shall process the first available message on the STREAM head read queue.  
 17259 However, a process may choose to retrieve only high-priority messages by setting the integer  
 17260 pointed to by *flagsp* to RS\_HIPRI. In this case, *getmsg()* shall only process the next message if it is  
 17261 a high-priority message. When the integer pointed to by *flagsp* is 0, any available message shall  
 17262 be retrieved. In this case, on return, the integer pointed to by *flagsp* shall be set to RS\_HIPRI if a  
 17263 high-priority message was retrieved, or 0 otherwise.

17264 For *getpmsg()*, the flags are different. The *flagsp* argument points to a bitmask with the following  
 17265 mutually-exclusive flags defined: MSG\_HIPRI, MSG\_BAND, and MSG\_ANY. Like *getmsg()*,

17266 *getpmsg()* shall process the first available message on the STREAM head read queue. A process  
 17267 may choose to retrieve only high-priority messages by setting the integer pointed to by *flagsp* to  
 17268 MSG\_HIPRI and the integer pointed to by *bandp* to 0. In this case, *getpmsg()* shall only process  
 17269 the next message if it is a high-priority message. In a similar manner, a process may choose to  
 17270 retrieve a message from a particular priority band by setting the integer pointed to by *flagsp* to  
 17271 MSG\_BAND and the integer pointed to by *bandp* to the priority band of interest. In this case,  
 17272 *getpmsg()* shall only process the next message if it is in a priority band equal to, or greater than,  
 17273 the integer pointed to by *bandp*, or if it is a high-priority message. If a process wants to get the  
 17274 first message off the queue, the integer pointed to by *flagsp* should be set to MSG\_ANY and the  
 17275 integer pointed to by *bandp* should be set to 0. On return, if the message retrieved was a high-  
 17276 priority message, the integer pointed to by *flagsp* shall be set to MSG\_HIPRI and the integer  
 17277 pointed to by *bandp* shall be set to 0. Otherwise, the integer pointed to by *flagsp* shall be set to  
 17278 MSG\_BAND and the integer pointed to by *bandp* shall be set to the priority band of the message.

17279 If O\_NONBLOCK is not set, *getmsg()* and *getpmsg()* shall block until a message of the type  
 17280 specified by *flagsp* is available at the front of the STREAM head read queue. If O\_NONBLOCK is  
 17281 set and a message of the specified type is not present at the front of the read queue, *getmsg()* and  
 17282 *getpmsg()* shall fail and set *errno* to [EAGAIN].

17283 If a hangup occurs on the STREAM from which messages are retrieved, *getmsg()* and *getpmsg()*  
 17284 shall continue to operate normally, as described above, until the STREAM head read queue is  
 17285 empty. Thereafter, they shall return 0 in the *len* members of *ctlptr* and *dataptr*.

#### 17286 RETURN VALUE

17287 Upon successful completion, *getmsg()* and *getpmsg()* shall return a non-negative value. A value  
 17288 of 0 indicates that a full message was read successfully. A return value of MORECTL indicates  
 17289 that more control information is waiting for retrieval. A return value of MOREDATA indicates  
 17290 that more data is waiting for retrieval. A return value of the bitwise-logical OR of MORECTL  
 17291 and MOREDATA indicates that both types of information remain. Subsequent *getmsg()* and  
 17292 *getpmsg()* calls shall retrieve the remainder of the message. However, if a message of higher  
 17293 priority has come in on the STREAM head read queue, the next call to *getmsg()* or *getpmsg()*  
 17294 shall retrieve that higher-priority message before retrieving the remainder of the previous  
 17295 message.

17296 If the high priority control part of the message is consumed, the message shall be placed back on  
 17297 the queue as a normal message of band 0. Subsequent *getmsg()* and *getpmsg()* calls shall retrieve  
 17298 the remainder of the message. If, however, a priority message arrives or already exists on the  
 17299 STREAM head, the subsequent call to *getmsg()* or *getpmsg()* shall retrieve the higher-priority  
 17300 message before retrieving the remainder of the message that was put back.

17301 Upon failure, *getmsg()* and *getpmsg()* shall return -1 and set *errno* to indicate the error.

#### 17302 ERRORS

17303 The *getmsg()* and *getpmsg()* functions shall fail if:

- |       |           |                                                                                                                                                                              |
|-------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 17304 | [EAGAIN]  | The O_NONBLOCK flag is set and no messages are available.                                                                                                                    |
| 17305 | [EBADF]   | The <i>fildev</i> argument is not a valid file descriptor open for reading.                                                                                                  |
| 17306 | [EBADMSG] | The queued message to be read is not valid for <i>getmsg()</i> or <i>getpmsg()</i> or a pending file descriptor is at the STREAM head.                                       |
| 17307 |           |                                                                                                                                                                              |
| 17308 | [EINTR]   | A signal was caught during <i>getmsg()</i> or <i>getpmsg()</i> .                                                                                                             |
| 17309 | [EINVAL]  | An illegal value was specified by <i>flagsp</i> , or the STREAM or multiplexer referenced by <i>fildev</i> is linked (directly or indirectly) downstream from a multiplexer. |
| 17310 |           |                                                                                                                                                                              |
| 17311 |           |                                                                                                                                                                              |

17312 [ENOSTR] A STREAM is not associated with *files*.

17313 In addition, *getmsg()* and *getpmsg()* shall fail if the STREAM head had processed an  
17314 asynchronous error before the call. In this case, the value of *errno* does not reflect the result of  
17315 *getmsg()* or *getpmsg()* but reflects the prior error.

#### 17316 EXAMPLES

##### 17317 Getting Any Message

17318 In the following example, the value of *fd* is assumed to refer to an open STREAMS file. The call  
17319 to *getmsg()* retrieves any available message on the associated STREAM-head read queue,  
17320 returning control and data information to the buffers pointed to by *ctrlbuf* and *databuf*,  
17321 respectively.

```
17322 #include <stropts.h>
17323 ...
17324 int fd;
17325 char ctrlbuf[128];
17326 char databuf[512];
17327 struct strbuf ctrl;
17328 struct strbuf data;
17329 int flags = 0;
17330 int ret;

17331 ctrl.buf = ctrlbuf;
17332 ctrl.maxlen = sizeof(ctrlbuf);

17333 data.buf = databuf;
17334 data.maxlen = sizeof(databuf);

17335 ret = getmsg (fd, &ctrl, &data, &flags);
```

##### 17336 Getting the First Message off the Queue

17337 In the following example, the call to *getpmsg()* retrieves the first available message on the  
17338 associated STREAM-head read queue.

```
17339 #include <stropts.h>
17340 ...

17341 int fd;
17342 char ctrlbuf[128];
17343 char databuf[512];
17344 struct strbuf ctrl;
17345 struct strbuf data;
17346 int band = 0;
17347 int flags = MSG_ANY;
17348 int ret;

17349 ctrl.buf = ctrlbuf;
17350 ctrl.maxlen = sizeof(ctrlbuf);

17351 data.buf = databuf;
17352 data.maxlen = sizeof(databuf);

17353 ret = getpmsg (fd, &ctrl, &data, &band, &flags);
```

17354 **APPLICATION USAGE**

17355 None.

17356 **RATIONALE**

17357 None.

17358 **FUTURE DIRECTIONS**

17359 None.

17360 **SEE ALSO**

17361 Section 2.6 (on page 38), *poll()*, *putmsg()*, *read()*, *write()*, the Base Definitions volume of  
17362 IEEE Std 1003.1-2001, <**stropts.h**>

17363 **CHANGE HISTORY**

17364 First released in Issue 4, Version 2.

17365 **Issue 5**

17366 Moved from X/OPEN UNIX extension to BASE.

17367 A paragraph regarding “high-priority control parts of messages” is added to the RETURN  
17368 VALUE section.

17369 **Issue 6**

17370 This function is marked as part of the XSI STREAMS Option Group.

17371 The **restrict** keyword is added to the *getmsg()* and *getpmsg()* prototypes for alignment with the  
17372 ISO/IEC 9899:1999 standard.

## 17373 NAME

17374 getnameinfo — get name information

## 17375 SYNOPSIS

17376 #include &lt;sys/socket.h&gt;

17377 #include &lt;netdb.h&gt;

```
17378 int getnameinfo(const struct sockaddr *restrict sa, socklen_t salen,
17379 char *restrict node, socklen_t nodelen, char *restrict service,
17380 socklen_t servicelen, int flags);
```

## 17381 DESCRIPTION

17382 The *getnameinfo()* function shall translate a socket address to a node name and service location,  
17383 all of which are defined as in *getaddrinfo()*.

17384 The *sa* argument points to a socket address structure to be translated.

17385 IP6 If the socket address structure contains an IPv4-mapped IPv6 address or an IPv4-compatible  
17386 IPv6 address, the implementation shall extract the embedded IPv4 address and lookup the node  
17387 name for that IPv4 address.

17388 **Note:** The IPv6 unspecified address ("::") and the IPv6 loopback address ("::1") are not IPv4-  
17389 compatible addresses. If the address is the IPv6 unspecified address ("::"), a lookup is not  
17390 performed, and the [EAI\_NONAME] error is returned.

17391 If the *node* argument is non-NULL and the *nodelen* argument is non-zero, then the *node* argument  
17392 points to a buffer able to contain up to *nodelen* characters that receives the node name as a null-  
17393 terminated string. If the *node* argument is NULL or the *nodelen* argument is zero, the node name  
17394 shall not be returned. If the node's name cannot be located, the numeric form of the address  
17395 contained in the socket address structure pointed to by the *sa* argument is returned instead of its  
17396 name.

17397 If the *service* argument is non-NULL and the *servicelen* argument is non-zero, then the *service*  
17398 argument points to a buffer able to contain up to *servicelen* bytes that receives the service name  
17399 as a null-terminated string. If the *service* argument is NULL or the *servicelen* argument is zero,  
17400 the service name shall not be returned. If the service's name cannot be located, the numeric form  
17401 of the service address (for example, its port number) shall be returned instead of its name.

17402 The *flags* argument is a flag that changes the default actions of the function. By default the fully-  
17403 qualified domain name (FQDN) for the host shall be returned, but:

- 17404 • If the flag bit NI\_NOFQDN is set, only the node name portion of the FQDN shall be returned  
17405 for local hosts.
- 17406 • If the flag bit NI\_NUMERICHOST is set, the numeric form of the address contained in the  
17407 socket address structure pointed to by the *sa* argument shall be returned instead of its name,  
17408 under all circumstances.
- 17409 • If the flag bit NI\_NAMEREQD is set, an error shall be returned if the host's name cannot be  
17410 located.
- 17411 • If the flag bit NI\_NUMERICSERV is set, the numeric form of the service address shall be  
17412 returned (for example, its port number) instead of its name, under all circumstances.
- 17413 • If the flag bit NI\_NUMERICSERVICE is set, the numeric form of the scope identifier shall be  
17414 returned (for example, interface index) instead of its name. This flag shall be ignored if the *sa*  
17415 argument is not an IPv6 address.
- 17416 • If the flag bit NI\_DGRAM is set, this indicates that the service is a datagram service  
17417 (SOCK\_DGRAM). The default behavior shall assume that the service is a stream service

17418 (SOCK\_STREAM).

17419 **Notes:**

- 17420 1. The two NI\_NUMERICxxx flags are required to support the `-n` flag that many  
17421 commands provide.
- 17422 2. The NI\_DGRAM flag is required for the few AF\_INET and AF\_INET6 port numbers (for  
17423 example, [512,514]) that represent different services for UDP and TCP.

17424 The `getnameinfo()` function shall be thread-safe.

#### 17425 RETURN VALUE

17426 A zero return value for `getnameinfo()` indicates successful completion; a non-zero return value  
17427 indicates failure. The possible values for the failures are listed in the ERRORS section.

17428 Upon successful completion, `getnameinfo()` shall return the *node* and *service* names, if requested,  
17429 in the buffers provided. The returned names are always null-terminated strings.

#### 17430 ERRORS

17431 The `getnameinfo()` function shall fail and return the corresponding value if:

17432 [EAI\_AGAIN] The name could not be resolved at this time. Future attempts may succeed.

17433 [EAI\_BADFLAGS]

17434 The *flags* had an invalid value.

17435 [EAI\_FAIL] A non-recoverable error occurred.

17436 [EAI\_FAMILY] The address family was not recognized or the address length was invalid for  
17437 the specified family.

17438 [EAI\_MEMORY] There was a memory allocation failure.

17439 [EAI\_NONAME] The name does not resolve for the supplied parameters.

17440 NI\_NAMEREQD is set and the host's name cannot be located, or both  
17441 *nodename* and *servname* were null.

17442 [EAI\_OVERFLOW]

17443 An argument buffer overflowed. The buffer pointed to by the *node* argument  
17444 or the *service* argument was too small.

17445 [EAI\_SYSTEM] A system error occurred. The error code can be found in *errno*.

#### 17446 EXAMPLES

17447 None.

#### 17448 APPLICATION USAGE

17449 If the returned values are to be used as part of any further name resolution (for example, passed  
17450 to `getaddrinfo()`), applications should provide buffers large enough to store any result possible  
17451 on the system.

17452 Given the IPv4-mapped IPv6 address "`::ffff:1.2.3.4`", the implementation performs a  
17453 lookup as if the socket address structure contains the IPv4 address "`1.2.3.4`".

#### 17454 RATIONALE

17455 None.

#### 17456 FUTURE DIRECTIONS

17457 None.

17458 **SEE ALSO**

17459 *gai\_strerror()*, *getaddrinfo()*, *getservbyname()*, *inet\_ntop()*, *socket()*, the Base Definitions volume of  
17460 IEEE Std 1003.1-2001, <**netdb.h**>, <**sys/socket.h**>

17461 **CHANGE HISTORY**

17462 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

17463 The **restrict** keyword is added to the *getnameinfo()* prototype for alignment with the  
17464 ISO/IEC 9899:1999 standard.

17465 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/23 is applied, making various changes in  
17466 the SYNOPSIS and DESCRIPTION for alignment with IPv6.

17467 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/24 is applied, adding the  
17468 [EAI\_OVERFLOW] error to the ERRORS section.

17469 **NAME**

17470           getnetbyaddr, getnetbyname, getnetent — network database functions

17471 **SYNOPSIS**

17472           #include <netdb.h>

17473           struct netent \*getnetbyaddr(uint32\_t net, int type);

17474           struct netent \*getnetbyname(const char \*name);

17475           struct netent \*getnetent(void);

17476 **DESCRIPTION**

17477           Refer to *endnetent()*.

## 17478 NAME

17479        getopt, optarg, opterr, optind, optopt — command option parsing

## 17480 SYNOPSIS

17481        #include &lt;unistd.h&gt;

17482        int getopt(int argc, char \* const argv[], const char \*optstring);

17483        extern char \*optarg;

17484        extern int optind, opterr, optopt;

## 17485 DESCRIPTION

17486        The *getopt()* function is a command-line parser that shall follow Utility Syntax Guidelines 3, 4, 5,  
 17487        6, 7, 9, and 10 in the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2, Utility Syntax  
 17488        Guidelines.

17489        The parameters *argc* and *argv* are the argument count and argument array as passed to *main()*  
 17490        (see *exec*). The argument *optstring* is a string of recognized option characters; if a character is  
 17491        followed by a colon, the option takes an argument. All option characters allowed by Utility  
 17492        Syntax Guideline 3 are allowed in *optstring*. The implementation may accept other characters as  
 17493        an extension.

17494        The variable *optind* is the index of the next element of the *argv[]* vector to be processed. It shall  
 17495        be initialized to 1 by the system, and *getopt()* shall update it when it finishes with each element  
 17496        of *argv[]*. When an element of *argv[]* contains multiple option characters, it is unspecified how  
 17497        *getopt()* determines which options have already been processed.

17498        The *getopt()* function shall return the next option character (if one is found) from *argv* that  
 17499        matches a character in *optstring*, if there is one that matches. If the option takes an argument,  
 17500        *getopt()* shall set the variable *optarg* to point to the option-argument as follows:

- 17501        1. If the option was the last character in the string pointed to by an element of *argv*, then  
 17502        *optarg* shall contain the next element of *argv*, and *optind* shall be incremented by 2. If the  
 17503        resulting value of *optind* is greater than *argc*, this indicates a missing option-argument, and  
 17504        *getopt()* shall return an error indication.
- 17505        2. Otherwise, *optarg* shall point to the string following the option character in that element of  
 17506        *argv*, and *optind* shall be incremented by 1.

17507        If, when *getopt()* is called:

17508        *argv[optind]*    is a null pointer  
 17509        \**argv[optind]*    is not the character -  
 17510        *argv[optind]*    points to the string "--"

17511        *getopt()* shall return -1 without changing *optind*. If:

17512        *argv[optind]*    points to the string "--"

17513        *getopt()* shall return -1 after incrementing *optind*.

17514        If *getopt()* encounters an option character that is not contained in *optstring*, it shall return the  
 17515        question-mark ('?') character. If it detects a missing option-argument, it shall return the colon  
 17516        character (':') if the first character of *optstring* was a colon, or a question-mark character ('?')  
 17517        otherwise. In either case, *getopt()* shall set the variable *optopt* to the option character that caused  
 17518        the error. If the application has not set the variable *opterr* to 0 and the first character of *optstring*  
 17519        is not a colon, *getopt()* shall also print a diagnostic message to *stderr* in the format specified for  
 17520        the *getopts* utility.

17521        The *getopt()* function need not be reentrant. A function that is not required to be reentrant is not  
 17522        required to be thread-safe.

17523 **RETURN VALUE**

17524 The *getopt()* function shall return the next option character specified on the command line.

17525 A colon (':') shall be returned if *getopt()* detects a missing argument and the first character of  
17526 *optstring* was a colon (':').

17527 A question mark ('?') shall be returned if *getopt()* encounters an option character not in  
17528 *optstring* or detects a missing argument and the first character of *optstring* was not a colon (':').

17529 Otherwise, *getopt()* shall return -1 when all command line options are parsed.

17530 **ERRORS**

17531 No errors are defined.

17532 **EXAMPLES**17533 **Parsing Command Line Options**

17534 The following code fragment shows how you might process the arguments for a utility that can  
17535 take the mutually-exclusive options *a* and *b* and the options *f* and *o*, both of which require  
17536 arguments:

```
17537 #include <unistd.h>
17538
17539 int
17540 main(int argc, char *argv[])
17541 {
17542 int c;
17543 int bflg, aflag, errflag;
17544 char *ifile;
17545 char *ofile;
17546 extern char *optarg;
17547 extern int optind, optopt;
17548 . . .
17549 while ((c = getopt(argc, argv, ":abf:o:")) != -1) {
17550 switch(c) {
17551 case 'a':
17552 if (bflg)
17553 errflag++;
17554 else
17555 aflag++;
17556 break;
17557 case 'b':
17558 if (aflag)
17559 errflag++;
17560 else {
17561 bflg++;
17562 bproc();
17563 }
17564 break;
17565 case 'f':
17566 ifile = optarg;
17567 break;
17568 case 'o':
17569 ofile = optarg;
17570 break;
```

```

17570 case ':': /* -f or -o without operand */
17571 fprintf(stderr,
17572 "Option -%c requires an operand\n", optopt);
17573 errflg++;
17574 break;
17575 case '?':
17576 fprintf(stderr,
17577 "Unrecognized option: -%c\n", optopt);
17578 errflg++;
17579 }
17580 }
17581 if (errflg) {
17582 fprintf(stderr, "usage: . . . ");
17583 exit(2);
17584 }
17585 for (; optind < argc; optind++) {
17586 if (access(argv[optind], R_OK)) {
17587 . . .
17588 }

```

17589 **This code accepts any of the following as equivalent:**

```

17590 cmd -ao arg path path
17591 cmd -a -o arg path path
17592 cmd -o arg -a path path
17593 cmd -a -o arg -- path path
17594 cmd -a -oarg path path
17595 cmd -aoarg path path

```

### 17596 **Checking Options and Arguments**

17597 The following example parses a set of command line options and prints messages to standard  
17598 output for each option and argument that it encounters.

```

17599 #include <unistd.h>
17600 #include <stdio.h>
17601 ...
17602 int c;
17603 char *filename;
17604 extern char *optarg;
17605 extern int optind, optopt, opterr;
17606 ...
17607 while ((c = getopt(argc, argv, ":abf:")) != -1) {
17608 switch(c) {
17609 case 'a':
17610 printf("a is set\n");
17611 break;
17612 case 'b':
17613 printf("b is set\n");
17614 break;
17615 case 'f':
17616 filename = optarg;
17617 printf("filename is %s\n", filename);
17618 break;

```

```

17619 case ':':
17620 printf("-%c without filename\n", optopt);
17621 break;
17622 case '?':
17623 printf("unknown arg %c\n", optopt);
17624 break;
17625 }
17626 }

```

### 17627 **Selecting Options from the Command Line**

17628 The following example selects the type of database routines the user wants to use based on the  
 17629 *Options* argument.

```

17630 #include <unistd.h>
17631 #include <string.h>
17632 ...
17633 char *Options = "hdbtl";
17634 ...
17635 int dbtype, i;
17636 char c;
17637 char *st;
17638 ...
17639 dbtype = 0;
17640 while ((c = getopt(argc, argv, Options)) != -1) {
17641 if ((st = strchr(Options, c)) != NULL) {
17642 dbtype = st - Options;
17643 break;
17644 }
17645 }

```

### 17646 **APPLICATION USAGE**

17647 The *getopt()* function is only required to support option characters included in Utility Syntax  
 17648 Guideline 3. Many historical implementations of *getopt()* support other characters as options.  
 17649 This is an allowed extension, but applications that use extensions are not maximally portable.  
 17650 Note that support for multi-byte option characters is only possible when such characters can be  
 17651 represented as type **int**.

### 17652 **RATIONALE**

17653 The *optopt* variable represents historical practice and allows the application to obtain the identity  
 17654 of the invalid option.

17655 The description has been written to make it clear that *getopt()*, like the *getopts* utility, deals with  
 17656 option-arguments whether separated from the option by <blank>s or not. Note that the  
 17657 requirements on *getopt()* and *getopts* are more stringent than the Utility Syntax Guidelines.

17658 The *getopt()* function shall return  $-1$ , rather than EOF, so that <**stdio.h**> is not required.

17659 The special significance of a colon as the first character of *optstring* makes *getopt()* consistent  
 17660 with the *getopts* utility. It allows an application to make a distinction between a missing  
 17661 argument and an incorrect option letter without having to examine the option letter. It is true  
 17662 that a missing argument can only be detected in one case, but that is a case that has to be  
 17663 considered.

17664 **FUTURE DIRECTIONS**

17665           None.

17666 **SEE ALSO**

17667           *exec*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>, the Shell and Utilities  
17668           volume of IEEE Std 1003.1-2001

17669 **CHANGE HISTORY**

17670           First released in Issue 1. Derived from Issue 1 of the SVID.

17671 **Issue 5**

17672           A note indicating that the *getopt()* function need not be reentrant is added to the DESCRIPTION.

17673 **Issue 6**

17674           IEEE PASC Interpretation 1003.2 #150 is applied.

17675 **NAME**

17676           getpeername — get the name of the peer socket

17677 **SYNOPSIS**

17678           #include <sys/socket.h>

17679           int getpeername(int *socket*, struct sockaddr \*restrict *address*,  
17680                           socklen\_t \*restrict *address\_len*);

17681 **DESCRIPTION**

17682           The *getpeername()* function shall retrieve the peer address of the specified socket, store this  
17683           address in the **sockaddr** structure pointed to by the *address* argument, and store the length of this  
17684           address in the object pointed to by the *address\_len* argument.

17685           If the actual length of the address is greater than the length of the supplied **sockaddr** structure,  
17686           the stored address shall be truncated.

17687           If the protocol permits connections by unbound clients, and the peer is not bound, then the value  
17688           stored in the object pointed to by *address* is unspecified.

17689 **RETURN VALUE**

17690           Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
17691           indicate the error.

17692 **ERRORS**

17693           The *getpeername()* function shall fail if:

17694           [EBADF]           The *socket* argument is not a valid file descriptor.

17695           [EINVAL]          The socket has been shut down.

17696           [ENOTCONN]       The socket is not connected or otherwise has not had the peer pre-specified.

17697           [ENOTSOCK]       The *socket* argument does not refer to a socket.

17698           [EOPNOTSUPP]      The operation is not supported for the socket protocol.

17699           The *getpeername()* function may fail if:

17700           [ENOBUFS]        Insufficient resources were available in the system to complete the call.

17701 **EXAMPLES**

17702           None.

17703 **APPLICATION USAGE**

17704           None.

17705 **RATIONALE**

17706           None.

17707 **FUTURE DIRECTIONS**

17708           None.

17709 **SEE ALSO**

17710           *accept()*, *bind()*, *getsockname()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
17711           <sys/socket.h>

17712 **CHANGE HISTORY**

17713           First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

17714           The **restrict** keyword is added to the *getpeername()* prototype for alignment with the  
17715           ISO/IEC 9899:1999 standard.

17716 **NAME**

17717       getpgid — get the process group ID for a process

17718 **SYNOPSIS**

17719 xSI       #include <unistd.h>

17720       pid\_t getpgid(pid\_t pid);

17721

17722 **DESCRIPTION**

17723       The *getpgid()* function shall return the process group ID of the process whose process ID is equal  
17724       to *pid*. If *pid* is equal to 0, *getpgid()* shall return the process group ID of the calling process.

17725 **RETURN VALUE**

17726       Upon successful completion, *getpgid()* shall return a process group ID. Otherwise, it shall return  
17727       (**pid\_t**)-1 and set *errno* to indicate the error.

17728 **ERRORS**

17729       The *getpgid()* function shall fail if:

17730       [EPERM]       The process whose process ID is equal to *pid* is not in the same session as the  
17731       calling process, and the implementation does not allow access to the process  
17732       group ID of that process from the calling process.

17733       [ESRCH]       There is no process with a process ID equal to *pid*.

17734       The *getpgid()* function may fail if:

17735       [EINVAL]       The value of the *pid* argument is invalid.

17736 **EXAMPLES**

17737       None.

17738 **APPLICATION USAGE**

17739       None.

17740 **RATIONALE**

17741       None.

17742 **FUTURE DIRECTIONS**

17743       None.

17744 **SEE ALSO**

17745       *exec*, *fork()*, *getpgrp()*, *getpid()*, *getsid()*, *setpgid()*, *setsid()*, the Base Definitions volume of  
17746       IEEE Std 1003.1-2001, <unistd.h>

17747 **CHANGE HISTORY**

17748       First released in Issue 4, Version 2.

17749 **Issue 5**

17750       Moved from X/OPEN UNIX extension to BASE.

17751 **NAME**

17752           getpgrp — get the process group ID of the calling process

17753 **SYNOPSIS**

17754           #include <unistd.h>

17755           pid\_t getpgrp(void);

17756 **DESCRIPTION**

17757           The *getpgrp()* function shall return the process group ID of the calling process.

17758 **RETURN VALUE**

17759           The *getpgrp()* function shall always be successful and no return value is reserved to indicate an error.

17761 **ERRORS**

17762           No errors are defined.

17763 **EXAMPLES**

17764           None.

17765 **APPLICATION USAGE**

17766           None.

17767 **RATIONALE**

17768           4.3 BSD provides a *getpgrp()* function that returns the process group ID for a specified process.  
17769           Although this function supports job control, all known job control shells always specify the  
17770           calling process with this function. Thus, the simpler System V *getpgrp()* suffices, and the added  
17771           complexity of the 4.3 BSD *getpgrp()* is provided by the XSI extension *getpgid()*.

17772 **FUTURE DIRECTIONS**

17773           None.

17774 **SEE ALSO**

17775           *exec*, *fork()*, *getpgid()*, *getpid()*, *getppid()*, *kill()*, *setpgid()*, *setsid()*, the Base Definitions volume of  
17776           IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>

17777 **CHANGE HISTORY**

17778           First released in Issue 1. Derived from Issue 1 of the SVID.

17779 **Issue 6**

17780           In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

17781           The following new requirements on POSIX implementations derive from alignment with the  
17782           Single UNIX Specification:

- 17783           • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
17784           required for conforming implementations of previous POSIX specifications, it was not  
17785           required for UNIX applications.

17786 **NAME**

17787            **getpid** — get the process ID

17788 **SYNOPSIS**

17789            #include <unistd.h>

17790            pid\_t getpid(void);

17791 **DESCRIPTION**

17792            The *getpid()* function shall return the process ID of the calling process.

17793 **RETURN VALUE**

17794            The *getpid()* function shall always be successful and no return value is reserved to indicate an error.

17796 **ERRORS**

17797            No errors are defined.

17798 **EXAMPLES**

17799            None.

17800 **APPLICATION USAGE**

17801            None.

17802 **RATIONALE**

17803            None.

17804 **FUTURE DIRECTIONS**

17805            None.

17806 **SEE ALSO**

17807            *exec*, *fork()*, *getpgrp()*, *getppid()*, *kill()*, *setpgid()*, *setsid()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>

17809 **CHANGE HISTORY**

17810            First released in Issue 1. Derived from Issue 1 of the SVID.

17811 **Issue 6**

17812            In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

17813            The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 17814
- 17815            • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
  - 17816            required for conforming implementations of previous POSIX specifications, it was not
  - 17817            required for UNIX applications.

17818 **NAME**

17819       getpmsg — receive next message from a STREAMS file

17820 **SYNOPSIS**

```
17821 xSI #include <stropts.h>
```

```
17822 int getpmsg(int fildev, struct strbuf *restrict ctlptr,
17823 struct strbuf *restrict dataptr, int *restrict bandp,
17824 int *restrict flagsp);
```

17825

17826 **DESCRIPTION**

17827       Refer to *getmsg()*.

17828 **NAME**

17829           getppid — get the parent process ID

17830 **SYNOPSIS**

17831           #include <unistd.h>

17832           pid\_t getppid(void);

17833 **DESCRIPTION**

17834           The *getppid()* function shall return the parent process ID of the calling process.

17835 **RETURN VALUE**

17836           The *getppid()* function shall always be successful and no return value is reserved to indicate an error.

17838 **ERRORS**

17839           No errors are defined.

17840 **EXAMPLES**

17841           None.

17842 **APPLICATION USAGE**

17843           None.

17844 **RATIONALE**

17845           None.

17846 **FUTURE DIRECTIONS**

17847           None.

17848 **SEE ALSO**

17849           *exec*, *fork()*, *getpgid()*, *getpgrp()*, *getpid()*, *kill()*, *setpgid()*, *setsid()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>

17851 **CHANGE HISTORY**

17852           First released in Issue 1. Derived from Issue 1 of the SVID.

17853 **Issue 6**

17854           In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

17855           The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 17857
  - The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- 17858
- 17859

17860 **NAME**

17861 getpriority, setpriority — get and set the nice value

17862 **SYNOPSIS**17863 XSI 

```
#include <sys/resource.h>
```

17864 

```
int getpriority(int which, id_t who);
```

17865 

```
int setpriority(int which, id_t who, int value);
```

17866

17867 **DESCRIPTION**

17868 The *getpriority()* function shall obtain the nice value of a process, process group, or user. The  
 17869 *setpriority()* function shall set the nice value of a process, process group, or user to  
 17870 *value*+{NZERO}.

17871 Target processes are specified by the values of the *which* and *who* arguments. The *which*  
 17872 argument may be one of the following values: PRIO\_PROCESS, PRIO\_PGRP, or PRIO\_USER,  
 17873 indicating that the *who* argument is to be interpreted as a process ID, a process group ID, or an  
 17874 effective user ID, respectively. A 0 value for the *who* argument specifies the current process,  
 17875 process group, or user.

17876 The nice value set with *setpriority()* shall be applied to the process. If the process is multi-  
 17877 threaded, the nice value shall affect all system scope threads in the process.

17878 If more than one process is specified, *getpriority()* shall return value {NZERO} less than the  
 17879 lowest nice value pertaining to any of the specified processes, and *setpriority()* shall set the nice  
 17880 values of all of the specified processes to *value*+{NZERO}.

17881 The default nice value is {NZERO}; lower nice values shall cause more favorable scheduling.  
 17882 While the range of valid nice values is [0,{NZERO}\*2-1], implementations may enforce more  
 17883 restrictive limits. If *value*+{NZERO} is less than the system's lowest supported nice value,  
 17884 *setpriority()* shall set the nice value to the lowest supported value; if *value*+{NZERO} is greater  
 17885 than the system's highest supported nice value, *setpriority()* shall set the nice value to the highest  
 17886 supported value.

17887 Only a process with appropriate privileges can lower its nice value.

17888 PS|TPS Any processes or threads using SCHED\_FIFO or SCHED\_RR shall be unaffected by a call to  
 17889 *setpriority()*. This is not considered an error. A process which subsequently reverts to  
 17890 SCHED\_OTHER need not have its priority affected by such a *setpriority()* call.

17891 The effect of changing the nice value may vary depending on the process-scheduling algorithm  
 17892 in effect.

17893 Since *getpriority()* can return the value -1 on successful completion, it is necessary to set *errno* to  
 17894 0 prior to a call to *getpriority()*. If *getpriority()* returns the value -1, then *errno* can be checked to  
 17895 see if an error occurred or if the value is a legitimate nice value.

17896 **RETURN VALUE**

17897 Upon successful completion, *getpriority()* shall return an integer in the range -{NZERO} to  
 17898 {NZERO}-1. Otherwise, -1 shall be returned and *errno* set to indicate the error.

17899 Upon successful completion, *setpriority()* shall return 0; otherwise, -1 shall be returned and *errno*  
 17900 set to indicate the error.

17901 **ERRORS**

17902 The *getpriority()* and *setpriority()* functions shall fail if:

17903 [ESRCH] No process could be located using the *which* and *who* argument values  
 17904 specified.

17905 [EINVAL] The value of the *which* argument was not recognized, or the value of the *who*  
 17906 argument is not a valid process ID, process group ID, or user ID.

17907 In addition, *setpriority()* may fail if:

17908 [EPERM] A process was located, but neither the real nor effective user ID of the  
 17909 executing process match the effective user ID of the process whose nice value  
 17910 is being changed.

17911 [EACCES] A request was made to change the nice value to a lower numeric value and  
 17912 the current process does not have appropriate privileges.

#### 17913 EXAMPLES

##### 17914 Using *getpriority()*

17915 The following example returns the current scheduling priority for the process ID returned by the  
 17916 call to *getpid()*.

```
17917 #include <sys/resource.h>
17918 ...
17919 int which = PRIO_PROCESS;
17920 id_t pid;
17921 int ret;

17922 pid = getpid();
17923 ret = getpriority(which, pid);
```

##### 17924 Using *setpriority()*

17925 The following example sets the priority for the current process ID to  $-20$ .

```
17926 #include <sys/resource.h>
17927 ...
17928 int which = PRIO_PROCESS;
17929 id_t pid;
17930 int priority = -20;
17931 int ret;

17932 pid = getpid();
17933 ret = setpriority(which, pid, priority);
```

#### 17934 APPLICATION USAGE

17935 The *getpriority()* and *setpriority()* functions work with an offset nice value (nice value  
 17936  $-\{\text{NZERO}\}$ ). The nice value is in the range  $[0, 2*\{\text{NZERO}\} - 1]$ , while the return value for  
 17937 *getpriority()* and the third parameter for *setpriority()* are in the range  $[-\{\text{NZERO}\}, \{\text{NZERO}\} - 1]$ .

#### 17938 RATIONALE

17939 None.

#### 17940 FUTURE DIRECTIONS

17941 None.

#### 17942 SEE ALSO

17943 *nice()*, *sched\_get\_priority\_max()*, *sched\_setscheduler()*, the Base Definitions volume of  
 17944 IEEE Std 1003.1-2001, [<sys/resource.h>](#)

17945 **CHANGE HISTORY**

17946 First released in Issue 4, Version 2.

17947 **Issue 5**

17948 Moved from X/OPEN UNIX extension to BASE.

17949 The DESCRIPTION is reworded in terms of the nice value rather than *priority* to avoid confusion  
17950 with functionality in the POSIX Realtime Extension.

17951 **NAME**

17952           getprotobyname, getprotobynumber, getprotent — network protocol database functions

17953 **SYNOPSIS**

17954           #include <netdb.h>

17955           struct protoent \*getprotobyname(const char \*name);

17956           struct protoent \*getprotobynumber(int proto);

17957           struct protoent \*getprotoent(void);

17958 **DESCRIPTION**

17959           Refer to *endprotoent()*.

17960 **NAME**

17961           getpwent — get user database entry

17962 **SYNOPSIS**

17963 xSI       #include &lt;pwd.h&gt;

17964           struct passwd \*getpwent(void);

17965

17966 **DESCRIPTION**17967           Refer to *endpwent()*.

## 17968 NAME

17969 getpwnam, getpwnam\_r — search user database for a name

## 17970 SYNOPSIS

17971 #include <pwd.h>

17972 struct passwd \*getpwnam(const char \*name);

17973 TSF int getpwnam\_r(const char \*name, struct passwd \*pwd, char \*buffer,

17974 size\_t bufsize, struct passwd \*\*result);

17975

## 17976 DESCRIPTION

17977 The *getpwnam()* function shall search the user database for an entry with a matching *name*.

17978 The *getpwnam()* function need not be reentrant. A function that is not required to be reentrant is  
17979 not required to be thread-safe.

17980 Applications wishing to check for error situations should set *errno* to 0 before calling  
17981 *getpwnam()*. If *getpwnam()* returns a null pointer and *errno* is non-zero, an error occurred.

17982 TSF The *getpwnam\_r()* function shall update the **passwd** structure pointed to by *pwd* and store a  
17983 pointer to that structure at the location pointed to by *result*. The structure shall contain an entry  
17984 from the user database with a matching *name*. Storage referenced by the structure is allocated  
17985 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. The  
17986 maximum size needed for this buffer can be determined with the `{_SC_GETPW_R_SIZE_MAX}`  
17987 *sysconf()* parameter. A NULL pointer shall be returned at the location pointed to by *result* on  
17988 error or if the requested entry is not found.

## 17989 RETURN VALUE

17990 The *getpwnam()* function shall return a pointer to a **struct passwd** with the structure as defined  
17991 in `<pwd.h>` with a matching entry if found. A null pointer shall be returned if the requested  
17992 entry is not found, or an error occurs. On error, *errno* shall be set to indicate the error.

17993 The return value may point to a static area which is overwritten by a subsequent call to  
17994 *getpwent()*, *getpwnam()*, or *getpwuid()*.

17995 TSF If successful, the *getpwnam\_r()* function shall return zero; otherwise, an error number shall be  
17996 returned to indicate the error.

## 17997 ERRORS

17998 The *getpwnam()* and *getpwnam\_r()* functions may fail if:

17999 [EIO] An I/O error has occurred.

18000 [EINTR] A signal was caught during *getpwnam()*.

18001 [EMFILE] `{OPEN_MAX}` file descriptors are currently open in the calling process.

18002 [ENFILE] The maximum allowable number of files is currently open in the system.

18003 The *getpwnam\_r()* function may fail if:

18004 TSF [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to  
18005 be referenced by the resulting **passwd** structure.

18006 **EXAMPLES**18007 **Getting an Entry for the Login Name**

18008 The following example uses the *getlogin()* function to return the name of the user who logged in;  
 18009 this information is passed to the *getpwnam()* function to get the user database entry for that user.

```

18010 #include <sys/types.h>
18011 #include <pwd.h>
18012 #include <unistd.h>
18013 #include <stdio.h>
18014 #include <stdlib.h>
18015 ...
18016 char *lgn;
18017 struct passwd *pw;
18018 ...
18019 if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
18020 fprintf(stderr, "Get of user information failed.\n"); exit(1);
18021 }
18022 ...

```

18023 **APPLICATION USAGE**

18024 Three names associated with the current process can be determined: *getpwuid(geteuid())* returns  
 18025 the name associated with the effective user ID of the process; *getlogin()* returns the name  
 18026 associated with the current login activity; and *getpwuid(getuid())* returns the name associated  
 18027 with the real user ID of the process.

18028 The *getpwnam\_r()* function is thread-safe and returns values in a user-supplied buffer instead of  
 18029 possibly using a static data area that may be overwritten by each call.

18030 **RATIONALE**

18031 None.

18032 **FUTURE DIRECTIONS**

18033 None.

18034 **SEE ALSO**

18035 *getpwuid()*, the Base Definitions volume of IEEE Std 1003.1-2001, <limits.h>, <pwd.h>,  
 18036 <sys/types.h>

18037 **CHANGE HISTORY**

18038 First released in Issue 1. Derived from System V Release 2.0.

18039 **Issue 5**

18040 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
 18041 VALUE section.

18042 The *getpwnam\_r()* function is included for alignment with the POSIX Threads Extension.

18043 A note indicating that the *getpwnam()* function need not be reentrant is added to the  
 18044 DESCRIPTION.

18045 **Issue 6**

18046 The *getpwnam\_r()* function is marked as part of the Thread-Safe Functions option.

18047 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION  
 18048 describing matching the *name*.

- 18049 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.
- 18050 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.
- 18051 The following new requirements on POSIX implementations derive from alignment with the  
18052 Single UNIX Specification:
- 18053 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
18054 required for conforming implementations of previous POSIX specifications, it was not  
18055 required for UNIX applications.
  - 18056 • In the RETURN VALUE section, the requirement to set *errno* on error is added.
  - 18057 • The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.
- 18058 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
18059 its avoidance of possibly using a static data area.
- 18060 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the  
18061 buffer from *bufsize* characters to bytes.

## 18062 NAME

18063 getpwuid, getpwuid\_r — search user database for a user ID

## 18064 SYNOPSIS

18065 #include &lt;pwd.h&gt;

18066 struct passwd \*getpwuid(uid\_t uid);

18067 TSF int getpwuid\_r(uid\_t uid, struct passwd \*pwd, char \*buffer,

18068 size\_t bufsize, struct passwd \*\*result);

18069

## 18070 DESCRIPTION

18071 The *getpwuid()* function shall search the user database for an entry with a matching *uid*.18072 The *getpwuid()* function need not be reentrant. A function that is not required to be reentrant is  
18073 not required to be thread-safe.18074 Applications wishing to check for error situations should set *errno* to 0 before calling *getpwuid()*.  
18075 If *getpwuid()* returns a null pointer and *errno* is set to non-zero, an error occurred.18076 TSF The *getpwuid\_r()* function shall update the **passwd** structure pointed to by *pwd* and store a  
18077 pointer to that structure at the location pointed to by *result*. The structure shall contain an entry  
18078 from the user database with a matching *uid*. Storage referenced by the structure is allocated  
18079 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. The  
18080 maximum size needed for this buffer can be determined with the `{_SC_GETPW_R_SIZE_MAX}`  
18081 *sysconf()* parameter. A NULL pointer shall be returned at the location pointed to by *result* on  
18082 error or if the requested entry is not found.

## 18083 RETURN VALUE

18084 The *getpwuid()* function shall return a pointer to a **struct passwd** with the structure as defined in  
18085 <**pwd.h**> with a matching entry if found. A null pointer shall be returned if the requested entry  
18086 is not found, or an error occurs. On error, *errno* shall be set to indicate the error.18087 The return value may point to a static area which is overwritten by a subsequent call to  
18088 *getpwent()*, *getpwnam()*, or *getpwuid()*.18089 TSF If successful, the *getpwuid\_r()* function shall return zero; otherwise, an error number shall be  
18090 returned to indicate the error.

## 18091 ERRORS

18092 The *getpwuid()* and *getpwuid\_r()* functions may fail if:

18093 [EIO] An I/O error has occurred.

18094 [EINTR] A signal was caught during *getpwuid()*.

18095 [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

18096 [ENFILE] The maximum allowable number of files is currently open in the system.

18097 The *getpwuid\_r()* function may fail if:18098 TSF [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to  
18099 be referenced by the resulting **passwd** structure.

## 18100 EXAMPLES

18101 **Getting an Entry for the Root User**

18102 The following example gets the user database entry for the user with user ID 0 (root).

```
18103 #include <sys/types.h>
18104 #include <pwd.h>
18105 ...
18106 uid_t id = 0;
18107 struct passwd *pwd;
18108 pwd = getpwuid(id);
```

18109 **Finding the Name for the Effective User ID**

18110 The following example defines *pws* as a pointer to a structure of type **passwd**, which is used to  
 18111 store the structure pointer returned by the call to the *getpwuid()* function. The *geteuid()* function  
 18112 shall return the effective user ID of the calling process; this is used as the search criteria for the  
 18113 *getpwuid()* function. The call to *getpwuid()* shall return a pointer to the structure containing that  
 18114 user ID value.

```
18115 #include <unistd.h>
18116 #include <sys/types.h>
18117 #include <pwd.h>
18118 ...
18119 struct passwd *pws;
18120 pws = getpwuid(geteuid());
```

18121 **Finding an Entry in the User Database**

18122 The following example uses *getpwuid()* to search the user database for a user ID that was  
 18123 previously stored in a **stat** structure, then prints out the user name if it is found. If the user is not  
 18124 found, the program prints the numeric value of the user ID for the entry.

```
18125 #include <sys/types.h>
18126 #include <pwd.h>
18127 #include <stdio.h>
18128 ...
18129 struct stat statbuf;
18130 struct passwd *pwd;
18131 ...
18132 if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
18133 printf(" %-8.8s", pwd->pw_name);
18134 else
18135 printf(" %-8d", statbuf.st_uid);
```

## 18136 APPLICATION USAGE

18137 Three names associated with the current process can be determined: *getpwuid(geteuid())* returns  
 18138 the name associated with the effective user ID of the process; *getlogin()* returns the name  
 18139 associated with the current login activity; and *getpwuid(getuid())* returns the name associated  
 18140 with the real user ID of the process.

18141 The *getpwuid\_r()* function is thread-safe and returns values in a user-supplied buffer instead of  
 18142 possibly using a static data area that may be overwritten by each call.

18143 **RATIONALE**

18144 None.

18145 **FUTURE DIRECTIONS**

18146 None.

18147 **SEE ALSO**

18148 *getpwnam()*, *geteuid()*, *getuid()*, *getlogin()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
 18149 **<limits.h>**, **<pwd.h>**, **<sys/types.h>**

18150 **CHANGE HISTORY**

18151 First released in Issue 1. Derived from System V Release 2.0.

18152 **Issue 5**

18153 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
 18154 VALUE section.

18155 The *getpwuid\_r()* function is included for alignment with the POSIX Threads Extension.

18156 A note indicating that the *getpwuid()* function need not be reentrant is added to the  
 18157 DESCRIPTION.

18158 **Issue 6**18159 The *getpwuid\_r()* function is marked as part of the Thread-Safe Functions option.

18160 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION  
 18161 describing matching the *uid*.

18162 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

18163 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

18164 The following new requirements on POSIX implementations derive from alignment with the  
 18165 Single UNIX Specification:

18166 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was  
 18167 required for conforming implementations of previous POSIX specifications, it was not  
 18168 required for UNIX applications.

18169 • In the RETURN VALUE section, the requirement to set *errno* on error is added.

18170 • The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

18171 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
 18172 its avoidance of possibly using a static data area.

18173 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the  
 18174 buffer from *bufsize* characters to bytes.

## 18175 NAME

18176 getrlimit, setrlimit — control maximum resource consumption

## 18177 SYNOPSIS

18178 XSI 

```
#include <sys/resource.h>
```

18179 

```
int getrlimit(int resource, struct rlimit *rlp);
```

18180 

```
int setrlimit(int resource, const struct rlimit *rlp);
```

18181

## 18182 DESCRIPTION

18183 The *getrlimit()* function shall get, and the *setrlimit()* function shall set, limits on the consumption  
18184 of a variety of resources.18185 Each call to either *getrlimit()* or *setrlimit()* identifies a specific resource to be operated upon as  
18186 well as a resource limit. A resource limit is represented by an **rlimit** structure. The *rlim\_cur*  
18187 member specifies the current or soft limit and the *rlim\_max* member specifies the maximum or  
18188 hard limit. Soft limits may be changed by a process to any value that is less than or equal to the  
18189 hard limit. A process may (irreversibly) lower its hard limit to any value that is greater than or  
18190 equal to the soft limit. Only a process with appropriate privileges can raise a hard limit. Both  
18191 hard and soft limits can be changed in a single call to *setrlimit()* subject to the constraints  
18192 described above.18193 The value RLIM\_INFINITY, defined in *<sys/resource.h>*, shall be considered to be larger than  
18194 any other limit value. If a call to *getrlimit()* returns RLIM\_INFINITY for a resource, it means the  
18195 implementation shall not enforce limits on that resource. Specifying RLIM\_INFINITY as any  
18196 resource limit value on a successful call to *setrlimit()* shall inhibit enforcement of that resource  
18197 limit.

18198 The following resources are defined:

18199 **RLIMIT\_CORE** This is the maximum size of a **core** file, in bytes, that may be created by a  
18200 process. A limit of 0 shall prevent the creation of a **core** file. If this limit is  
18201 exceeded, the writing of a **core** file shall terminate at this size.18202 **RLIMIT\_CPU** This is the maximum amount of CPU time, in seconds, used by a process.  
18203 If this limit is exceeded, SIGXCPU shall be generated for the process. If  
18204 the process is catching or ignoring SIGXCPU, or all threads belonging to  
18205 that process are blocking SIGXCPU, the behavior is unspecified.18206 **RLIMIT\_DATA** This is the maximum size of a process' data segment, in bytes. If this limit  
18207 is exceeded, the *malloc()* function shall fail with *errno* set to [ENOMEM].18208 **RLIMIT\_FSIZE** This is the maximum size of a file, in bytes, that may be created by a  
18209 process. If a write or truncate operation would cause this limit to be  
18210 exceeded, SIGXFSZ shall be generated for the thread. If the thread is  
18211 blocking, or the process is catching or ignoring SIGXFSZ, continued  
18212 attempts to increase the size of a file from end-of-file to beyond the limit  
18213 shall fail with *errno* set to [EFBIG].18214 **RLIMIT\_NOFILE** This is a number one greater than the maximum value that the system  
18215 may assign to a newly-created descriptor. If this limit is exceeded,  
18216 functions that allocate a file descriptor shall fail with *errno* set to  
18217 [EMFILE]. This limit constrains the number of file descriptors that a  
18218 process may allocate.18219 **RLIMIT\_STACK** This is the maximum size of a process' stack, in bytes. The  
18220 implementation does not automatically grow the stack beyond this limit.

- 18221 If this limit is exceeded, SIGSEGV shall be generated for the thread. If the  
 18222 thread is blocking SIGSEGV, or the process is ignoring or catching  
 18223 SIGSEGV and has not made arrangements to use an alternate stack, the  
 18224 disposition of SIGSEGV shall be set to SIG\_DFL before it is generated.
- 18225 **RLIMIT\_AS** This is the maximum size of a process' total available memory, in bytes. If  
 18226 this limit is exceeded, the *malloc()* and *mmap()* functions shall fail with  
 18227 *errno* set to [ENOMEM]. In addition, the automatic stack growth fails  
 18228 with the effects outlined above.
- 18229 When using the *getrlimit()* function, if a resource limit can be represented correctly in an object  
 18230 of type **rlim\_t**, then its representation is returned; otherwise, if the value of the resource limit is  
 18231 equal to that of the corresponding saved hard limit, the value returned shall be  
 18232 RLIM\_SAVED\_MAX; otherwise, the value returned shall be RLIM\_SAVED\_CUR.
- 18233 When using the *setrlimit()* function, if the requested new limit is RLIM\_INFINITY, the new limit  
 18234 shall be “no limit”; otherwise, if the requested new limit is RLIM\_SAVED\_MAX, the new limit  
 18235 shall be the corresponding saved hard limit; otherwise, if the requested new limit is  
 18236 RLIM\_SAVED\_CUR, the new limit shall be the corresponding saved soft limit; otherwise, the  
 18237 new limit shall be the requested value. In addition, if the corresponding saved limit can be  
 18238 represented correctly in an object of type **rlim\_t** then it shall be overwritten with the new limit.
- 18239 The result of setting a limit to RLIM\_SAVED\_MAX or RLIM\_SAVED\_CUR is unspecified unless  
 18240 a previous call to *getrlimit()* returned that value as the soft or hard limit for the corresponding  
 18241 resource limit.
- 18242 The determination of whether a limit can be correctly represented in an object of type **rlim\_t** is  
 18243 implementation-defined. For example, some implementations permit a limit whose value is  
 18244 greater than RLIM\_INFINITY and others do not.
- 18245 The *exec* family of functions shall cause resource limits to be saved.
- 18246 **RETURN VALUE**
- 18247 Upon successful completion, *getrlimit()* and *setrlimit()* shall return 0. Otherwise, these functions  
 18248 shall return -1 and set *errno* to indicate the error.
- 18249 **ERRORS**
- 18250 The *getrlimit()* and *setrlimit()* functions shall fail if:
- 18251 [EINVAL] An invalid *resource* was specified; or in a *setrlimit()* call, the new *rlim\_cur*  
 18252 exceeds the new *rlim\_max*.
- 18253 [EPERM] The limit specified to *setrlimit()* would have raised the maximum limit value,  
 18254 and the calling process does not have appropriate privileges.
- 18255 The *setrlimit()* function may fail if:
- 18256 [EINVAL] The limit specified cannot be lowered because current usage is already higher  
 18257 than the limit.

18258 **EXAMPLES**

18259 None.

18260 **APPLICATION USAGE**18261 If a process attempts to set the hard limit or soft limit for RLIMIT\_NOFILE to less than the value  
18262 of `{_POSIX_OPEN_MAX}` from `<limits.h>`, unexpected behavior may occur. |18263 If a process attempts to set the hard limit or soft limit for RLIMIT\_NOFILE to less than the  
18264 highest currently open file descriptor +1, unexpected behavior may occur. |18265 **RATIONALE**

18266 None.

18267 **FUTURE DIRECTIONS**

18268 None.

18269 **SEE ALSO**18270 *exec*, *fork()*, *malloc()*, *open()*, *sigaltstack()*, *sysconf()*, *ulimit()*, the Base Definitions volume of  
18271 IEEE Std 1003.1-2001, `<stropts.h>`, `<sys/resource.h>`18272 **CHANGE HISTORY**

18273 First released in Issue 4, Version 2.

18274 **Issue 5**

18275 Moved from X/OPEN UNIX extension to BASE.

18276 An APPLICATION USAGE section is added.

18277 Large File Summit extensions are added.

18278 **Issue 6**18279 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/25 is applied, changing wording for  
18280 RLIMIT\_NOFILE in the DESCRIPTION related to functions that allocate a file descriptor failing  
18281 with [EMFILE]. Text is added to the APPLICATION USAGE section noting the consequences of  
18282 a process attempting to set the hard or soft limit for RLIMIT\_NOFILE less than the highest  
18283 currently open file descriptor +1. |

18284 **NAME**

18285           getrusage — get information about resource utilization

18286 **SYNOPSIS**

18287 XSI       #include &lt;sys/resource.h&gt;

18288       int getrusage(int *who*, struct rusage \**r\_usage*);

18289

18290 **DESCRIPTION**

18291       The *getrusage()* function shall provide measures of the resources used by the current process or  
 18292       its terminated and waited-for child processes. If the value of the *who* argument is  
 18293       RUSAGE\_SELF, information shall be returned about resources used by the current process. If the  
 18294       value of the *who* argument is RUSAGE\_CHILDREN, information shall be returned about  
 18295       resources used by the terminated and waited-for children of the current process. If the child is  
 18296       never waited for (for example, if the parent has SA\_NOCLDWAIT set or sets SIGCHLD to  
 18297       SIG\_IGN), the resource information for the child process is discarded and not included in the  
 18298       resource information provided by *getrusage()*.

18299       The *r\_usage* argument is a pointer to an object of type **struct rusage** in which the returned  
 18300       information is stored.

18301 **RETURN VALUE**

18302       Upon successful completion, *getrusage()* shall return 0; otherwise, -1 shall be returned and *errno*  
 18303       set to indicate the error.

18304 **ERRORS**18305       The *getrusage()* function shall fail if:18306       [EINVAL]       The value of the *who* argument is not valid.18307 **EXAMPLES**18308       **Using getrusage()**

18309       The following example returns information about the resources used by the current process.

```
18310 #include <sys/resource.h>
18311 ...
18312 int who = RUSAGE_SELF;
18313 struct rusage usage;
18314 int ret;
18315 ret = getrusage(who, &usage);
```

18316 **APPLICATION USAGE**

18317       None.

18318 **RATIONALE**

18319       None.

18320 **FUTURE DIRECTIONS**

18321       None.

18322 **SEE ALSO**

18323       *exit()*, *sigaction()*, *time()*, *times()*, *wait()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
 18324       <sys/resource.h>

18325 **CHANGE HISTORY**

18326 First released in Issue 4, Version 2.

18327 **Issue 5**

18328 Moved from X/OPEN UNIX extension to BASE.

18329 **NAME**

18330 gets — get a string from a stdin stream

18331 **SYNOPSIS**

18332 #include <stdio.h>

18333 char \*gets(char \*s);

18334 **DESCRIPTION**

18335 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
18336 conflict between the requirements described here and the ISO C standard is unintentional. This  
18337 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

18338 The *gets()* function shall read bytes from the standard input stream, *stdin*, into the array pointed  
18339 to by *s*, until a <newline> is read or an end-of-file condition is encountered. Any <newline> shall  
18340 be discarded and a null byte shall be placed immediately after the last byte read into the array.

18341 CX The *gets()* function may mark the *st\_atime* field of the file associated with *stream* for update. The  
18342 *st\_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,  
18343 *fread()*, *getc()*, *getchar()*, *gets()*, *fscanf()*, or *scanf()* using *stream* that returns data not supplied by  
18344 a prior call to *ungetc()*.

18345 **RETURN VALUE**

18346 Upon successful completion, *gets()* shall return *s*. If the stream is at end-of-file, the end-of-file  
18347 indicator for the stream shall be set and *gets()* shall return a null pointer. If a read error occurs,  
18348 CX the error indicator for the stream shall be set, *gets()* shall return a null pointer, and set *errno* to  
18349 indicate the error.

18350 **ERRORS**

18351 Refer to *fgetc()*.

18352 **EXAMPLES**

18353 None.

18354 **APPLICATION USAGE**

18355 Reading a line that overflows the array pointed to by *s* results in undefined behavior. The use of  
18356 *fgets()* is recommended.

18357 Since the user cannot specify the length of the buffer passed to *gets()*, use of this function is  
18358 discouraged. The length of the string read is unlimited. It is possible to overflow this buffer in  
18359 such a way as to cause applications to fail, or possible system security violations.

18360 It is recommended that the *fgets()* function should be used to read input lines.

18361 **RATIONALE**

18362 None.

18363 **FUTURE DIRECTIONS**

18364 None.

18365 **SEE ALSO**

18366 *feof()*, *ferror()*, *fgets()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

18367 **CHANGE HISTORY**

18368 First released in Issue 1. Derived from Issue 1 of the SVID.

18369 **Issue 6**

18370 Extensions beyond the ISO C standard are marked.

18371 **NAME**

18372        getservbyname, getservbyport, getservent — network services database functions

18373 **SYNOPSIS**

18374        #include <netdb.h>

18375        struct servent \*getservbyname(const char \*name, const char \*proto);

18376        struct servent \*getservbyport(int port, const char \*proto);

18377        struct servent \*getservent(void);

18378 **DESCRIPTION**

18379        Refer to *endservent()*.

18380 **NAME**

18381        *getsid* — get the process group ID of a session leader

18382 **SYNOPSIS**

18383 XSI        #include <unistd.h>

18384        pid\_t getsid(pid\_t pid);

18385

18386 **DESCRIPTION**

18387        The *getsid()* function shall obtain the process group ID of the process that is the session leader of  
18388        the process specified by *pid*. If *pid* is (**pid\_t**)0, it specifies the calling process.

18389 **RETURN VALUE**

18390        Upon successful completion, *getsid()* shall return the process group ID of the session leader of  
18391        the specified process. Otherwise, it shall return (**pid\_t**)-1 and set *errno* to indicate the error.

18392 **ERRORS**

18393        The *getsid()* function shall fail if:

18394        [EPERM]        The process specified by *pid* is not in the same session as the calling process,  
18395        and the implementation does not allow access to the process group ID of the  
18396        session leader of that process from the calling process.

18397        [ESRCH]        There is no process with a process ID equal to *pid*.

18398 **EXAMPLES**

18399        None.

18400 **APPLICATION USAGE**

18401        None.

18402 **RATIONALE**

18403        None.

18404 **FUTURE DIRECTIONS**

18405        None.

18406 **SEE ALSO**

18407        *exec*, *fork()*, *getpid()*, *getpgid()*, *setpgid()*, *setsid()*, the Base Definitions volume of  
18408        IEEE Std 1003.1-2001, <unistd.h>

18409 **CHANGE HISTORY**

18410        First released in Issue 4, Version 2.

18411 **Issue 5**

18412        Moved from X/OPEN UNIX extension to BASE.

18413 **NAME**

18414 getsockname — get the socket name

18415 **SYNOPSIS**

18416 #include &lt;sys/socket.h&gt;

18417 int getsockname(int *socket*, struct sockaddr \*restrict *address*,  
18418 socklen\_t \*restrict *address\_len*);18419 **DESCRIPTION**18420 The *getsockname()* function shall retrieve the locally-bound name of the specified socket, store  
18421 this address in the **sockaddr** structure pointed to by the *address* argument, and store the length of  
18422 this address in the object pointed to by the *address\_len* argument.18423 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,  
18424 the stored address shall be truncated.18425 If the socket has not been bound to a local name, the value stored in the object pointed to by  
18426 *address* is unspecified.18427 **RETURN VALUE**18428 Upon successful completion, 0 shall be returned, the *address* argument shall point to the address  
18429 of the socket, and the *address\_len* argument shall point to the length of the address. Otherwise, -1  
18430 shall be returned and *errno* set to indicate the error.18431 **ERRORS**18432 The *getsockname()* function shall fail if:18433 [EBADF] The *socket* argument is not a valid file descriptor.18434 [ENOTSOCK] The *socket* argument does not refer to a socket.

18435 [EOPNOTSUPP] The operation is not supported for this socket's protocol.

18436 The *getsockname()* function may fail if:

18437 [EINVAL] The socket has been shut down.

18438 [ENOBUFS] Insufficient resources were available in the system to complete the function.

18439 **EXAMPLES**

18440 None.

18441 **APPLICATION USAGE**

18442 None.

18443 **RATIONALE**

18444 None.

18445 **FUTURE DIRECTIONS**

18446 None.

18447 **SEE ALSO**18448 *accept()*, *bind()*, *getpeername()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
18449 <sys/socket.h>18450 **CHANGE HISTORY**

18451 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

18452 The **restrict** keyword is added to the *getsockname()* prototype for alignment with the  
18453 ISO/IEC 9899:1999 standard.

18454 **NAME**

18455 getsockopt — get the socket options

18456 **SYNOPSIS**

18457 #include &lt;sys/socket.h&gt;

```
18458 int getsockopt(int socket, int level, int option_name,
18459 void *restrict option_value, socklen_t *restrict option_len);
```

18460 **DESCRIPTION**18461 The *getsockopt()* function manipulates options associated with a socket.

18462 The *getsockopt()* function shall retrieve the value for the option specified by the *option\_name*  
 18463 argument for the socket specified by the *socket* argument. If the size of the option value is greater  
 18464 than *option\_len*, the value stored in the object pointed to by the *option\_value* argument shall be  
 18465 silently truncated. Otherwise, the object pointed to by the *option\_len* argument shall be modified  
 18466 to indicate the actual length of the value.

18467 The *level* argument specifies the protocol level at which the option resides. To retrieve options at  
 18468 the socket level, specify the *level* argument as SOL\_SOCKET. To retrieve options at other levels,  
 18469 supply the appropriate level identifier for the protocol controlling the option. For example, to  
 18470 indicate that an option is interpreted by the TCP (Transmission Control Protocol), set *level* to  
 18471 IPPROTO\_TCP as defined in the <netinet/in.h> header.

18472 The socket in use may require the process to have appropriate privileges to use the *getsockopt()*  
 18473 function.

18474 The *option\_name* argument specifies a single option to be retrieved. It can be one of the following  
 18475 values defined in <sys/socket.h>:

18476 SO\_DEBUG Reports whether debugging information is being recorded. This option  
 18477 shall store an **int** value. This is a Boolean option.

18478 SO\_ACCEPTCONN Reports whether socket listening is enabled. This option shall store an **int**  
 18479 value. This is a Boolean option.

18480 SO\_BROADCAST Reports whether transmission of broadcast messages is supported, if this  
 18481 is supported by the protocol. This option shall store an **int** value. This is a  
 18482 Boolean option.

18483 SO\_REUSEADDR Reports whether the rules used in validating addresses supplied to *bind()*  
 18484 should allow reuse of local addresses, if this is supported by the protocol.  
 18485 This option shall store an **int** value. This is a Boolean option.

18486 SO\_KEEPALIVE Reports whether connections are kept active with periodic transmission  
 18487 of messages, if this is supported by the protocol.

18488 If the connected socket fails to respond to these messages, the connection  
 18489 shall be broken and threads writing to that socket shall be notified with a  
 18490 SIGPIPE signal. This option shall store an **int** value. This is a Boolean  
 18491 option.

18492 SO\_LINGER Reports whether the socket lingers on *close()* if data is present. If  
 18493 SO\_LINGER is set, the system blocks the process during *close()* until it  
 18494 can transmit the data or until the end of the interval indicated by the  
 18495 *l\_linger* member, whichever comes first. If SO\_LINGER is not specified,  
 18496 and *close()* is issued, the system handles the call in a way that allows the  
 18497 process to continue as quickly as possible. This option shall store a **linger**  
 18498 structure.

|       |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 18499 | SO_OOINLINE  | Reports whether the socket leaves received out-of-band data (data marked urgent) inline. This option shall store an <b>int</b> value. This is a Boolean option.                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18500 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18501 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18502 | SO_SNDBUF    | Reports send buffer size information. This option shall store an <b>int</b> value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 18503 | SO_RCVBUF    | Reports receive buffer size information. This option shall store an <b>int</b> value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 18504 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18505 | SO_ERROR     | Reports information about error status and clears it. This option shall store an <b>int</b> value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 18506 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18507 | SO_TYPE      | Reports the socket type. This option shall store an <b>int</b> value. Socket types are described in Section 2.10.6 (on page 59).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 18508 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18509 | SO_DONTROUTE | Reports whether outgoing messages bypass the standard routing facilities. The destination shall be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. The effect, if any, of this option depends on what protocol is in use. This option shall store an <b>int</b> value. This is a Boolean option.                                                                                                                                                                                                                             |
| 18510 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18511 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18512 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18513 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18514 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18515 | SO_RCVLOWAT  | Reports the minimum number of bytes to process for socket input operations. The default value for SO_RCVLOWAT is 1. If SO_RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. (They may return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different from that returned; for example, out-of-band data.) This option shall store an <b>int</b> value. Note that not all implementations allow this option to be retrieved. |
| 18516 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18517 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18518 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18519 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18520 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18521 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18522 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18523 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18524 | SO_RCVTIMEO  | Reports the timeout value for input operations. This option shall store a <b>timeval</b> structure with the number of seconds and microseconds specifying the limit on how long to wait for an input operation to complete. If a receive operation has blocked for this much time without receiving additional data, it shall return with a partial count or <i>errno</i> set to [EAGAIN] or [EWOULDBLOCK] if no data was received. The default for this option is zero, which indicates that a receive operation shall not time out. Note that not all implementations allow this option to be retrieved.         |
| 18525 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18526 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18527 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18528 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18529 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18530 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18531 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18532 | SO_SNDLOWAT  | Reports the minimum number of bytes to process for socket output operations. Non-blocking output operations shall process no data if flow control does not allow the smaller of the send low water mark value or the entire request to be processed. This option shall store an <b>int</b> value. Note that not all implementations allow this option to be retrieved.                                                                                                                                                                                                                                             |
| 18533 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18534 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18535 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18536 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18537 | SO_SNDTIMEO  | Reports the timeout value specifying the amount of time that an output function blocks because flow control prevents data from being sent. If a send operation has blocked for this time, it shall return with a partial count or with <i>errno</i> set to [EAGAIN] or [EWOULDBLOCK] if no data was sent. The default for this option is zero, which indicates that a send operation shall not time out. The option shall store a <b>timeval</b> structure. Note that not all implementations allow this option to be retrieved.                                                                                   |
| 18538 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18539 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18540 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18541 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18542 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18543 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18544 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 18545 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|       |              | For Boolean options, a zero value indicates that the option is disabled and a non-zero value indicates that the option is enabled.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

18546 **RETURN VALUE**

18547       Upon successful completion, *getsockopt()* shall return 0; otherwise, -1 shall be returned and *errno*  
18548       set to indicate the error.

18549 **ERRORS**

18550       The *getsockopt()* function shall fail if:

18551       [EBADF]        The *socket* argument is not a valid file descriptor.

18552       [EINVAL]       The specified option is invalid at the specified socket level.

18553       [ENOPROTOOPT]

18554                The option is not supported by the protocol.

18555       [ENOTSOCK]   The *socket* argument does not refer to a socket.

18556       The *getsockopt()* function may fail if:

18557       [EACCES]       The calling process does not have the appropriate privileges.

18558       [EINVAL]       The socket has been shut down.

18559       [ENOBUFS]     Insufficient resources are available in the system to complete the function.

18560 **EXAMPLES**

18561       None.

18562 **APPLICATION USAGE**

18563       None.

18564 **RATIONALE**

18565       None.

18566 **FUTURE DIRECTIONS**

18567       None.

18568 **SEE ALSO**

18569       *bind()*, *close()*, *endprotoent()*, *setsockopt()*, *socket()*, the Base Definitions volume of  
18570       IEEE Std 1003.1-2001, <sys/socket.h>, <netinet/in.h>

18571 **CHANGE HISTORY**

18572       First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

18573       The **restrict** keyword is added to the *getsockopt()* prototype for alignment with the

18574       ISO/IEC 9899:1999 standard.

18575 **NAME**

18576       getsubopt — parse suboption arguments from a string

18577 **SYNOPSIS**

18578 XSI       #include &lt;stdlib.h&gt;

18579       int getsubopt(char \*\*optionp, char \* const \*keylistp, char \*\*valuep);

18580

18581 **DESCRIPTION**18582       The *getsubopt()* function shall parse suboption arguments in a flag argument. Such options often  
18583 result from the use of *getopt()*.18584       The *getsubopt()* argument *optionp* is a pointer to a pointer to the option argument string. The  
18585 suboption arguments shall be separated by commas and each may consist of either a single  
18586 token, or a token-value pair separated by an equal sign.18587       The *keylistp* argument shall be a pointer to a vector of strings. The end of the vector is identified  
18588 by a null pointer. Each entry in the vector is one of the possible tokens that might be found in  
18589 *\*optionp*. Since commas delimit suboption arguments in *optionp*, they should not appear in any of  
18590 the strings pointed to by *keylistp*. Similarly, because an equal sign separates a token from its  
18591 value, the application should not include an equal sign in any of the strings pointed to by  
18592 *keylistp*.18593       The *valuep* argument is the address of a value string pointer.18594       If a comma appears in *optionp*, it shall be interpreted as a suboption separator. After commas  
18595 have been processed, if there are one or more equal signs in a suboption string, the first equal  
18596 sign in any suboption string shall be interpreted as a separator between a token and a value.  
18597 Subsequent equal signs in a suboption string shall be interpreted as part of the value.18598       If the string at *\*optionp* contains only one suboption argument (equivalently, no commas),  
18599 *getsubopt()* shall update *\*optionp* to point to the null character at the end of the string. Otherwise,  
18600 it shall isolate the suboption argument by replacing the comma separator with a null character,  
18601 and shall update *\*optionp* to point to the start of the next suboption argument. If the suboption  
18602 argument has an associated value (equivalently, contains an equal sign), *getsubopt()* shall update  
18603 *\*valuep* to point to the value's first character. Otherwise, it shall set *\*valuep* to a null pointer. The  
18604 calling application may use this information to determine whether the presence or absence of a  
18605 value for the suboption is an error.18606       Additionally, when *getsubopt()* fails to match the suboption argument with a token in the *keylistp*  
18607 array, the calling application should decide if this is an error, or if the unrecognized option  
18608 should be processed in another way.18609 **RETURN VALUE**18610       The *getsubopt()* function shall return the index of the matched token string, or -1 if no token  
18611 strings were matched.18612 **ERRORS**

18613       No errors are defined.

18614 **EXAMPLES**

```
18615 #include <stdio.h>
18616 #include <stdlib.h>

18617 int do_all;
18618 const char *type;
18619 int read_size;
18620 int write_size;
18621 int read_only;

18622 enum
18623 {
18624 RO_OPTION = 0,
18625 RW_OPTION,
18626 READ_SIZE_OPTION,
18627 WRITE_SIZE_OPTION
18628 };

18629 const char *mount_opts[] =
18630 {
18631 [RO_OPTION] = "ro",
18632 [RW_OPTION] = "rw",
18633 [READ_SIZE_OPTION] = "rsize",
18634 [WRITE_SIZE_OPTION] = "wsize",
18635 NULL
18636 };

18637 int
18638 main(int argc, char *argv[])
18639 {
18640 char *subopts, *value;
18641 int opt;

18642 while ((opt = getopt(argc, argv, "at:o:")) != -1)
18643 switch(opt)
18644 {
18645 case 'a':
18646 do_all = 1;
18647 break;
18648 case 't':
18649 type = optarg;
18650 break;
18651 case 'o':
18652 subopts = optarg;
18653 while (*subopts != '\0')
18654 switch(getsubopt(&subopts, mount_opts, &value))
18655 {
18656 case RO_OPTION:
18657 read_only = 1;
18658 break;
18659 case RW_OPTION:
18660 read_only = 0;
18661 break;
18662 case READ_SIZE_OPTION:
```

```

18663 if (value == NULL)
18664 abort();
18665 read_size = atoi(value);
18666 break;
18667 case WRITE_SIZE_OPTION:
18668 if (value == NULL)
18669 abort();
18670 write_size = atoi(value);
18671 break;
18672 default:
18673 /* Unknown suboption. */
18674 printf("Unknown suboption '%s'\n", value);
18675 break;
18676 }
18677 break;
18678 default:
18679 abort();
18680 }
18681 /* Do the real work. */
18682 return 0;
18683 }

```

### 18684 Parsing Suboptions

18685 The following example uses the *getsubopt()* function to parse a *value* argument in the *optarg*  
 18686 external variable returned by a call to *getopt()*.

```

18687 #include <stdlib.h>
18688 ...
18689 char *tokens[] = {"HOME", "PATH", "LOGNAME", (char *) NULL };
18690 char *value;
18691 int opt, index;
18692 while ((opt = getopt(argc, argv, "e:")) != -1) {
18693 switch(opt) {
18694 case 'e' :
18695 while ((index = getsubopt(&optarg, tokens, &value)) != -1) {
18696 switch(index) {
18697 ...
18698 }
18699 break;
18700 ...
18701 }
18702 }
18703 ...

```

### 18704 APPLICATION USAGE

18705 None.

### 18706 RATIONALE

18707 None.

18708 **FUTURE DIRECTIONS**

18709 None.

18710 **SEE ALSO**18711 *getopt()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>18712 **CHANGE HISTORY**

18713 First released in Issue 4, Version 2.

18714 **Issue 5**

18715 Moved from X/OPEN UNIX extension to BASE.

18716 **Issue 6**18717 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/26 is applied, correcting an editorial error |  
18718 in the SYNOPSIS. |

18719 **NAME**

18720           gettimeofday — get the date and time

18721 **SYNOPSIS**

```
18722 xSI #include <sys/time.h>
```

```
18723 int gettimeofday(struct timeval *restrict tp, void *restrict tzp);
```

18724

18725 **DESCRIPTION**

18726       The *gettimeofday()* function shall obtain the current time, expressed as seconds and  
18727       microseconds since the Epoch, and store it in the **timeval** structure pointed to by *tp*. The  
18728       resolution of the system clock is unspecified.

18729       If *tzp* is not a null pointer, the behavior is unspecified.

18730 **RETURN VALUE**

18731       The *gettimeofday()* function shall return 0 and no value shall be reserved to indicate an error.

18732 **ERRORS**

18733       No errors are defined.

18734 **EXAMPLES**

18735       None.

18736 **APPLICATION USAGE**

18737       None.

18738 **RATIONALE**

18739       None.

18740 **FUTURE DIRECTIONS**

18741       None.

18742 **SEE ALSO**

18743       *ctime()*, *ftime()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/time.h>

18744 **CHANGE HISTORY**

18745       First released in Issue 4, Version 2.

18746 **Issue 5**

18747       Moved from X/OPEN UNIX extension to BASE.

18748 **Issue 6**

18749       The DESCRIPTION is updated to refer to “seconds since the Epoch” rather than “seconds since  
18750       00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time*  
18751       functions.

18752       The **restrict** keyword is added to the *gettimeofday()* prototype for alignment with the  
18753       ISO/IEC 9899:1999 standard.

18754 **NAME**

18755           getuid — get a real user ID

18756 **SYNOPSIS**

18757           #include <unistd.h>

18758           uid\_t getuid(void);

18759 **DESCRIPTION**

18760           The *getuid()* function shall return the real user ID of the calling process.

18761 **RETURN VALUE**

18762           The *getuid()* function shall always be successful and no return value is reserved to indicate the error.

18764 **ERRORS**

18765           No errors are defined.

18766 **EXAMPLES**18767           **Setting the Effective User ID to the Real User ID**

18768           The following example sets the effective user ID and the real user ID of the current process to the real user ID of the caller.

```
18770 #include <unistd.h>
18771 #include <sys/types.h>
18772 ...
18773 setreuid(getuid(), getuid());
18774 ...
```

18775 **APPLICATION USAGE**

18776           None.

18777 **RATIONALE**

18778           None.

18779 **FUTURE DIRECTIONS**

18780           None.

18781 **SEE ALSO**

18782           *getegid()*, *geteuid()*, *getgid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>

18784 **CHANGE HISTORY**

18785           First released in Issue 1. Derived from Issue 1 of the SVID.

18786 **Issue 6**

18787           In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

18788           The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 18790           • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- 18791
- 18792

18793 **NAME**

18794           getutxent, getutxid, getutxline — get user accounting database entries

18795 **SYNOPSIS**

18796 XSI       #include <utmpx.h>

18797           struct utmpx \*getutxent(void);

18798           struct utmpx \*getutxid(const struct utmpx \*id);

18799           struct utmpx \*getutxline(const struct utmpx \*line);

18800

18801 **DESCRIPTION**

18802           Refer to *endutxent()*.

18803 **NAME**

18804 getwc — get a wide character from a stream

18805 **SYNOPSIS**

18806 #include &lt;stdio.h&gt;

18807 #include &lt;wchar.h&gt;

18808 wint\_t getwc(FILE \*stream);

18809 **DESCRIPTION**

18810 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
18811 conflict between the requirements described here and the ISO C standard is unintentional. This  
18812 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

18813 The *getwc()* function shall be equivalent to *fgetwc()*, except that if it is implemented as a macro it  
18814 may evaluate *stream* more than once, so the argument should never be an expression with side  
18815 effects.

18816 **RETURN VALUE**18817 Refer to *fgetwc()*.18818 **ERRORS**18819 Refer to *fgetwc()*.18820 **EXAMPLES**

18821 None.

18822 **APPLICATION USAGE**

18823 Since it may be implemented as a macro, *getwc()* may treat incorrectly a *stream* argument with  
18824 side effects. In particular, *getwc(\*f++)* does not necessarily work as expected. Therefore, use of  
18825 this function is not recommended; *fgetwc()* should be used instead.

18826 **RATIONALE**

18827 None.

18828 **FUTURE DIRECTIONS**

18829 None.

18830 **SEE ALSO**18831 *fgetwc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>, <wchar.h>18832 **CHANGE HISTORY**

18833 First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working  
18834 draft.

18835 **Issue 5**

18836 The Optional Header (OH) marking is removed from &lt;stdio.h&gt;.

18837 **NAME**

18838           getwchar — get a wide character from a stdin stream

18839 **SYNOPSIS**

18840           #include <wchar.h>

18841           wint\_t getwchar(void);

18842 **DESCRIPTION**

18843 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
18844       conflict between the requirements described here and the ISO C standard is unintentional. This  
18845       volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

18846       The *getwchar()* function shall be equivalent to *getwc(stdin)*.

18847 **RETURN VALUE**

18848       Refer to *fgetwc()*.

18849 **ERRORS**

18850       Refer to *fgetwc()*.

18851 **EXAMPLES**

18852       None.

18853 **APPLICATION USAGE**

18854       If the **wint\_t** value returned by *getwchar()* is stored into a variable of type **wchar\_t** and then  
18855       compared against the **wint\_t** macro **WEOF**, the result may be incorrect. Only the **wint\_t** type is  
18856       guaranteed to be able to represent any wide character and **WEOF**.

18857 **RATIONALE**

18858       None.

18859 **FUTURE DIRECTIONS**

18860       None.

18861 **SEE ALSO**

18862       *fgetwc()*, *getwc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>

18863 **CHANGE HISTORY**

18864       First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working  
18865       draft.

18866 **NAME**18867 getwd — get the current working directory pathname (**LEGACY**)18868 **SYNOPSIS**18869 XSI `#include <unistd.h>`18870 `char *getwd(char *path_name);`

18871

18872 **DESCRIPTION**

18873 The *getwd()* function shall determine an absolute pathname of the current working directory of  
18874 the calling process, and copy a string containing that pathname into the array pointed to by the  
18875 *path\_name* argument.

18876 If the length of the pathname of the current working directory is greater than ( $\{\text{PATH\_MAX}\}+1$ )  
18877 including the null byte, *getwd()* shall fail and return a null pointer.

18878 **RETURN VALUE**

18879 Upon successful completion, a pointer to the string containing the absolute pathname of the  
18880 current working directory shall be returned. Otherwise, *getwd()* shall return a null pointer and  
18881 the contents of the array pointed to by *path\_name* are undefined.

18882 **ERRORS**

18883 No errors are defined.

18884 **EXAMPLES**

18885 None.

18886 **APPLICATION USAGE**

18887 For applications portability, the *getcwd()* function should be used to determine the current  
18888 working directory instead of *getwd()*.

18889 **RATIONALE**

18890 Since the user cannot specify the length of the buffer passed to *getwd()*, use of this function is  
18891 discouraged. The length of a pathname described in  $\{\text{PATH\_MAX}\}$  is file system-dependent and  
18892 may vary from one mount point to another, or might even be unlimited. It is possible to  
18893 overflow this buffer in such a way as to cause applications to fail, or possible system security  
18894 violations.

18895 It is recommended that the *getcwd()* function should be used to determine the current working  
18896 directory.

18897 **FUTURE DIRECTIONS**

18898 This function may be withdrawn in a future version.

18899 **SEE ALSO**18900 *getcwd()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<unistd.h>`18901 **CHANGE HISTORY**

18902 First released in Issue 4, Version 2.

18903 **Issue 5**

18904 Moved from X/OPEN UNIX extension to BASE.

18905 **Issue 6**

18906 This function is marked LEGACY.

## 18907 NAME

18908 glob, globfree — generate pathnames matching a pattern

## 18909 SYNOPSIS

18910 #include &lt;glob.h&gt;

```
18911 int glob(const char *restrict pattern, int flags,
18912 int(*errfunc)(const char *epath, int eerrno),
18913 glob_t *restrict pglob);
18914 void globfree(glob_t *pglob);
```

## 18915 DESCRIPTION

18916 The *glob()* function is a pathname generator that shall implement the rules defined in the Shell  
 18917 and Utilities volume of IEEE Std 1003.1-2001, Section 2.13, Pattern Matching Notation, with  
 18918 optional support for rule 3 in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section  
 18919 2.13.3, Patterns Used for Filename Expansion.

18920 The structure type **glob\_t** is defined in <glob.h> and includes at least the following members:

18921

18922

18923

18924

18925

| Member Type | Member Name     | Description                                            |
|-------------|-----------------|--------------------------------------------------------|
| size_t      | <i>gl_pathc</i> | Count of paths matched by <i>pattern</i> .             |
| char **     | <i>gl_pathv</i> | Pointer to a list of matched pathnames.                |
| size_t      | <i>gl_offs</i>  | Slots to reserve at the beginning of <i>gl_pathv</i> . |

18926

18927

18928

18929

18930

The argument *pattern* is a pointer to a pathname pattern to be expanded. The *glob()* function shall match all accessible pathnames against this pattern and develop a list of all pathnames that match. In order to have access to a pathname, *glob()* requires search permission on every component of a path except the last, and read permission on each directory of any filename component of *pattern* that contains any of the following special characters: '\*', '?', and '['.

18931

18932

18933

18934

18935

18936

18937

The *glob()* function shall store the number of matched pathnames into *pglob->gl\_pathc* and a pointer to a list of pointers to pathnames into *pglob->gl\_pathv*. The pathnames shall be in sort order as defined by the current setting of the *LC\_COLLATE* category; see the Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3.2, *LC\_COLLATE*. The first pointer after the last pathname shall be a null pointer. If the pattern does not match any pathnames, the returned number of matched paths is set to 0, and the contents of *pglob->gl\_pathv* are implementation-defined.

18938

18939

18940

It is the caller's responsibility to create the structure pointed to by *pglob*. The *glob()* function shall allocate other space as needed, including the memory pointed to by *gl\_pathv*. The *globfree()* function shall free any space associated with *pglob* from a previous call to *glob()*.

18941

18942

The *flags* argument is used to control the behavior of *glob()*. The value of *flags* is a bitwise-inclusive OR of zero or more of the following constants, which are defined in <glob.h>:

18943

**GLOB\_APPEND** Append pathnames generated to the ones from a previous call to *glob()*.

18944

18945

18946

18947

18948

**GLOB\_DOOFFS** Make use of *pglob->gl\_offs*. If this flag is set, *pglob->gl\_offs* is used to specify how many null pointers to add to the beginning of *pglob->gl\_pathv*. In other words, *pglob->gl\_pathv* shall point to *pglob->gl\_offs* null pointers, followed by *pglob->gl\_pathc* pathname pointers, followed by a null pointer.

18949

18950

**GLOB\_ERR** Cause *glob()* to return when it encounters a directory that it cannot open or read. Ordinarily, *glob()* continues to find matches.

|       |               |                                                                                                                                                                                                                                                                                                      |
|-------|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 18951 | GLOB_MARK     | Each pathname that is a directory that matches <i>pattern</i> shall have a slash appended.                                                                                                                                                                                                           |
| 18952 |               |                                                                                                                                                                                                                                                                                                      |
| 18953 | GLOB_NOCHECK  | Supports rule 3 in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.13.3, Patterns Used for Filename Expansion. If <i>pattern</i> does not match any pathname, then <i>glob()</i> shall return a list consisting of only <i>pattern</i> , and the number of matched pathnames is 1. |
| 18954 |               |                                                                                                                                                                                                                                                                                                      |
| 18955 |               |                                                                                                                                                                                                                                                                                                      |
| 18956 |               |                                                                                                                                                                                                                                                                                                      |
| 18957 | GLOB_NOESCAPE | Disable backslash escaping.                                                                                                                                                                                                                                                                          |
| 18958 | GLOB_NOSORT   | Ordinarily, <i>glob()</i> sorts the matching pathnames according to the current setting of the <i>LC_COLLATE</i> category; see the Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3.2, <i>LC_COLLATE</i> . When this flag is used, the order of pathnames returned is unspecified.       |
| 18959 |               |                                                                                                                                                                                                                                                                                                      |
| 18960 |               |                                                                                                                                                                                                                                                                                                      |
| 18961 |               |                                                                                                                                                                                                                                                                                                      |

18962 The GLOB\_APPEND flag can be used to append a new set of pathnames to those found in a  
 18963 previous call to *glob()*. The following rules apply to applications when two or more calls to  
 18964 *glob()* are made with the same value of *pglob* and without intervening calls to *globfree()*:

- 18965 1. The first such call shall not set GLOB\_APPEND. All subsequent calls shall set it.
- 18966 2. All the calls shall set GLOB\_DOOFFS, or all shall not set it.
- 18967 3. After the second call, *pglob->gl\_pathv* points to a list containing the following:
  - 18968 a. Zero or more null pointers, as specified by GLOB\_DOOFFS and *pglob->gl\_offs*.
  - 18969 b. Pointers to the pathnames that were in the *pglob->gl\_pathv* list before the call, in the  
 18970 same order as before.
  - 18971 c. Pointers to the new pathnames generated by the second call, in the specified order.
- 18972 4. The count returned in *pglob->gl\_pathc* shall be the total number of pathnames from the two  
 18973 calls.
- 18974 5. The application can change any of the fields after a call to *glob()*. If it does, the application  
 18975 shall reset them to the original value before a subsequent call, using the same *pglob* value,  
 18976 to *globfree()* or *glob()* with the GLOB\_APPEND flag.

18977 If, during the search, a directory is encountered that cannot be opened or read and *errfunc* is not  
 18978 a null pointer, *glob()* calls *(\*errfunc())* with two arguments:

- 18979 1. The *epath* argument is a pointer to the path that failed.
- 18980 2. The *errno* argument is the value of *errno* from the failure, as set by *opendir()*, *readdir()*, or  
 18981 *stat()*. (Other values may be used to report other errors not explicitly documented for  
 18982 those functions.)

18983 If *(\*errfunc())* is called and returns non-zero, or if the GLOB\_ERR flag is set in *flags*, *glob()* shall  
 18984 stop the scan and return GLOB\_ABORTED after setting *gl\_pathc* and *gl\_pathv* in *pglob* to reflect  
 18985 the paths already scanned. If GLOB\_ERR is not set and either *errfunc* is a null pointer or  
 18986 *(\*errfunc())* returns 0, the error shall be ignored.

18987 The *glob()* function shall not fail because of large files.

#### 18988 RETURN VALUE

18989 Upon successful completion, *glob()* shall return 0. The argument *pglob->gl\_pathc* shall return the  
 18990 number of matched pathnames and the argument *pglob->gl\_pathv* shall contain a pointer to a  
 18991 null-terminated list of matched and sorted pathnames. However, if *pglob->gl\_pathc* is 0, the  
 18992 content of *pglob->gl\_pathv* is undefined.

18993 The *globfree()* function shall not return a value.

18994 If *glob()* terminates due to an error, it shall return one of the non-zero constants defined in  
18995 `<glob.h>`. The arguments *pglob->gl\_pathc* and *pglob->gl\_pathv* are still set as defined above.

**18996 ERRORS**

18997 The *glob()* function shall fail and return the corresponding value if:

|       |              |                                                                        |
|-------|--------------|------------------------------------------------------------------------|
| 18998 | GLOB_ABORTED | The scan was stopped because GLOB_ERR was set or ( <i>*errfunc()</i> ) |
| 18999 |              | returned non-zero.                                                     |
| 19000 | GLOB_NOMATCH | The pattern does not match any existing pathname, and                  |
| 19001 |              | GLOB_NOCHECK was not set in flags.                                     |
| 19002 | GLOB_NOSPACE | An attempt to allocate memory failed.                                  |

**19003 EXAMPLES**

19004 One use of the GLOB\_DOOFFS flag is by applications that build an argument list for use with  
19005 *execv()*, *execve()*, or *execvp()*. Suppose, for example, that an application wants to do the  
19006 equivalent of:

```
19007 ls -l *.c
```

19008 but for some reason:

```
19009 system("ls -l *.c")
```

19010 is not acceptable. The application could obtain approximately the same result using the  
19011 sequence:

```
19012 globbuf.gl_offs = 2;
19013 glob("*.c", GLOB_DOOFFS, NULL, &globbuf);
19014 globbuf.gl_pathv[0] = "ls";
19015 globbuf.gl_pathv[1] = "-l";
19016 execvp("ls", &globbuf.gl_pathv[0]);
```

19017 Using the same example:

```
19018 ls -l *.c *.h
```

19019 could be approximately simulated using GLOB\_APPEND as follows:

```
19020 globbuf.gl_offs = 2;
19021 glob("*.c", GLOB_DOOFFS, NULL, &globbuf);
19022 glob("*.h", GLOB_DOOFFS|GLOB_APPEND, NULL, &globbuf);
19023 ...
```

**19024 APPLICATION USAGE**

19025 This function is not provided for the purpose of enabling utilities to perform pathname  
19026 expansion on their arguments, as this operation is performed by the shell, and utilities are  
19027 explicitly not expected to redo this. Instead, it is provided for applications that need to do  
19028 pathname expansion on strings obtained from other sources, such as a pattern typed by a user or  
19029 read from a file.

19030 If a utility needs to see if a pathname matches a given pattern, it can use *fnmatch()*.

19031 Note that *gl\_pathc* and *gl\_pathv* have meaning even if *glob()* fails. This allows *glob()* to report  
19032 partial results in the event of an error. However, if *gl\_pathc* is 0, *gl\_pathv* is unspecified even if  
19033 *glob()* did not return an error.

19034 The GLOB\_NOCHECK option could be used when an application wants to expand a pathname  
19035 if wildcards are specified, but wants to treat the pattern as just a string otherwise. The *sh* utility

19036 might use this for option-arguments, for example.

19037 The new pathnames generated by a subsequent call with GLOB\_APPEND are not sorted  
 19038 together with the previous pathnames. This mirrors the way that the shell handles pathname  
 19039 expansion when multiple expansions are done on a command line.

19040 Applications that need tilde and parameter expansion should use *wordexp()*.

#### 19041 RATIONALE

19042 It was claimed that the GLOB\_DOOFFS flag is unnecessary because it could be simulated using:

```
19043 new = (char **)malloc((n + pglob->gl_pathc + 1)
19044 * sizeof(char *));
19045 (void) memcpy(new+n, pglob->gl_pathv,
19046 pglob->gl_pathc * sizeof(char *));
19047 (void) memset(new, 0, n * sizeof(char *));
19048 free(pglob->gl_pathv);
19049 pglob->gl_pathv = new;
```

19050 However, this assumes that the memory pointed to by *gl\_pathv* is a block that was separately  
 19051 created using *malloc()*. This is not necessarily the case. An application should make no  
 19052 assumptions about how the memory referenced by fields in *pglob* was allocated. It might have  
 19053 been obtained from *malloc()* in a large chunk and then carved up within *glob()*, or it might have  
 19054 been created using a different memory allocator. It is not the intent of the standard developers to  
 19055 specify or imply how the memory used by *glob()* is managed.

19056 The GLOB\_APPEND flag would be used when an application wants to expand several different  
 19057 patterns into a single list.

#### 19058 FUTURE DIRECTIONS

19059 None.

#### 19060 SEE ALSO

19061 *exec*, *fnmatch()*, *opendir()*, *readdir()*, *stat()*, *wordexp()*, the Base Definitions volume of  
 19062 IEEE Std 1003.1-2001, <**glob.h**>, the Shell and Utilities volume of IEEE Std 1003.1-2001

#### 19063 CHANGE HISTORY

19064 First released in Issue 4. Derived from the ISO POSIX-2 standard.

#### 19065 Issue 5

19066 Moved from POSIX2 C-language Binding to BASE.

#### 19067 Issue 6

19068 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

19069 The **restrict** keyword is added to the *glob()* prototype for alignment with the ISO/IEC 9899:1999  
 19070 standard.

## 19071 NAME

19072 gmtime, gmtime\_r — convert a time value to a broken-down UTC time

## 19073 SYNOPSIS

19074 #include <time.h>

19075 struct tm \*gmtime(const time\_t \*timer);

19076 TSF struct tm \*gmtime\_r(const time\_t \*restrict timer,

19077 struct tm \*restrict result);

19078

## 19079 DESCRIPTION

19080 CX For *gmtime()*: The functionality described on this reference page is aligned with the ISO C  
19081 standard. Any conflict between the requirements described here and the ISO C standard is  
19082 unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

19083 The *gmtime()* function shall convert the time in seconds since the Epoch pointed to by *timer* into  
19084 a broken-down time, expressed as Coordinated Universal Time (UTC).

19085 CX The relationship between a time in seconds since the Epoch used as an argument to *gmtime()*  
19086 and the **tm** structure (defined in the <time.h> header) is that the result shall be as specified in the  
19087 expression given in the definition of seconds since the Epoch (see the Base Definitions volume of  
19088 IEEE Std 1003.1-2001, Section 4.14, Seconds Since the Epoch), where the names in the structure  
19089 and in the expression correspond.

19090 TSF The same relationship shall apply for *gmtime\_r()*.

19091 CX The *gmtime()* function need not be reentrant. A function that is not required to be reentrant is not  
19092 required to be thread-safe.

19093 The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static  
19094 objects: a broken-down time structure and an array of type **char**. Execution of any of the  
19095 functions may overwrite the information returned in either of these objects by any of the other  
19096 functions.

19097 TSF The *gmtime\_r()* function shall convert the time in seconds since the Epoch pointed to by *timer*  
19098 into a broken-down time expressed as Coordinated Universal Time (UTC). The broken-down  
19099 time is stored in the structure referred to by *result*. The *gmtime\_r()* function shall also return the  
19100 address of the same structure.

## 19101 RETURN VALUE

19102 Upon successful completion, the *gmtime()* function shall return a pointer to a **struct tm**. If an  
19103 CX error is detected, *gmtime()* shall return a null pointer and set *errno* to indicate the error.

19104 TSF Upon successful completion, *gmtime\_r()* shall return the address of the structure pointed to by  
19105 the argument *result*. If an error is detected, *gmtime\_r()* shall return a null pointer.

## 19106 ERRORS

19107 The *gmtime()* function shall fail if:

19108 CX [EOVERFLOW] The result cannot be represented.

19109 **EXAMPLES**

19110 None.

19111 **APPLICATION USAGE**

19112 The *gmtime\_r()* function is thread-safe and returns values in a user-supplied buffer instead of  
19113 possibly using a static data area that may be overwritten by each call.

19114 **RATIONALE**

19115 None.

19116 **FUTURE DIRECTIONS**

19117 None.

19118 **SEE ALSO**

19119 *asctime()*, *clock()*, *ctime()*, *difftime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*,  
19120 the Base Definitions volume of IEEE Std 1003.1-2001, <**time.h**>

19121 **CHANGE HISTORY**

19122 First released in Issue 1. Derived from Issue 1 of the SVID.

19123 **Issue 5**

19124 A note indicating that the *gmtime()* function need not be reentrant is added to the  
19125 DESCRIPTION.

19126 The *gmtime\_r()* function is included for alignment with the POSIX Threads Extension.19127 **Issue 6**19128 The *gmtime\_r()* function is marked as part of the Thread-Safe Functions option.

19129 Extensions beyond the ISO C standard are marked.

19130 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
19131 its avoidance of possibly using a static data area.

19132 The **restrict** keyword is added to the *gmtime\_r()* prototype for alignment with the  
19133 ISO/IEC 9899:1999 standard.

19134 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/27 is applied, adding the [Eoverflow]  
19135 error. |

19136 **NAME**

19137 grantpt — grant access to the slave pseudo-terminal device

19138 **SYNOPSIS**19139 XSI `#include <stdlib.h>`19140 `int grantpt(int fildev);`

19141

19142 **DESCRIPTION**

19143 The *grantpt()* function shall change the mode and ownership of the slave pseudo-terminal  
 19144 device associated with its master pseudo-terminal counterpart. The *fildev* argument is a file  
 19145 descriptor that refers to a master pseudo-terminal device. The user ID of the slave shall be set to  
 19146 the real UID of the calling process and the group ID shall be set to an unspecified group ID. The  
 19147 permission mode of the slave pseudo-terminal shall be set to readable and writable by the  
 19148 owner, and writable by the group.

19149 The behavior of the *grantpt()* function is unspecified if the application has installed a signal  
 19150 handler to catch SIGCHLD signals.

19151 **RETURN VALUE**

19152 Upon successful completion, *grantpt()* shall return 0; otherwise, it shall return -1 and set *errno* to  
 19153 indicate the error.

19154 **ERRORS**19155 The *grantpt()* function may fail if:

- |       |          |                                                                                    |
|-------|----------|------------------------------------------------------------------------------------|
| 19156 | [EBADF]  | The <i>fildev</i> argument is not a valid open file descriptor.                    |
| 19157 | [EINVAL] | The <i>fildev</i> argument is not associated with a master pseudo-terminal device. |
| 19158 | [EACCES] | The corresponding slave pseudo-terminal device could not be accessed.              |

19159 **EXAMPLES**

19160 None.

19161 **APPLICATION USAGE**

19162 None.

19163 **RATIONALE**

19164 None.

19165 **FUTURE DIRECTIONS**

19166 None.

19167 **SEE ALSO**19168 *open()*, *ptsname()*, *unlockpt()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>19169 **CHANGE HISTORY**

19170 First released in Issue 4, Version 2.

19171 **Issue 5**

19172 Moved from X/OPEN UNIX extension to BASE.

19173 The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

19174 **NAME**

19175 h\_errno — error return value for network database operations

19176 **SYNOPSIS**

```
19177 OB #include <netdb.h>
```

19178

19179 **DESCRIPTION**

19180 This method of returning errors is used only in connection with obsolescent functions.

19181 The <netdb.h> header provides a declaration of *h\_errno* as a modifiable lvalue of type **int**.

19182 It is unspecified whether *h\_errno* is a macro or an identifier declared with external linkage. If a  
19183 macro definition is suppressed in order to access an actual object, or a program defines an  
19184 identifier with the name *h\_errno*, the behavior is undefined.

19185 **RETURN VALUE**

19186 None.

19187 **ERRORS**

19188 No errors are defined.

19189 **EXAMPLES**

19190 None.

19191 **APPLICATION USAGE**

19192 Applications should obtain the definition of *h\_errno* by the inclusion of the <netdb.h> header.

19193 **RATIONALE**

19194 None.

19195 **FUTURE DIRECTIONS**

19196 *h\_errno* may be withdrawn in a future version.

19197 **SEE ALSO**

19198 *endhostent()*, *errno*, the Base Definitions volume of IEEE Std 1003.1-2001, <netdb.h>

19199 **CHANGE HISTORY**

19200 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

## 19201 NAME

19202 hcreate, hdestroy, hsearch — manage hash search table

## 19203 SYNOPSIS

```
19204 xSI #include <search.h>
19205
19205 int hcreate(size_t nel);
19206 void hdestroy(void);
19207 ENTRY *hsearch(ENTRY item, ACTION action);
19208
```

## 19209 DESCRIPTION

19210 The *hcreate()*, *hdestroy()*, and *hsearch()* functions shall manage hash search tables.

19211 The *hcreate()* function shall allocate sufficient space for the table, and the application shall ensure it is called before *hsearch()* is used. The *nel* argument is an estimate of the maximum number of entries that the table shall contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favorable circumstances.

19215 The *hdestroy()* function shall dispose of the search table, and may be followed by another call to *hcreate()*. After the call to *hdestroy()*, the data can no longer be considered accessible.

19217 The *hsearch()* function is a hash-table search routine. It shall return a pointer into a hash table indicating the location at which an entry can be found. The *item* argument is a structure of type **ENTRY** (defined in the *<search.h>* header) containing two pointers: *item.key* points to the comparison key (a **char** \*), and *item.data* (a **void** \*) points to any other data to be associated with that key. The comparison function used by *hsearch()* is *strcmp()*. The *action* argument is a member of an enumeration type **ACTION** indicating the disposition of the entry if it cannot be found in the table. **ENTER** indicates that the item should be inserted in the table at an appropriate point. **FIND** indicates that no entry should be made. Unsuccessful resolution is indicated by the return of a null pointer.

19226 These functions need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

## 19228 RETURN VALUE

19229 The *hcreate()* function shall return 0 if it cannot allocate sufficient space for the table; otherwise, it shall return non-zero.

19231 The *hdestroy()* function shall not return a value.

19232 The *hsearch()* function shall return a null pointer if either the action is **FIND** and the item could not be found or the action is **ENTER** and the table is full.

## 19234 ERRORS

19235 The *hcreate()* and *hsearch()* functions may fail if:

19236 [ENOMEM] Insufficient storage space is available.

## 19237 EXAMPLES

19238 The following example reads in strings followed by two numbers and stores them in a hash table, discarding duplicates. It then reads in strings and finds the matching entry in the hash table and prints it out.

```
19241 #include <stdio.h>
19242 #include <search.h>
19243 #include <string.h>
19244 struct info { /* This is the info stored in the table */
19245 int age, room; /* other than the key. */
```

```

19246 };
19247 #define NUM_EMPL 5000 /* # of elements in search table. */
19248 int main(void)
19249 {
19250 char string_space[NUM_EMPL*20]; /* Space to store strings. */
19251 struct info info_space[NUM_EMPL]; /* Space to store employee info. */
19252 char *str_ptr = string_space; /* Next space in string_space. */
19253 struct info *info_ptr = info_space;
19254 /* Next space in info_space. */
19255 ENTRY item;
19256 ENTRY *found_item; /* Name to look for in table. */
19257 char name_to_find[30];
19258
19259 int i = 0;
19260
19261 /* Create table; no error checking is performed. */
19262 (void) hcreate(NUM_EMPL);
19263 while (scanf("%s%d%d", str_ptr, &info_ptr->age,
19264 &info_ptr->room) != EOF && i++ < NUM_EMPL) {
19265 /* Put information in structure, and structure in item. */
19266 item.key = str_ptr;
19267 item.data = info_ptr;
19268 str_ptr += strlen(str_ptr) + 1;
19269 info_ptr++;
19270
19271 /* Put item into table. */
19272 (void) hsearch(item, ENTER);
19273 }
19274
19275 /* Access table. */
19276 item.key = name_to_find;
19277 while (scanf("%s", item.key) != EOF) {
19278 if ((found_item = hsearch(item, FIND)) != NULL) {
19279 /* If item is in the table. */
19280 (void)printf("found %s, age = %d, room = %d\n",
19281 found_item->key,
19282 ((struct info *)found_item->data)->age,
19283 ((struct info *)found_item->data)->room);
19284 } else
19285 (void)printf("no such employee %s\n", name_to_find);
19286 }
19287 return 0;
19288 }

```

**19285 APPLICATION USAGE**

19286 The *hcreate()* and *hsearch()* functions may use *malloc()* to allocate space.

**19287 RATIONALE**

19288 None.

19289 **FUTURE DIRECTIONS**

19290           None.

19291 **SEE ALSO**

19292           *bsearch()*, *lsearch()*, *malloc()*, *strcmp()*, *tsearch()*, the Base Definitions volume of  
19293           IEEE Std 1003.1-2001, <**search.h**>

19294 **CHANGE HISTORY**

19295           First released in Issue 1. Derived from Issue 1 of the SVID.

19296 **Issue 6**

19297           The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

19298           A note indicating that this function need not be reentrant is added to the DESCRIPTION.

19299 **NAME**

19300 htonl, htons, ntohl, ntohs — convert values between host and network byte order

19301 **SYNOPSIS**

19302 #include <arpa/inet.h>

19303 uint32\_t htonl(uint32\_t *hostlong*);  
19304 uint16\_t htons(uint16\_t *hostshort*);  
19305 uint32\_t ntohl(uint32\_t *netlong*);  
19306 uint16\_t ntohs(uint16\_t *netshort*);

19307 **DESCRIPTION**

19308 These functions shall convert 16-bit and 32-bit quantities between network byte order and host  
19309 byte order.

19310 On some implementations, these functions are defined as macros.

19311 The **uint32\_t** and **uint16\_t** types are defined in <inttypes.h>.

19312 **RETURN VALUE**

19313 The *htonl()* and *htons()* functions shall return the argument value converted from host to  
19314 network byte order.

19315 The *ntohl()* and *ntohs()* functions shall return the argument value converted from network to  
19316 host byte order.

19317 **ERRORS**

19318 No errors are defined.

19319 **EXAMPLES**

19320 None.

19321 **APPLICATION USAGE**

19322 These functions are most often used in conjunction with IPv4 addresses and ports as returned by  
19323 *gethostent()* and *getservent()*.

19324 **RATIONALE**

19325 None.

19326 **FUTURE DIRECTIONS**

19327 None.

19328 **SEE ALSO**

19329 *endhostent()*, *endservent()*, the Base Definitions volume of IEEE Std 1003.1-2001, <inttypes.h>,  
19330 <arpa/inet.h>

19331 **CHANGE HISTORY**

19332 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

## 19333 NAME

19334 hypot, hypotf, hypotl — Euclidean distance function

## 19335 SYNOPSIS

19336 #include &lt;math.h&gt;

19337 double hypot(double x, double y);

19338 float hypotf(float x, float y);

19339 long double hypotl(long double x, long double y);

## 19340 DESCRIPTION

19341 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 19342 conflict between the requirements described here and the ISO C standard is unintentional. This  
 19343 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

19344 These functions shall compute the value of the square root of  $x^2+y^2$  without undue overflow or  
 19345 underflow.

19346 An application wishing to check for error situations should set *errno* to zero and call  
 19347 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 19348 *fetetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 19349 zero, an error has occurred.

## 19350 RETURN VALUE

19351 Upon successful completion, these functions shall return the length of the hypotenuse of a  
 19352 right-angled triangle with sides of length *x* and *y*.

19353 If the correct value would cause overflow, a range error shall occur and *hypot()*, *hypotf()*, and  
 19354 *hypotl()* shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL,  
 19355 respectively.

19356 MX If *x* or *y* is  $\pm\text{Inf}$ ,  $+\text{Inf}$  shall be returned (even if one of *x* or *y* is NaN).

19357 If *x* or *y* is NaN, and the other is not  $\pm\text{Inf}$ , a NaN shall be returned.

19358 If both arguments are subnormal and the correct result is subnormal, a range error may occur  
 19359 and the correct result is returned.

## 19360 ERRORS

19361 These functions shall fail if:

19362 Range Error The result overflows.

19363 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 19364 then *errno* shall be set to [ERANGE]. If the integer expression  
 19365 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow  
 19366 floating-point exception shall be raised.

19367 These functions may fail if:

19368 MX Range Error The result underflows.

19369 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 19370 then *errno* shall be set to [ERANGE]. If the integer expression  
 19371 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow  
 19372 floating-point exception shall be raised.

19373 **EXAMPLES**

19374 None.

19375 **APPLICATION USAGE**19376 *hypot(x,y)*, *hypot(y,x)*, and *hypot(x, -y)* are equivalent.19377 *hypot(x, ±0)* is equivalent to *fabs(x)*.19378 Underflow only happens when both *x* and *y* are subnormal and the (inexact) result is also  
19379 subnormal.19380 These functions take precautions against overflow during intermediate steps of the  
19381 computation.19382 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
19383 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.19384 **RATIONALE**

19385 None.

19386 **FUTURE DIRECTIONS**

19387 None.

19388 **SEE ALSO**19389 *feclearexcept()*, *fetestexcept()*, *isnan()*, *sqrt()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
19390 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>19391 **CHANGE HISTORY**

19392 First released in Issue 1. Derived from Issue 1 of the SVID.

19393 **Issue 5**19394 The DESCRIPTION is updated to indicate how an application should check for an error. This  
19395 text was previously published in the APPLICATION USAGE section.19396 **Issue 6**19397 The *hypot()* function is no longer marked as an extension.19398 The *hypotf()* and *hypotl()* functions are added for alignment with the ISO/IEC 9899:1999  
19399 standard.19400 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
19401 revised to align with the ISO/IEC 9899:1999 standard.19402 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
19403 marked.

## 19404 NAME

19405 iconv — codeset conversion function

## 19406 SYNOPSIS

19407 XSI 

```
#include <iconv.h>
```

```
19408 size_t iconv(iconv_t cd, char **restrict inbuf,
19409 size_t *restrict inbytesleft, char **restrict outbuf,
19410 size_t *restrict outbytesleft);
```

19411

## 19412 DESCRIPTION

19413 The *iconv()* function shall convert the sequence of characters from one codeset, in the array  
 19414 specified by *inbuf*, into a sequence of corresponding characters in another codeset, in the array  
 19415 specified by *outbuf*. The codesets are those specified in the *iconv\_open()* call that returned the  
 19416 conversion descriptor, *cd*. The *inbuf* argument points to a variable that points to the first  
 19417 character in the input buffer and *inbytesleft* indicates the number of bytes to the end of the buffer  
 19418 to be converted. The *outbuf* argument points to a variable that points to the first available byte in  
 19419 the output buffer and *outbytesleft* indicates the number of the available bytes to the end of the  
 19420 buffer.

19421 For state-dependent encodings, the conversion descriptor *cd* is placed into its initial shift state by  
 19422 a call for which *inbuf* is a null pointer, or for which *inbuf* points to a null pointer. When *iconv()* is  
 19423 called in this way, and if *outbuf* is not a null pointer or a pointer to a null pointer, and *outbytesleft*  
 19424 points to a positive value, *iconv()* shall place, into the output buffer, the byte sequence to change  
 19425 the output buffer to its initial shift state. If the output buffer is not large enough to hold the  
 19426 entire reset sequence, *iconv()* shall fail and set *errno* to [E2BIG]. Subsequent calls with *inbuf* as  
 19427 other than a null pointer or a pointer to a null pointer cause the conversion to take place from  
 19428 the current state of the conversion descriptor.

19429 If a sequence of input bytes does not form a valid character in the specified codeset, conversion  
 19430 shall stop after the previous successfully converted character. If the input buffer ends with an  
 19431 incomplete character or shift sequence, conversion shall stop after the previous successfully  
 19432 converted bytes. If the output buffer is not large enough to hold the entire converted input,  
 19433 conversion shall stop just prior to the input bytes that would cause the output buffer to  
 19434 overflow. The variable pointed to by *inbuf* shall be updated to point to the byte following the last  
 19435 byte successfully used in the conversion. The value pointed to by *inbytesleft* shall be  
 19436 decremented to reflect the number of bytes still not converted in the input buffer. The variable  
 19437 pointed to by *outbuf* shall be updated to point to the byte following the last byte of converted  
 19438 output data. The value pointed to by *outbytesleft* shall be decremented to reflect the number of  
 19439 bytes still available in the output buffer. For state-dependent encodings, the conversion  
 19440 descriptor shall be updated to reflect the shift state in effect at the end of the last successfully  
 19441 converted byte sequence.

19442 If *iconv()* encounters a character in the input buffer that is valid, but for which an identical  
 19443 character does not exist in the target codeset, *iconv()* shall perform an implementation-defined  
 19444 conversion on this character.

## 19445 RETURN VALUE

19446 The *iconv()* function shall update the variables pointed to by the arguments to reflect the extent  
 19447 of the conversion and return the number of non-identical conversions performed. If the entire  
 19448 string in the input buffer is converted, the value pointed to by *inbytesleft* shall be 0. If the input  
 19449 conversion is stopped due to any conditions mentioned above, the value pointed to by *inbytesleft*  
 19450 shall be non-zero and *errno* shall be set to indicate the condition. If an error occurs, *iconv()* shall  
 19451 return (*size\_t*)-1 and set *errno* to indicate the error.

19452 **ERRORS**19453 The *iconv()* function shall fail if:19454 [EILSEQ] Input conversion stopped due to an input byte that does not belong to the  
19455 input codeset.

19456 [E2BIG] Input conversion stopped due to lack of space in the output buffer.

19457 [EINVAL] Input conversion stopped due to an incomplete character or shift sequence at  
19458 the end of the input buffer.19459 The *iconv()* function may fail if:19460 [EBADF] The *cd* argument is not a valid open conversion descriptor.19461 **EXAMPLES**

19462 None.

19463 **APPLICATION USAGE**

19464 The *inbuf* argument indirectly points to the memory area which contains the conversion input  
19465 data. The *outbuf* argument indirectly points to the memory area which is to contain the result of  
19466 the conversion. The objects indirectly pointed to by *inbuf* and *outbuf* are not restricted to  
19467 containing data that is directly representable in the ISO C standard language **char** data type. The  
19468 type of *inbuf* and *outbuf*, **char \*\***, does not imply that the objects pointed to are interpreted as  
19469 null-terminated C strings or arrays of characters. Any interpretation of a byte sequence that  
19470 represents a character in a given character set encoding scheme is done internally within the  
19471 codeset converters. For example, the area pointed to indirectly by *inbuf* and/or *outbuf* can  
19472 contain all zero octets that are not interpreted as string terminators but as coded character data  
19473 according to the respective codeset encoding scheme. The type of the data (**char**, **short**, **long**, and  
19474 so on) read or stored in the objects is not specified, but may be inferred for both the input and  
19475 output data by the converters determined by the *fromcode* and *toctype* arguments of *iconv\_open()*.

19476 Regardless of the data type inferred by the converter, the size of the remaining space in both  
19477 input and output objects (the *inbytesleft* and *outbytesleft* arguments) is always measured in bytes.

19478 For implementations that support the conversion of state-dependent encodings, the conversion  
19479 descriptor must be able to accurately reflect the shift-state in effect at the end of the last  
19480 successful conversion. It is not required that the conversion descriptor itself be updated, which  
19481 would require it to be a pointer type. Thus, implementations are free to implement the  
19482 descriptor as a handle (other than a pointer type) by which the conversion information can be  
19483 accessed and updated.

19484 **RATIONALE**

19485 None.

19486 **FUTURE DIRECTIONS**

19487 None.

19488 **SEE ALSO**19489 *iconv\_open()*, *iconv\_close()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**iconv.h**>19490 **CHANGE HISTORY**

19491 First released in Issue 4. Derived from the HP-UX Manual.

19492 **Issue 6**19493 The SYNOPSIS has been corrected to align with the <**iconv.h**> reference page.

19494 The **restrict** keyword is added to the *iconv()* prototype for alignment with the  
19495 ISO/IEC 9899:1999 standard.

19496 **NAME**

19497 iconv\_close — codeset conversion deallocation function

19498 **SYNOPSIS**19499 XSI `#include <iconv.h>`19500 `int iconv_close(iconv_t cd);`

19501

19502 **DESCRIPTION**19503 The *iconv\_close()* function shall deallocate the conversion descriptor *cd* and all other associated  
19504 resources allocated by *iconv\_open()*.19505 If a file descriptor is used to implement the type **iconv\_t**, that file descriptor shall be closed.19506 **RETURN VALUE**19507 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
19508 indicate the error.19509 **ERRORS**19510 The *iconv\_close()* function may fail if:

19511 [EBADF] The conversion descriptor is invalid.

19512 **EXAMPLES**

19513 None.

19514 **APPLICATION USAGE**

19515 None.

19516 **RATIONALE**

19517 None.

19518 **FUTURE DIRECTIONS**

19519 None.

19520 **SEE ALSO**19521 *iconv()*, *iconv\_open()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**iconv.h**>19522 **CHANGE HISTORY**

19523 First released in Issue 4. Derived from the HP-UX Manual.

19524 **NAME**

19525 iconv\_open — codeset conversion allocation function

19526 **SYNOPSIS**19527 XSI 

```
#include <iconv.h>
```

19528 

```
iconv_t iconv_open(const char *tocode, const char *fromcode);
```

19529

19530 **DESCRIPTION**

19531 The *iconv\_open()* function shall return a conversion descriptor that describes a conversion from  
 19532 the codeset specified by the string pointed to by the *fromcode* argument to the codeset specified  
 19533 by the string pointed to by the *tocode* argument. For state-dependent encodings, the conversion  
 19534 descriptor shall be in a codeset-dependent initial shift state, ready for immediate use with  
 19535 *iconv()*.

19536 Settings of *fromcode* and *tocode* and their permitted combinations are implementation-defined.

19537 A conversion descriptor shall remain valid until it is closed by *iconv\_close()* or an implicit close.

19538 If a file descriptor is used to implement conversion descriptors, the FD\_CLOEXEC flag shall be  
 19539 set; see <fcntl.h>.

19540 **RETURN VALUE**

19541 Upon successful completion, *iconv\_open()* shall return a conversion descriptor for use on  
 19542 subsequent calls to *iconv()*. Otherwise, *iconv\_open()* shall return **(iconv\_t)-1** and set *errno* to  
 19543 indicate the error.

19544 **ERRORS**

19545 The *iconv\_open()* function may fail if:

19546 [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

19547 [ENFILE] Too many files are currently open in the system.

19548 [ENOMEM] Insufficient storage space is available.

19549 [EINVAL] The conversion specified by *fromcode* and *tocode* is not supported by the  
 19550 implementation.

19551 **EXAMPLES**

19552 None.

19553 **APPLICATION USAGE**

19554 Some implementations of *iconv\_open()* use *malloc()* to allocate space for internal buffer areas.  
 19555 The *iconv\_open()* function may fail if there is insufficient storage space to accommodate these  
 19556 buffers.

19557 Conforming applications must assume that conversion descriptors are not valid after a call to  
 19558 one of the *exec* functions.

19559 Application developers should consult the system documentation to determine the supported  
 19560 codesets and their naming schemes.

19561 **RATIONALE**

19562 None.

19563 **FUTURE DIRECTIONS**

19564 None.

19565 **SEE ALSO**

19566 *iconv()*, *iconv\_close()*, the Base Definitions volume of IEEE Std 1003.1-2001, <fcntl.h>, <iconv.h>

19567 **CHANGE HISTORY**

19568 First released in Issue 4. Derived from the HP-UX Manual.

19569 **NAME**

19570 if\_freenameindex — free memory allocated by if\_nameindex

19571 **SYNOPSIS**

19572 #include <net/if.h>

19573 void if\_freenameindex(struct if\_nameindex \*ptr);

19574 **DESCRIPTION**

19575 The *if\_freenameindex()* function shall free the memory allocated by *if\_nameindex()*. The *ptr*  
19576 argument shall be a pointer that was returned by *if\_nameindex()*. After *if\_freenameindex()* has  
19577 been called, the application shall not use the array of which *ptr* is the address.

19578 **RETURN VALUE**

19579 None.

19580 **ERRORS**

19581 No errors are defined.

19582 **EXAMPLES**

19583 None.

19584 **APPLICATION USAGE**

19585 None.

19586 **RATIONALE**

19587 None.

19588 **FUTURE DIRECTIONS**

19589 None.

19590 **SEE ALSO**

19591 *getsockopt()*, *if\_indextoname()*, *if\_nameindex()*, *if\_nametoindex()*, *setsockopt()*, the Base Definitions  
19592 volume of IEEE Std 1003.1-2001, <net/if.h>

19593 **CHANGE HISTORY**

19594 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19595 **NAME**

19596 if\_indextoname — map a network interface index to its corresponding name

19597 **SYNOPSIS**

19598 #include <net/if.h>

19599 char \*if\_indextoname(unsigned *ifindex*, char \**ifname*);

19600 **DESCRIPTION**

19601 The *if\_indextoname*() function shall map an interface index to its corresponding name.

19602 When this function is called, *ifname* shall point to a buffer of at least {IF\_NAMESIZE} bytes. The  
19603 function shall place in this buffer the name of the interface with index *ifindex*.

19604 **RETURN VALUE**

19605 If *ifindex* is an interface index, then the function shall return the value supplied in *ifname*, which  
19606 points to a buffer now containing the interface name. Otherwise, the function shall return a  
19607 NULL pointer and set *errno* to indicate the error.

19608 **ERRORS**

19609 The *if\_indextoname*() function shall fail if:

19610 [ENXIO] The interface does not exist.

19611 **EXAMPLES**

19612 None.

19613 **APPLICATION USAGE**

19614 None.

19615 **RATIONALE**

19616 None.

19617 **FUTURE DIRECTIONS**

19618 None.

19619 **SEE ALSO**

19620 *getsockopt*(), *if\_freenameindex*(), *if\_nameindex*(), *if\_nametoindex*(), *setsockopt*(), the Base  
19621 Definitions volume of IEEE Std 1003.1-2001, <net/if.h>

19622 **CHANGE HISTORY**

19623 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19624 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/28 is applied, changing {IFNAMSIZ} to  
19625 {IF\_NAMESIZ} in the DESCRIPTION. |

19626 **NAME**

19627 if\_nameindex — return all network interface names and indexes

19628 **SYNOPSIS**

19629 #include <net/if.h>

19630 struct if\_nameindex \*if\_nameindex(void);

19631 **DESCRIPTION**

19632 The *if\_nameindex()* function shall return an array of *if\_nameindex* structures, one structure per  
19633 interface. The end of the array is indicated by a structure with an *if\_index* field of zero and an  
19634 *if\_name* field of NULL.

19635 Applications should call *if\_freenameindex()* to release the memory that may be dynamically  
19636 allocated by this function, after they have finished using it.

19637 **RETURN VALUE**

19638 An array of structures identifying local interfaces. A NULL pointer is returned upon an error,  
19639 with *errno* set to indicate the error.

19640 **ERRORS**

19641 The *if\_nameindex()* function may fail if:

19642 [ENOBUFS] Insufficient resources are available to complete the function.

19643 **EXAMPLES**

19644 None.

19645 **APPLICATION USAGE**

19646 None.

19647 **RATIONALE**

19648 None.

19649 **FUTURE DIRECTIONS**

19650 None.

19651 **SEE ALSO**

19652 *getsockopt()*, *if\_freenameindex()*, *if\_indextoname()*, *if\_nametoindex()*, *setsockopt()*, the Base  
19653 Definitions volume of IEEE Std 1003.1-2001, <net/if.h>

19654 **CHANGE HISTORY**

19655 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19656 **NAME**

19657 if\_nametoindex — map a network interface name to its corresponding index

19658 **SYNOPSIS**

19659 #include <net/if.h>

19660 unsigned if\_nametoindex(const char \*ifname);

19661 **DESCRIPTION**

19662 The *if\_nametoindex()* function shall return the interface index corresponding to name *ifname*.

19663 **RETURN VALUE**

19664 The corresponding index if *ifname* is the name of an interface; otherwise, zero.

19665 **ERRORS**

19666 No errors are defined.

19667 **EXAMPLES**

19668 None.

19669 **APPLICATION USAGE**

19670 None.

19671 **RATIONALE**

19672 None.

19673 **FUTURE DIRECTIONS**

19674 None.

19675 **SEE ALSO**

19676 *getsockopt()*, *if\_freenameindex()*, *if\_indextoname()*, *if\_nameindex()*, *setsockopt()*, the Base  
19677 Definitions volume of IEEE Std 1003.1-2001, <net/if.h>

19678 **CHANGE HISTORY**

19679 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19680 **NAME**

19681           ilogb, ilogbf, ilogbl — return an unbiased exponent

19682 **SYNOPSIS**

```
19683 #include <math.h>

19684 int ilogb(double x);
19685 int ilogbf(float x);
19686 int ilogbl(long double x);
```

19687 **DESCRIPTION**

19688 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 19689 conflict between the requirements described here and the ISO C standard is unintentional. This  
 19690 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

19691       These functions shall return the exponent part of their argument  $x$ . Formally, the return value is  
 19692 the integral part of  $\log_r |x|$  as a signed integral value, for non-zero  $x$ , where  $r$  is the radix of the  
 19693 machine's floating-point arithmetic, which is the value of `FLT_RADIX` defined in `<float.h>`.

19694       An application wishing to check for error situations should set `errno` to zero and call  
 19695 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or  
 19696 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-  
 19697 zero, an error has occurred.

19698 **RETURN VALUE**

19699       Upon successful completion, these functions shall return the exponent part of  $x$  as a signed  
 19700 integer value. They are equivalent to calling the corresponding `logb()` function and casting the  
 19701 returned value to type `int`.

19702 **XSI**       If  $x$  is 0, a domain error shall occur, and the value `FP_ILOGB0` shall be returned.

19703 **XSI**       If  $x$  is  $\pm\text{Inf}$ , a domain error shall occur, and the value `{INT_MAX}` shall be returned.

19704 **XSI**       If  $x$  is a NaN, a domain error shall occur, and the value `FP_ILOGBNAN` shall be returned.

19705 **XSI**       If the correct value is greater than `{INT_MAX}`, `{INT_MAX}` shall be returned and a domain error  
 19706 shall occur.

19707       If the correct value is less than `{INT_MIN}`, `{INT_MIN}` shall be returned and a domain error  
 19708 shall occur.

19709 **ERRORS**

19710       These functions shall fail if:

|                  |                     |                                                                                                              |
|------------------|---------------------|--------------------------------------------------------------------------------------------------------------|
| 19711 <b>XSI</b> | <b>Domain Error</b> | The $x$ argument is zero, NaN, or $\pm\text{Inf}$ , or the correct value is not representable as an integer. |
|------------------|---------------------|--------------------------------------------------------------------------------------------------------------|

|       |                                                                                                                                                                                                                                                                                                              |
|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 19713 | If the integer expression <code>(math_errhandling &amp; MATH_ERRNO)</code> is non-zero, then <code>errno</code> shall be set to <code>[EDOM]</code> . If the integer expression <code>(math_errhandling &amp; MATH_ERREXCEPT)</code> is non-zero, then the invalid floating-point exception shall be raised. |
| 19714 |                                                                                                                                                                                                                                                                                                              |
| 19715 |                                                                                                                                                                                                                                                                                                              |
| 19716 |                                                                                                                                                                                                                                                                                                              |

19717 **EXAMPLES**

19718 None.

19719 **APPLICATION USAGE**

19720 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
19721 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

19722 **RATIONALE**

19723 The errors come from taking the expected floating-point value and converting it to **int**, which is  
19724 an invalid operation in IEEE Std 754-1985 (since overflow, infinity, and NaN are not  
19725 representable in a type **int**), so should be a domain error.

19726 There are no known implementations that overflow. For overflow to happen, {INT\_MAX} must  
19727 be less than  $LDBL\_MAX\_EXP * \log_2(\text{FLT\_RADIX})$  or {INT\_MIN} must be greater than  
19728  $LDBL\_MIN\_EXP * \log_2(\text{FLT\_RADIX})$  if subnormals are not supported, or {INT\_MIN} must be  
19729 greater than  $(LDBL\_MIN\_EXP - LDBL\_MANT\_DIG) * \log_2(\text{FLT\_RADIX})$  if subnormals are  
19730 supported.

19731 **FUTURE DIRECTIONS**

19732 None.

19733 **SEE ALSO**

19734 *feclearexcept()*, *fetestexcept()*, *logb()*, *scalb()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
19735 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <float.h>, <math.h>

19736 **CHANGE HISTORY**

19737 First released in Issue 4, Version 2.

19738 **Issue 5**

19739 Moved from X/OPEN UNIX extension to BASE.

19740 **Issue 6**19741 The *ilogb()* function is no longer marked as an extension.

19742 The *ilogbf()* and *ilogbl()* functions are added for alignment with the ISO/IEC 9899:1999  
19743 standard.

19744 The RETURN VALUE section is revised for alignment with the ISO/IEC 9899:1999 standard.

19745 XSI extensions are marked.

19746 **NAME**

19747            *imaxabs* — return absolute value

19748 **SYNOPSIS**

19749            `#include <inttypes.h>`

19750            `intmax_t imaxabs(intmax_t j);`

19751 **DESCRIPTION**

19752 *CX*        The functionality described on this reference page is aligned with the ISO C standard. Any  
19753            conflict between the requirements described here and the ISO C standard is unintentional. This  
19754            volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

19755            The *imaxabs()* function shall compute the absolute value of an integer *j*. If the result cannot be  
19756            represented, the behavior is undefined.

19757 **RETURN VALUE**

19758            The *imaxabs()* function shall return the absolute value.

19759 **ERRORS**

19760            No errors are defined.

19761 **EXAMPLES**

19762            None.

19763 **APPLICATION USAGE**

19764            The absolute value of the most negative number cannot be represented in two's complement.

19765 **RATIONALE**

19766            None.

19767 **FUTURE DIRECTIONS**

19768            None.

19769 **SEE ALSO**

19770            *imaxdiv()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<inttypes.h>`

19771 **CHANGE HISTORY**

19772            First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

19773 **NAME**

19774 imaxdiv — return quotient and remainder

19775 **SYNOPSIS**

19776 #include <inttypes.h>

19777 imaxdiv\_t imaxdiv(intmax\_t numer, intmax\_t denom);

19778 **DESCRIPTION**

19779 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
19780 conflict between the requirements described here and the ISO C standard is unintentional. This  
19781 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

19782 The *imaxdiv()* function shall compute *numer* / *denom* and *numer* % *denom* in a single operation.

19783 **RETURN VALUE**

19784 The *imaxdiv()* function shall return a structure of type **imaxdiv\_t**, comprising both the quotient  
19785 and the remainder. The structure shall contain (in either order) the members *quot* (the quotient)  
19786 and *rem* (the remainder), each of which has type **intmax\_t**.

19787 If either part of the result cannot be represented, the behavior is undefined.

19788 **ERRORS**

19789 No errors are defined.

19790 **EXAMPLES**

19791 None.

19792 **APPLICATION USAGE**

19793 None.

19794 **RATIONALE**

19795 None.

19796 **FUTURE DIRECTIONS**

19797 None.

19798 **SEE ALSO**

19799 *imaxabs()*, the Base Definitions volume of IEEE Std 1003.1-2001, <inttypes.h>

19800 **CHANGE HISTORY**

19801 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

19802 **NAME**19803           index — character string operations (**LEGACY**)19804 **SYNOPSIS**

19805 XSI        #include &lt;strings.h&gt;

19806           char \*index(const char \*s, int c);

19807

19808 **DESCRIPTION**19809           The *index()* function shall be equivalent to *strchr()*.19810 **RETURN VALUE**19811           See *strchr()*.19812 **ERRORS**19813           See *strchr()*.19814 **EXAMPLES**

19815           None.

19816 **APPLICATION USAGE**19817           The *strchr()* function is preferred over this function.19818           For maximum portability, it is recommended to replace the function call to *index()* as follows:

19819           #define index(a,b) strchr((a),(b))

19820 **RATIONALE**

19821           None.

19822 **FUTURE DIRECTIONS**

19823           This function may be withdrawn in a future version.

19824 **SEE ALSO**19825           *strchr()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**strings.h**>19826 **CHANGE HISTORY**

19827           First released in Issue 4, Version 2.

19828 **Issue 5**

19829           Moved from X/OPEN UNIX extension to BASE.

19830 **Issue 6**

19831           This function is marked LEGACY.

19832 **NAME**

19833 inet\_addr, inet\_ntoa — IPv4 address manipulation

19834 **SYNOPSIS**

19835 #include &lt;arpa/inet.h&gt;

19836 in\_addr\_t inet\_addr(const char \*cp);

19837 char \*inet\_ntoa(struct in\_addr in);

19838 **DESCRIPTION**19839 The *inet\_addr()* function shall convert the string pointed to by *cp*, in the standard IPv4 dotted decimal notation, to an integer value suitable for use as an Internet address.19841 The *inet\_ntoa()* function shall convert the Internet host address specified by *in* to a string in the Internet standard dot notation.19843 The *inet\_ntoa()* function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

19845 All Internet addresses shall be returned in network order (bytes ordered from left to right).

19846 Values specified using IPv4 dotted decimal notation take one of the following forms:

19847 a.b.c.d When four parts are specified, each shall be interpreted as a byte of data and  
19848 assigned, from left to right, to the four bytes of an Internet address.19849 a.b.c When a three-part address is specified, the last part shall be interpreted as a 16-bit  
19850 quantity and placed in the rightmost two bytes of the network address. This makes  
19851 the three-part address format convenient for specifying Class B network addresses  
19852 as "128.net.host".19853 a.b When a two-part address is supplied, the last part shall be interpreted as a 24-bit  
19854 quantity and placed in the rightmost three bytes of the network address. This  
19855 makes the two-part address format convenient for specifying Class A network  
19856 addresses as "net.host".19857 a When only one part is given, the value shall be stored directly in the network  
19858 address without any byte rearrangement.19859 All numbers supplied as parts in IPv4 dotted decimal notation may be decimal, octal, or  
19860 hexadecimal, as specified in the ISO C standard (that is, a leading 0x or 0X implies hexadecimal;  
19861 otherwise, a leading '0' implies octal; otherwise, the number is interpreted as decimal).19862 **RETURN VALUE**19863 Upon successful completion, *inet\_addr()* shall return the Internet address. Otherwise, it shall  
19864 return (**in\_addr\_t**)(-1).19865 The *inet\_ntoa()* function shall return a pointer to the network address in Internet standard dot  
19866 notation.19867 **ERRORS**

19868 No errors are defined.

19869 **EXAMPLES**

19870           None.

19871 **APPLICATION USAGE**19872           The return value of *inet\_ntoa()* may point to static data that may be overwritten by subsequent  
19873           calls to *inet\_ntoa()*.19874 **RATIONALE**

19875           None.

19876 **FUTURE DIRECTIONS**

19877           None.

19878 **SEE ALSO**19879           *endhostent()*, *endnetent()*, the Base Definitions volume of IEEE Std 1003.1-2001, <arpa/inet.h>19880 **CHANGE HISTORY**

19881           First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

## 19882 NAME

19883 inet\_ntop, inet\_pton — convert IPv4 and IPv6 addresses between binary and text form

## 19884 SYNOPSIS

19885 #include <arpa/inet.h>

```
19886 const char *inet_ntop(int af, const void *restrict src,
19887 char *restrict dst, socklen_t size);
19888 int inet_pton(int af, const char *restrict src, void *restrict dst);
```

## 19889 DESCRIPTION

19890 The *inet\_ntop()* function shall convert a numeric address into a text string suitable for  
 19891 IP6 presentation. The *af* argument shall specify the family of the address. This can be AF\_INET or  
 19892 AF\_INET6. The *src* argument points to a buffer holding an IPv4 address if the *af* argument is  
 19893 IP6 AF\_INET, or an IPv6 address if the *af* argument is AF\_INET6; the address must be in network  
 19894 byte order. The *dst* argument points to a buffer where the function stores the resulting text  
 19895 string; it shall not be NULL. The *size* argument specifies the size of this buffer, which shall be  
 19896 IP6 large enough to hold the text string (INET\_ADDRSTRLEN characters for IPv4,  
 19897 INET6\_ADDRSTRLEN characters for IPv6).

19898 The *inet\_pton()* function shall convert an address in its standard text presentation form into its  
 19899 IP6 numeric binary form. The *af* argument shall specify the family of the address. The AF\_INET and  
 19900 AF\_INET6 address families shall be supported. The *src* argument points to the string being  
 19901 passed in. The *dst* argument points to a buffer into which the function stores the numeric  
 19902 IP6 address; this shall be large enough to hold the numeric address (32 bits for AF\_INET, 128 bits for  
 19903 AF\_INET6).

19904 If the *af* argument of *inet\_pton()* is AF\_INET, the *src* string shall be in the standard IPv4 dotted-  
 19905 decimal form:

19906 ddd.ddd.ddd.ddd

19907 where "ddd" is a one to three digit decimal number between 0 and 255 (see *inet\_addr()*). The  
 19908 *inet\_pton()* function does not accept other formats (such as the octal numbers, hexadecimal  
 19909 numbers, and fewer than four numbers that *inet\_addr()* accepts).

19910 IP6 If the *af* argument of *inet\_pton()* is AF\_INET6, the *src* string shall be in one of the following  
 19911 standard IPv6 text forms:

19912 1. The preferred form is "x:x:x:x:x:x:x:x", where the 'x's are the hexadecimal values  
 19913 of the eight 16-bit pieces of the address. Leading zeros in individual fields can be omitted,  
 19914 but there shall be at least one numeral in every field.

19915 2. A string of contiguous zero fields in the preferred form can be shown as "::". The "::"  
 19916 can only appear once in an address. Unspecified addresses ("0:0:0:0:0:0:0:0") may  
 19917 be represented simply as "::".

19918 3. A third form that is sometimes more convenient when dealing with a mixed environment  
 19919 of IPv4 and IPv6 nodes is "x:x:x:x:x:x.d.d.d.d", where the 'x's are the  
 19920 hexadecimal values of the six high-order 16-bit pieces of the address, and the 'd's are the  
 19921 decimal values of the four low-order 8-bit pieces of the address (standard IPv4  
 19922 representation).

19923 **Note:** A more extensive description of the standard representations of IPv6 addresses can be found in  
 19924 RFC 2373.

19925

19926 **RETURN VALUE**

19927 The *inet\_ntop()* function shall return a pointer to the buffer containing the text string if the  
19928 conversion succeeds, and NULL otherwise, and set *errno* to indicate the error.

19929 The *inet\_pton()* function shall return 1 if the conversion succeeds, with the address pointed to by  
19930 *dst* in network byte order. It shall return 0 if the input is not a valid IPv4 dotted-decimal string or  
19931 a valid IPv6 address string, or -1 with *errno* set to [EAFNOSUPPORT] if the *af* argument is  
19932 unknown.

19933 **ERRORS**

19934 The *inet\_ntop()* and *inet\_pton()* functions shall fail if:

19935 [EAFNOSUPPORT]

19936 The *af* argument is invalid.

19937 [ENOSPC] The size of the *inet\_ntop()* result buffer is inadequate.

19938 **EXAMPLES**

19939 None.

19940 **APPLICATION USAGE**

19941 None.

19942 **RATIONALE**

19943 None.

19944 **FUTURE DIRECTIONS**

19945 None.

19946 **SEE ALSO**

19947 The Base Definitions volume of IEEE Std 1003.1-2001, <[arpa/inet.h](#)>

19948 **CHANGE HISTORY**

19949 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19950 IPv6 extensions are marked.

19951 The **restrict** keyword is added to the *inet\_ntop()* and *inet\_pton()* prototypes for alignment with  
19952 the ISO/IEC 9899:1999 standard.

19953 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/29 is applied, adding “the address must  
19954 be in network byte order” to the end of the fourth sentence of the first paragraph in the  
19955 DESCRIPTION.

## 19956 NAME

19957       initstate, random, setstate, srandom — pseudo-random number functions

## 19958 SYNOPSIS

```
19959 xSI #include <stdlib.h>
19960 char *initstate(unsigned seed, char *state, size_t size);
19961 long random(void);
19962 char *setstate(const char *state);
19963 void srandom(unsigned seed);
```

19964

## 19965 DESCRIPTION

19966       The *random()* function shall use a non-linear additive feedback random-number generator  
 19967       employing a default state array size of 31 **long** integers to return successive pseudo-random  
 19968       numbers in the range from 0 to  $2^{31}-1$ . The period of this random-number generator is  
 19969       approximately  $16 \times (2^{31}-1)$ . The size of the state array determines the period of the random-  
 19970       number generator. Increasing the state array size shall increase the period.

19971       With 256 bytes of state information, the period of the random-number generator shall be greater  
 19972       than  $2^{69}$ .

19973       Like *rand()*, *random()* shall produce by default a sequence of numbers that can be duplicated by  
 19974       calling *srandom()* with 1 as the seed.

19975       The *srandom()* function shall initialize the current state array using the value of *seed*.

19976       The *initstate()* and *setstate()* functions handle restarting and changing random-number  
 19977       generators. The *initstate()* function allows a state array, pointed to by the *state* argument, to be  
 19978       initialized for future use. The *size* argument, which specifies the size in bytes of the state array,  
 19979       shall be used by *initstate()* to decide what type of random-number generator to use; the larger  
 19980       the state array, the more random the numbers. Values for the amount of state information are 8,  
 19981       32, 64, 128, and 256 bytes. Other values greater than 8 bytes are rounded down to the nearest one  
 19982       of these values. If *initstate()* is called with  $8 \leq \text{size} < 32$ , then *random()* shall use a simple linear  
 19983       congruential random number generator. The *seed* argument specifies a starting point for the  
 19984       random-number sequence and provides for restarting at the same point. The *initstate()* function  
 19985       shall return a pointer to the previous state information array.

19986       If *initstate()* has not been called, then *random()* shall behave as though *initstate()* had been called  
 19987       with *seed*=1 and *size*=128.

19988       Once a state has been initialized, *setstate()* allows switching between state arrays. The array  
 19989       defined by the *state* argument shall be used for further random-number generation until  
 19990       *initstate()* is called or *setstate()* is called again. The *setstate()* function shall return a pointer to the  
 19991       previous state array.

## 19992 RETURN VALUE

19993       If *initstate()* is called with *size* less than 8, it shall return NULL.

19994       The *random()* function shall return the generated pseudo-random number.

19995       The *srandom()* function shall not return a value.

19996       Upon successful completion, *initstate()* and *setstate()* shall return a pointer to the previous state  
 19997       array; otherwise, a null pointer shall be returned.

19998 **ERRORS**

19999 No errors are defined.

20000 **EXAMPLES**

20001 None.

20002 **APPLICATION USAGE**

20003 After initialization, a state array can be restarted at a different point in one of two ways:

- 20004 1. The *initstate()* function can be used, with the desired seed, state array, and size of the  
20005 array.
- 20006 2. The *setstate()* function, with the desired state, can be used, followed by *srandom()* with the  
20007 desired seed. The advantage of using both of these functions is that the size of the state  
20008 array does not have to be saved once it is initialized.

20009 Although some implementations of *random()* have written messages to standard error, such  
20010 implementations do not conform to IEEE Std 1003.1-2001.

20011 Issue 5 restored the historical behavior of this function.

20012 Threaded applications should use *erand48()*, *nrand48()*, or *jrand48()* instead of *random()* when  
20013 an independent random number sequence in multiple threads is required.20014 **RATIONALE**

20015 None.

20016 **FUTURE DIRECTIONS**

20017 None.

20018 **SEE ALSO**20019 *drand48()*, *rand()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>20020 **CHANGE HISTORY**

20021 First released in Issue 4, Version 2.

20022 **Issue 5**

20023 Moved from X/OPEN UNIX extension to BASE.

20024 In the DESCRIPTION, the phrase “values smaller than 8” is replaced with “values greater than  
20025 or equal to 8, or less than 32”, “*size*<8” is replaced with “ $8 \leq \textit{size} < 32$ ”, and a new first paragraph  
20026 is added to the RETURN VALUE section. A note is added to the APPLICATION USAGE  
20027 indicating that these changes restore the historical behavior of the function.

20028 **Issue 6**

20029 In the DESCRIPTION, duplicate text “For values greater than or equal to 8 . . .” is removed.

20030 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/30 is applied, removing *rand\_r()* from the  
20031 list of suggested functions in the APPLICATION USAGE section.

20032 **NAME**

20033 insque, remque — insert or remove an element in a queue

20034 **SYNOPSIS**

```
20035 xSI #include <search.h>
20036 void insque(void *element, void *pred);
20037 void remque(void *element);
20038
```

20039 **DESCRIPTION**

20040 The *insque()* and *remque()* functions shall manipulate queues built from doubly-linked lists. The  
20041 queue can be either circular or linear. An application using *insque()* or *remque()* shall ensure it  
20042 defines a structure in which the first two members of the structure are pointers to the same type  
20043 of structure, and any further members are application-specific. The first member of the structure  
20044 is a forward pointer to the next entry in the queue. The second member is a backward pointer to  
20045 the previous entry in the queue. If the queue is linear, the queue is terminated with null  
20046 pointers. The names of the structure and of the pointer members are not subject to any special  
20047 restriction.

20048 The *insque()* function shall insert the element pointed to by *element* into a queue immediately  
20049 after the element pointed to by *pred*.

20050 The *remque()* function shall remove the element pointed to by *element* from a queue.

20051 If the queue is to be used as a linear list, invoking *insque(&element, NULL)*, where *element* is the  
20052 initial element of the queue, shall initialize the forward and backward pointers of *element* to null  
20053 pointers.

20054 If the queue is to be used as a circular list, the application shall ensure it initializes the forward  
20055 pointer and the backward pointer of the initial element of the queue to the element's own  
20056 address.

20057 **RETURN VALUE**20058 The *insque()* and *remque()* functions do not return a value.20059 **ERRORS**

20060 No errors are defined.

20061 **EXAMPLES**20062 **Creating a Linear Linked List**

20063 The following example creates a linear linked list.

```
20064 #include <search.h>
20065 ...
20066 struct myque element1;
20067 struct myque element2;
20068 char *data1 = "DATA1";
20069 char *data2 = "DATA2";
20070 ...
20071 element1.data = data1;
20072 element2.data = data2;
20073 insque (&element1, NULL);
20074 insque (&element2, &element1);
```

20075 **Creating a Circular Linked List**

20076 The following example creates a circular linked list.

```

20077 #include <search.h>
20078 ...
20079 struct myque element1;
20080 struct myque element2;

20081 char *data1 = "DATA1";
20082 char *data2 = "DATA2";
20083 ...
20084 element1.data = data1;
20085 element2.data = data2;

20086 element1.fwd = &element1;
20087 element1.bck = &element1;

20088 insque (&element2, &element1);

```

20089 **Removing an Element**20090 The following example removes the element pointed to by *element1*.

```

20091 #include <search.h>
20092 ...
20093 struct myque element1;
20094 ...
20095 remque (&element1);

```

20096 **APPLICATION USAGE**

20097 The historical implementations of these functions described the arguments as being of type  
 20098 **struct qelem** \* rather than as being of type **void** \* as defined here. In those implementations,  
 20099 **struct qelem** was commonly defined in <**search.h**> as:

```

20100 struct qelem {
20101 struct qelem *q_forw;
20102 struct qelem *q_back;
20103 };

```

20104 Applications using these functions, however, were never able to use this structure directly since  
 20105 it provided no room for the actual data contained in the elements. Most applications defined  
 20106 structures that contained the two pointers as the initial elements and also provided space for, or  
 20107 pointers to, the object's data. Applications that used these functions to update more than one  
 20108 type of table also had the problem of specifying two or more different structures with the same  
 20109 name, if they literally used **struct qelem** as specified.

20110 As described here, the implementations were actually expecting a structure type where the first  
 20111 two members were forward and backward pointers to structures. With C compilers that didn't  
 20112 provide function prototypes, applications used structures as specified in the DESCRIPTION  
 20113 above and the compiler did what the application expected.

20114 If this method had been carried forward with an ISO C standard compiler and the historical  
 20115 function prototype, most applications would have to be modified to cast pointers to the  
 20116 structures actually used to be pointers to **struct qelem** to avoid compilation warnings. By  
 20117 specifying **void** \* as the argument type, applications do not need to change (unless they  
 20118 specifically referenced **struct qelem** and depended on it being defined in <**search.h**>).

20119 **RATIONALE**

20120 None.

20121 **FUTURE DIRECTIONS**

20122 None.

20123 **SEE ALSO**

20124 The Base Definitions volume of IEEE Std 1003.1-2001, &lt;search.h&gt;

20125 **CHANGE HISTORY**

20126 First released in Issue 4, Version 2.

20127 **Issue 5**

20128 Moved from X/OPEN UNIX extension to BASE.

20129 **Issue 6**

20130 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20131 **NAME**20132            **ioctl** — control a STREAMS device (**STREAMS**)20133 **SYNOPSIS**

20134 XSR        #include &lt;stropts.h&gt;

20135            int ioctl(int *fildev*, int *request*, ... /\* *arg* \*/);

20136

20137 **DESCRIPTION**

20138        The *ioctl()* function shall perform a variety of control functions on STREAMS devices. For non-  
 20139        STREAMS devices, the functions performed by this call are unspecified. The *request* argument  
 20140        and an optional third argument (with varying type) shall be passed to and interpreted by the  
 20141        appropriate part of the STREAM associated with *fildev*.

20142        The *fildev* argument is an open file descriptor that refers to a device.

20143        The *request* argument selects the control function to be performed and shall depend on the  
 20144        STREAMS device being addressed.

20145        The *arg* argument represents additional information that is needed by this specific STREAMS  
 20146        device to perform the requested function. The type of *arg* depends upon the particular control  
 20147        request, but it shall be either an integer or a pointer to a device-specific data structure.

20148        The *ioctl()* commands applicable to STREAMS, their arguments, and error conditions that apply  
 20149        to each individual command are described below.

20150        The following *ioctl()* commands, with error values indicated, are applicable to all STREAMS  
 20151        files:

20152        **I\_PUSH**            Pushes the module whose name is pointed to by *arg* onto the top of the  
 20153        current STREAM, just below the STREAM head. It then calls the *open()*  
 20154        function of the newly-pushed module.

20155            The *ioctl()* function with the I\_PUSH command shall fail if:

20156            [EINVAL]        Invalid module name.

20157            [ENXIO]        Open function of new module failed.

20158            [ENXIO]        Hangup received on *fildev*.

20159        **I\_POP**            Removes the module just below the STREAM head of the STREAM pointed to  
 20160        by *fildev*. The *arg* argument should be 0 in an I\_POP request.

20161            The *ioctl()* function with the I\_POP command shall fail if:

20162            [EINVAL]        No module present in the STREAM.

20163            [ENXIO]        Hangup received on *fildev*.

20164        **I\_LOOK**           Retrieves the name of the module just below the STREAM head of the  
 20165        STREAM pointed to by *fildev*, and places it in a character string pointed to by  
 20166        *arg*. The buffer pointed to by *arg* should be at least FMNAMESZ+1 bytes long,  
 20167        where FMNAMESZ is defined in <stropts.h>.

20168            The *ioctl()* function with the I\_LOOK command shall fail if:

20169            [EINVAL]        No module present in the STREAM.

20170        **I\_FLUSH**          Flushes read and/or write queues, depending on the value of *arg*. Valid *arg*  
 20171        values are:

|       |             |                                                                                            |                                                              |
|-------|-------------|--------------------------------------------------------------------------------------------|--------------------------------------------------------------|
| 20172 |             | FLUSHR                                                                                     | Flush all read queues.                                       |
| 20173 |             | FLUSHW                                                                                     | Flush all write queues.                                      |
| 20174 |             | FLUSHRW                                                                                    | Flush all read and all write queues.                         |
| 20175 |             | The <i>ioctl()</i> function with the <code>L_FLUSH</code> command shall fail if:           |                                                              |
| 20176 |             | [EINVAL]                                                                                   | Invalid <i>arg</i> value.                                    |
| 20177 |             | [EAGAIN] or [ENOSR]                                                                        | Unable to allocate buffers for flush message.                |
| 20178 |             |                                                                                            |                                                              |
| 20179 |             | [ENXIO]                                                                                    | Hangup received on <i>fildev</i> .                           |
| 20180 | I_FLUSHBAND | Flushes a particular band of messages. The <i>arg</i> argument points to a <b>bandinfo</b> |                                                              |
| 20181 |             | structure. The <i>bi_flag</i> member may be one of FLUSHR, FLUSHW, or                      |                                                              |
| 20182 |             | FLUSHRW as described above. The <i>bi_pri</i> member determines the priority               |                                                              |
| 20183 |             | band to be flushed.                                                                        |                                                              |
| 20184 | I_SETSIG    | Requests that the STREAMS implementation send the SIGPOLL signal to the                    |                                                              |
| 20185 |             | calling process when a particular event has occurred on the STREAM                         |                                                              |
| 20186 |             | associated with <i>fildev</i> . I_SETSIG supports an asynchronous processing               |                                                              |
| 20187 |             | capability in STREAMS. The value of <i>arg</i> is a bitmask that specifies the events      |                                                              |
| 20188 |             | for which the process should be signaled. It is the bitwise-inclusive OR of any            |                                                              |
| 20189 |             | combination of the following constants:                                                    |                                                              |
| 20190 |             | S_RDNORM                                                                                   | A normal (priority band set to 0) message has arrived at the |
| 20191 |             |                                                                                            | head of a STREAM head read queue. A signal shall be          |
| 20192 |             |                                                                                            | generated even if the message is of zero length.             |
| 20193 |             | S_RDBAND                                                                                   | A message with a non-zero priority band has arrived at the   |
| 20194 |             |                                                                                            | head of a STREAM head read queue. A signal shall be          |
| 20195 |             |                                                                                            | generated even if the message is of zero length.             |
| 20196 |             | S_INPUT                                                                                    | A message, other than a high-priority message, has arrived   |
| 20197 |             |                                                                                            | at the head of a STREAM head read queue. A signal shall be   |
| 20198 |             |                                                                                            | generated even if the message is of zero length.             |
| 20199 |             | S_HIPRI                                                                                    | A high-priority message is present on a STREAM head read     |
| 20200 |             |                                                                                            | queue. A signal shall be generated even if the message is of |
| 20201 |             |                                                                                            | zero length.                                                 |
| 20202 |             | S_OUTPUT                                                                                   | The write queue for normal data (priority band 0) just       |
| 20203 |             |                                                                                            | below the STREAM head is no longer full. This notifies the   |
| 20204 |             |                                                                                            | process that there is room on the queue for sending (or      |
| 20205 |             |                                                                                            | writing) normal data downstream.                             |
| 20206 |             | S_WRNORM                                                                                   | Equivalent to S_OUTPUT.                                      |
| 20207 |             | S_WRBAND                                                                                   | The write queue for a non-zero priority band just below the  |
| 20208 |             |                                                                                            | STREAM head is no longer full. This notifies the process     |
| 20209 |             |                                                                                            | that there is room on the queue for sending (or writing)     |
| 20210 |             |                                                                                            | priority data downstream.                                    |
| 20211 |             | S_MSG                                                                                      | A STREAMS signal message that contains the SIGPOLL           |
| 20212 |             |                                                                                            | signal has reached the front of the STREAM head read         |
| 20213 |             |                                                                                            | queue.                                                       |
| 20214 |             | S_ERROR                                                                                    | Notification of an error condition has reached the STREAM    |
| 20215 |             |                                                                                            | head.                                                        |

|       |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------|----------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 20216 |          | S_HANGUP  | Notification of a hangup has reached the STREAM head.                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 20217 |          | S_BANDURG | When used in conjunction with S_RDBAND, SIGURG is generated instead of SIGPOLL when a priority message reaches the front of the STREAM head read queue.                                                                                                                                                                                                                                                                                                                  |
| 20218 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20219 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20220 |          |           | If <i>arg</i> is 0, the calling process shall be unregistered and shall not receive further SIGPOLL signals for the stream associated with <i>files</i> .                                                                                                                                                                                                                                                                                                                |
| 20221 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20222 |          |           | Processes that wish to receive SIGPOLL signals shall ensure that they explicitly register to receive them using I_SETSIG. If several processes register to receive this signal for the same event on the same STREAM, each process shall be signaled when the event occurs.                                                                                                                                                                                              |
| 20223 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20224 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20225 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20226 |          |           | The <i>ioctl()</i> function with the I_SETSIG command shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20227 |          | [EINVAL]  | The value of <i>arg</i> is invalid.                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 20228 |          | [EINVAL]  | The value of <i>arg</i> is 0 and the calling process is not registered to receive the SIGPOLL signal.                                                                                                                                                                                                                                                                                                                                                                    |
| 20229 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20230 |          | [EAGAIN]  | There were insufficient resources to store the signal request.                                                                                                                                                                                                                                                                                                                                                                                                           |
| 20231 | I_GETSIG |           | Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask in an <b>int</b> pointed to by <i>arg</i> , where the events are those specified in the description of I_SETSIG above.                                                                                                                                                                                                        |
| 20232 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20233 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20234 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20235 |          |           | The <i>ioctl()</i> function with the I_GETSIG command shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20236 |          | [EINVAL]  | Process is not registered to receive the SIGPOLL signal.                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20237 | I_FIND   |           | Compares the names of all modules currently present in the STREAM to the name pointed to by <i>arg</i> , and returns 1 if the named module is present in the STREAM, or returns 0 if the named module is not present.                                                                                                                                                                                                                                                    |
| 20238 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20239 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20240 |          |           | The <i>ioctl()</i> function with the I_FIND command shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20241 |          | [EINVAL]  | <i>arg</i> does not contain a valid module name.                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20242 | I_PEEK   |           | Retrieves the information in the first message on the STREAM head read queue without taking the message off the queue. It is analogous to <i>getmsg()</i> except that this command does not remove the message from the queue. The <i>arg</i> argument points to a <b>strpeek</b> structure.                                                                                                                                                                             |
| 20243 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20244 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20245 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20246 |          |           | The application shall ensure that the <i>maxlen</i> member in the <b>ctlbuf</b> and <b>databuf</b> structures is set to the number of bytes of control information and/or data information, respectively, to retrieve. The <i>flags</i> member may be marked RS_HIPRI or 0, as described by <i>getmsg()</i> . If the process sets <i>flags</i> to RS_HIPRI, for example, I_PEEK shall only look for a high-priority message on the STREAM head read queue.               |
| 20247 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20248 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20249 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20250 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20251 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20252 |          |           | I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the STREAM head read queue, or if the RS_HIPRI flag was set in <i>flags</i> and a high-priority message was not present on the STREAM head read queue. It does not wait for a message to arrive. On return, <b>ctlbuf</b> specifies information in the control buffer, <b>databuf</b> specifies information in the data buffer, and <i>flags</i> contains the value RS_HIPRI or 0. |
| 20253 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20254 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20255 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20256 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20257 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 20258 | I_SRDOPT |           | Sets the read mode using the value of the argument <i>arg</i> . Read modes are described in <i>read()</i> . Valid <i>arg</i> flags are:                                                                                                                                                                                                                                                                                                                                  |
| 20259 |          |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

|       |            |           |                                                                                               |
|-------|------------|-----------|-----------------------------------------------------------------------------------------------|
| 20260 |            | RNORM     | Byte-stream mode, the default.                                                                |
| 20261 |            | RMSGD     | Message-discard mode.                                                                         |
| 20262 |            | RMSGN     | Message-nondiscard mode.                                                                      |
| 20263 |            |           | The bitwise-inclusive OR of RMSGD and RMSGN shall return [EINVAL]. The                        |
| 20264 |            |           | bitwise-inclusive OR of RNORM and either RMSGD or RMSGN shall result in                       |
| 20265 |            |           | the other flag overriding RNORM which is the default.                                         |
| 20266 |            |           | In addition, treatment of control messages by the STREAM head may be                          |
| 20267 |            |           | changed by setting any of the following flags in <i>arg</i> :                                 |
| 20268 |            | RPROTNORM | Fail <i>read()</i> with [EBADMSG] if a message containing a                                   |
| 20269 |            |           | control part is at the front of the STREAM head read queue.                                   |
| 20270 |            | RPROTDAT  | Deliver the control part of a message as data when a                                          |
| 20271 |            |           | process issues a <i>read()</i> .                                                              |
| 20272 |            | RPROTDIS  | Discard the control part of a message, delivering any data                                    |
| 20273 |            |           | portion, when a process issues a <i>read()</i> .                                              |
| 20274 |            |           | The <i>ioctl()</i> function with the I_SRDOPT command shall fail if:                          |
| 20275 |            | [EINVAL]  | The <i>arg</i> argument is not valid.                                                         |
| 20276 | I_GRDOPT   |           | Returns the current read mode setting, as described above, in an <b>int</b> pointed to        |
| 20277 |            |           | by the argument <i>arg</i> . Read modes are described in <i>read()</i> .                      |
| 20278 | I_NREAD    |           | Counts the number of data bytes in the data part of the first message on the                  |
| 20279 |            |           | STREAM head read queue and places this value in the <b>int</b> pointed to by <i>arg</i> .     |
| 20280 |            |           | The return value for the command shall be the number of messages on the                       |
| 20281 |            |           | STREAM head read queue. For example, if 0 is returned in <i>arg</i> , but the <i>ioctl()</i>  |
| 20282 |            |           | return value is greater than 0, this indicates that a zero-length message is next             |
| 20283 |            |           | on the queue.                                                                                 |
| 20284 | I_FDINSERT |           | Creates a message from specified buffer(s), adds information about another                    |
| 20285 |            |           | STREAM, and sends the message downstream. The message contains a                              |
| 20286 |            |           | control part and an optional data part. The data and control parts to be sent                 |
| 20287 |            |           | are distinguished by placement in separate buffers, as described below. The                   |
| 20288 |            |           | <i>arg</i> argument points to a <b>strfdinsert</b> structure.                                 |
| 20289 |            |           | The application shall ensure that the <i>len</i> member in the <b>ctlbuf strbuf</b> structure |
| 20290 |            |           | is set to the size of a <b>t_uscalar_t</b> plus the number of bytes of control                |
| 20291 |            |           | information to be sent with the message. The <i>fildev</i> member specifies the file          |
| 20292 |            |           | descriptor of the other STREAM, and the <i>offset</i> member, which must be                   |
| 20293 |            |           | suitably aligned for use as a <b>t_uscalar_t</b> , specifies the offset from the start of     |
| 20294 |            |           | the control buffer where I_FDINSERT shall store a <b>t_uscalar_t</b> whose                    |
| 20295 |            |           | interpretation is specific to the STREAM end. The application shall ensure that               |
| 20296 |            |           | the <i>len</i> member in the <b>databuf strbuf</b> structure is set to the number of bytes of |
| 20297 |            |           | data information to be sent with the message, or to 0 if no data part is to be                |
| 20298 |            |           | sent.                                                                                         |
| 20299 |            |           | The <i>flags</i> member specifies the type of message to be created. A normal                 |
| 20300 |            |           | message is created if <i>flags</i> is set to 0, and a high-priority message is created if     |
| 20301 |            |           | <i>flags</i> is set to RS_HIPRI. For non-priority messages, I_FDINSERT shall block if         |
| 20302 |            |           | the STREAM write queue is full due to internal flow control conditions. For                   |
| 20303 |            |           | priority messages, I_FDINSERT does not block on this condition. For non-                      |
| 20304 |            |           | priority messages, I_FDINSERT does not block when the write queue is full                     |

20305 and O\_NONBLOCK is set. Instead, it fails and sets *errno* to [EAGAIN].

20306 I\_FDINSERT also blocks, unless prevented by lack of internal resources,  
 20307 waiting for the availability of message blocks in the STREAM, regardless of  
 20308 priority or whether O\_NONBLOCK has been specified. No partial message is  
 20309 sent.

20310 The *ioctl()* function with the I\_FDINSERT command shall fail if:

20311 [EAGAIN] A non-priority message is specified, the O\_NONBLOCK  
 20312 flag is set, and the STREAM write queue is full due to  
 20313 internal flow control conditions.

20314 [EAGAIN] or [ENOSR]  
 20315 Buffers cannot be allocated for the message that is to be  
 20316 created.

20317 [EINVAL] One of the following:

20318 — The *fildev* member of the **strfdinsert** structure is not a  
 20319 valid, open STREAM file descriptor.

20320 — The size of a **t\_uscalar\_t** plus *offset* is greater than the *len*  
 20321 member for the buffer specified through **ctlbuf**.

20322 — The *offset* member does not specify a properly-aligned  
 20323 location in the data buffer.

20324 — An undefined value is stored in *flags*.

20325 [ENXIO] Hangupt received on the STREAM identified by either the  
 20326 *fildev* argument or the *fildev* member of the **strfdinsert**  
 20327 structure.

20328 [ERANGE] The *len* member for the buffer specified through **databuf**  
 20329 does not fall within the range specified by the maximum  
 20330 and minimum packet sizes of the topmost STREAM  
 20331 module; or the *len* member for the buffer specified through  
 20332 **databuf** is larger than the maximum configured size of the  
 20333 data part of a message; or the *len* member for the buffer  
 20334 specified through **ctlbuf** is larger than the maximum  
 20335 configured size of the control part of a message.

20336 I\_STR Constructs an internal STREAMS *ioctl()* message from the data pointed to by  
 20337 *arg*, and sends that message downstream.

20338 This mechanism is provided to send *ioctl()* requests to downstream modules  
 20339 and drivers. It allows information to be sent with *ioctl()*, and returns to the  
 20340 process any information sent upstream by the downstream recipient. I\_STR  
 20341 shall block until the system responds with either a positive or negative  
 20342 acknowledgement message, or until the request times out after some period of  
 20343 time. If the request times out, it shall fail with *errno* set to [ETIME].

20344 At most, one I\_STR can be active on a STREAM. Further I\_STR calls shall  
 20345 block until the active I\_STR completes at the STREAM head. The default  
 20346 timeout interval for these requests is 15 seconds. The O\_NONBLOCK flag has  
 20347 no effect on this call.

20348 To send requests downstream, the application shall ensure that *arg* points to a  
 20349 **strioc** structure.

20350 The *ic\_cmd* member is the internal *ioctl()* command intended for a  
 20351 downstream module or driver and *ic\_timeout* is the number of seconds  
 20352 (−1=infinite, 0=use implementation-defined timeout interval, >0=as specified)  
 20353 an I\_STR request shall wait for acknowledgement before timing out. *ic\_len* is  
 20354 the number of bytes in the data argument, and *ic\_dp* is a pointer to the data  
 20355 argument. The *ic\_len* member has two uses: on input, it contains the length of  
 20356 the data argument passed in, and on return from the command, it contains the  
 20357 number of bytes being returned to the process (the buffer pointed to by *ic\_dp*  
 20358 should be large enough to contain the maximum amount of data that any  
 20359 module or the driver in the STREAM can return).

20360 The STREAM head shall convert the information pointed to by the **strioc**  
 20361 **tl** structure to an internal *ioctl()* command message and send it downstream.

20362 The *ioctl()* function with the I\_STR command shall fail if:

20363 [EAGAIN] or [ENOSR]  
 20364 Unable to allocate buffers for the *ioctl()* message.

20365 [EINVAL] The *ic\_len* member is less than 0 or larger than the  
 20366 maximum configured size of the data part of a message, or  
 20367 *ic\_timeout* is less than −1.

20368 [ENXIO] Hangup received on *fil*des.

20369 [ETIME] A downstream *ioctl()* timed out before acknowledgement  
 20370 was received.

20371 An I\_STR can also fail while waiting for an acknowledgement if a message  
 20372 indicating an error or a hangup is received at the STREAM head. In addition,  
 20373 an error code can be returned in the positive or negative acknowledgement  
 20374 message, in the event the *ioctl()* command sent downstream fails. For these  
 20375 cases, I\_STR shall fail with *errno* set to the value in the message.

20376 I\_SWROPT Sets the write mode using the value of the argument *arg*. Valid bit settings for  
 20377 *arg* are:

20378 SNDZERO Send a zero-length message downstream when a *write()* of  
 20379 0 bytes occurs. To not send a zero-length message when a  
 20380 *write()* of 0 bytes occurs, the application shall ensure that  
 20381 this bit is not set in *arg* (for example, *arg* would be set to 0).

20382 The *ioctl()* function with the I\_SWROPT command shall fail if:

20383 [EINVAL] *arg* is not the above value.

20384 I\_GWROPT Returns the current write mode setting, as described above, in the **int** that is  
 20385 pointed to by the argument *arg*.

20386 I\_SENDFD Creates a new reference to the open file description associated with the file  
 20387 descriptor *arg*, and writes a message on the STREAMS-based pipe *fil*des  
 20388 containing this reference, together with the user ID and group ID of the calling  
 20389 process.

20390 The *ioctl()* function with the I\_SENDFD command shall fail if:

20391 [EAGAIN] The sending STREAM is unable to allocate a message block  
 20392 to contain the file pointer; or the read queue of the receiving  
 20393 STREAM head is full and cannot accept the message sent by  
 20394 I\_SENDFD.

|       |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------|----------|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 20395 |          | [EBADF]             | The <i>arg</i> argument is not a valid, open file descriptor.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 20396 |          | [EINVAL]            | The <i>fildev</i> argument is not connected to a STREAM pipe.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 20397 |          | [ENXIO]             | Hangup received on <i>fildev</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20398 | I_RECVFD |                     | Retrieves the reference to an open file description from a message written to a STREAMS-based pipe using the I_SENDFD command, and allocates a new file descriptor in the calling process that refers to this open file description. The <i>arg</i> argument is a pointer to a <b>strrecvfd</b> data structure as defined in <b>&lt;stropts.h&gt;</b> .                                                                                                                                                                                                                                                                                                                    |
| 20399 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20400 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20401 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20402 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20403 |          |                     | The <i>fd</i> member is a file descriptor. The <i>uid</i> and <i>gid</i> members are the effective user ID and effective group ID, respectively, of the sending process.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 20404 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20405 |          |                     | If O_NONBLOCK is not set, I_RECVFD shall block until a message is present at the STREAM head. If O_NONBLOCK is set, I_RECVFD shall fail with <i>errno</i> set to [EAGAIN] if no message is present at the STREAM head.                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20406 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20407 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20408 |          |                     | If the message at the STREAM head is a message sent by an I_SENDFD, a new file descriptor shall be allocated for the open file descriptor referenced in the message. The new file descriptor is placed in the <i>fd</i> member of the <b>strrecvfd</b> structure pointed to by <i>arg</i> .                                                                                                                                                                                                                                                                                                                                                                                |
| 20409 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20410 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20411 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20412 |          |                     | The <i>ioctl()</i> function with the I_RECVFD command shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 20413 |          | [EAGAIN]            | A message is not present at the STREAM head read queue and the O_NONBLOCK flag is set.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20414 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20415 |          | [EBADMSG]           | The message at the STREAM head read queue is not a message containing a passed file descriptor.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20416 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20417 |          | [EMFILE]            | The process has the maximum number of file descriptors currently open that it is allowed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 20418 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20419 |          | [ENXIO]             | Hangup received on <i>fildev</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20420 | I_LIST   |                     | Allows the process to list all the module names on the STREAM, up to and including the topmost driver name. If <i>arg</i> is a null pointer, the return value shall be the number of modules, including the driver, that are on the STREAM pointed to by <i>fildev</i> . This lets the process allocate enough space for the module names. Otherwise, it should point to a <b>str_list</b> structure.                                                                                                                                                                                                                                                                      |
| 20421 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20422 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20423 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20424 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20425 |          |                     | The <i>sl_nmods</i> member indicates the number of entries the process has allocated in the array. Upon return, the <i>sl_modlist</i> member of the <b>str_list</b> structure shall contain the list of module names, and the number of entries that have been filled into the <i>sl_modlist</i> array is found in the <i>sl_nmods</i> member (the number includes the number of modules including the driver). The return value from <i>ioctl()</i> shall be 0. The entries are filled in starting at the top of the STREAM and continuing downstream until either the end of the STREAM is reached, or the number of requested modules ( <i>sl_nmods</i> ) is satisfied. |
| 20426 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20427 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20428 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20429 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20430 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20431 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20432 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20433 |          |                     | The <i>ioctl()</i> function with the I_LIST command shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20434 |          | [EINVAL]            | The <i>sl_nmods</i> member is less than 1.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20435 |          | [EAGAIN] or [ENOSR] | Unable to allocate buffers.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 20436 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20437 | I_ATMARK |                     | Allows the process to see if the message at the head of the STREAM head read queue is marked by some module downstream. The <i>arg</i> argument determines                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20438 |          |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

|       |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-------|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 20439 |             | how the checking is done when there may be multiple marked messages on the STREAM head read queue. It may take on the following values:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 20440 |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20441 | ANYMARK     | Check if the message is marked.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20442 | LASTMARK    | Check if the message is the last one marked on the queue.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 20443 |             | The bitwise-inclusive OR of the flags ANYMARK and LASTMARK is permitted.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 20444 |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20445 |             | The return value shall be 1 if the mark condition is satisfied; otherwise, the value shall be 0.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 20446 |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20447 |             | The <i>ioctl()</i> function with the L_ATMARK command shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 20448 |             | [EINVAL] Invalid <i>arg</i> value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 20449 | I_CKBAND    | Checks if the message of a given priority band exists on the STREAM head read queue. This shall return 1 if a message of the given priority exists, 0 if no such message exists, or -1 on error. <i>arg</i> should be of type <b>int</b> .                                                                                                                                                                                                                                                                                                                                                                                          |
| 20450 |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20451 |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20452 |             | The <i>ioctl()</i> function with the I_CKBAND command shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 20453 |             | [EINVAL] Invalid <i>arg</i> value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 20454 | I_GETBAND   | Returns the priority band of the first message on the STREAM head read queue in the integer referenced by <i>arg</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 20455 |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20456 |             | The <i>ioctl()</i> function with the I_GETBAND command shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 20457 |             | [ENODATA] No message on the STREAM head read queue.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 20458 | I_CANPUT    | Checks if a certain band is writable. <i>arg</i> is set to the priority band in question. The return value shall be 0 if the band is flow-controlled, 1 if the band is writable, or -1 on error.                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 20459 |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20460 |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20461 |             | The <i>ioctl()</i> function with the I_CANPUT command shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 20462 |             | [EINVAL] Invalid <i>arg</i> value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 20463 | I_SETCLTIME | This request allows the process to set the time the STREAM head shall delay when a STREAM is closing and there is data on the write queues. Before closing each module or driver, if there is data on its write queue, the STREAM head shall delay for the specified amount of time to allow the data to drain. If, after the delay, data is still present, it shall be flushed. The <i>arg</i> argument is a pointer to an integer specifying the number of milliseconds to delay, rounded up to the nearest valid value. If I_SETCLTIME is not performed on a STREAM, an implementation-defined default timeout interval is used. |
| 20464 |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20465 |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20466 |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20467 |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20468 |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20469 |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20470 |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20471 |             | The <i>ioctl()</i> function with the I_SETCLTIME command shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 20472 |             | [EINVAL] Invalid <i>arg</i> value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 20473 | I_GETCLTIME | Returns the close time delay in the integer pointed to by <i>arg</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

20474 **Multiplexed STREAMS Configurations**

20475 The following commands are used for connecting and disconnecting multiplexed STREAMS  
20476 configurations. These commands use an implementation-defined default timeout interval.

20477 **I\_LINK** Connects two STREAMs, where *fildev* is the file descriptor of the STREAM  
20478 connected to the multiplexing driver, and *arg* is the file descriptor of the  
20479 STREAM connected to another driver. The STREAM designated by *arg* is  
20480 connected below the multiplexing driver. I\_LINK requires the multiplexing  
20481 driver to send an acknowledgement message to the STREAM head regarding  
20482 the connection. This call shall return a multiplexer ID number (an identifier  
20483 used to disconnect the multiplexer; see I\_UNLINK) on success, and -1 on  
20484 failure.

20485 The *ioctl()* function with the I\_LINK command shall fail if:

20486 [ENXIO] Hangup received on *fildev*.  
20487 [ETIME] Timeout before acknowledgement message was received at  
20488 STREAM head.  
20489 [EAGAIN] or [ENOSR]  
20490 Unable to allocate STREAMS storage to perform the  
20491 I\_LINK.  
20492 [EBADF] The *arg* argument is not a valid, open file descriptor.  
20493 [EINVAL] The *fildev* argument does not support multiplexing; or *arg* is  
20494 not a STREAM or is already connected downstream from a  
20495 multiplexer; or the specified I\_LINK operation would  
20496 connect the STREAM head in more than one place in the  
20497 multiplexed STREAM.

20498 An I\_LINK can also fail while waiting for the multiplexing driver to  
20499 acknowledge the request, if a message indicating an error or a hangup is  
20500 received at the STREAM head of *fildev*. In addition, an error code can be  
20501 returned in the positive or negative acknowledgement message. For these  
20502 cases, I\_LINK fails with *errno* set to the value in the message.

20503 **I\_UNLINK** Disconnects the two STREAMs specified by *fildev* and *arg*. *fildev* is the file  
20504 descriptor of the STREAM connected to the multiplexing driver. The *arg*  
20505 argument is the multiplexer ID number that was returned by the I\_LINK  
20506 *ioctl()* command when a STREAM was connected downstream from the  
20507 multiplexing driver. If *arg* is MUXID\_ALL, then all STREAMs that were  
20508 connected to *fildev* shall be disconnected. As in I\_LINK, this command  
20509 requires acknowledgement.

20510 The *ioctl()* function with the I\_UNLINK command shall fail if:

20511 [ENXIO] Hangup received on *fildev*.  
20512 [ETIME] Timeout before acknowledgement message was received at  
20513 STREAM head.  
20514 [EAGAIN] or [ENOSR]  
20515 Unable to allocate buffers for the acknowledgement  
20516 message.  
20517 [EINVAL] Invalid multiplexer ID number.

|       |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 20518 |                     | An I_UNLINK can also fail while waiting for the multiplexing driver to acknowledge the request if a message indicating an error or a hangup is received at the STREAM head of <i>filde</i> s. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_UNLINK shall fail with <i>errno</i> set to the value in the message.                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 20519 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20520 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20521 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20522 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20523 | I_PLINK             | Creates a <i>persistent connection</i> between two STREAMs, where <i>filde</i> s is the file descriptor of the STREAM connected to the multiplexing driver, and <i>arg</i> is the file descriptor of the STREAM connected to another driver. This call shall create a persistent connection which can exist even if the file descriptor <i>filde</i> s associated with the upper STREAM to the multiplexing driver is closed. The STREAM designated by <i>arg</i> gets connected via a persistent connection below the multiplexing driver. I_PLINK requires the multiplexing driver to send an acknowledgement message to the STREAM head. This call shall return a multiplexer ID number (an identifier that may be used to disconnect the multiplexer; see I_PUNLINK) on success, and -1 on failure. |
| 20524 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20525 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20526 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20527 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20528 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20529 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20530 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20531 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20532 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20533 |                     | The <i>ioctl</i> () function with the I_PLINK command shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 20534 | [ENXIO]             | Hangup received on <i>filde</i> s.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 20535 | [ETIME]             | Timeout before acknowledgement message was received at STREAM head.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20536 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20537 | [EAGAIN] or [ENOSR] |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20538 |                     | Unable to allocate STREAMS storage to perform the I_PLINK.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 20539 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20540 | [EBADF]             | The <i>arg</i> argument is not a valid, open file descriptor.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 20541 | [EINVAL]            | The <i>filde</i> s argument does not support multiplexing; or <i>arg</i> is not a STREAM or is already connected downstream from a multiplexer; or the specified I_PLINK operation would connect the STREAM head in more than one place in the multiplexed STREAM.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 20542 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20543 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20544 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20545 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20546 |                     | An I_PLINK can also fail while waiting for the multiplexing driver to acknowledge the request, if a message indicating an error or a hangup is received at the STREAM head of <i>filde</i> s. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_PLINK shall fail with <i>errno</i> set to the value in the message.                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 20547 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20548 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20549 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20550 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20551 | I_PUNLINK           | Disconnects the two STREAMs specified by <i>filde</i> s and <i>arg</i> from a persistent connection. The <i>filde</i> s argument is the file descriptor of the STREAM connected to the multiplexing driver. The <i>arg</i> argument is the multiplexer ID number that was returned by the I_PLINK <i>ioctl</i> () command when a STREAM was connected downstream from the multiplexing driver. If <i>arg</i> is MUXID_ALL, then all STREAMs which are persistent connections to <i>filde</i> s shall be disconnected. As in I_PLINK, this command requires the multiplexing driver to acknowledge the request.                                                                                                                                                                                          |
| 20552 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20553 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20554 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20555 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20556 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20557 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20558 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 20559 |                     | The <i>ioctl</i> () function with the I_PUNLINK command shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 20560 | [ENXIO]             | Hangup received on <i>filde</i> s.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 20561 | [ETIME]             | Timeout before acknowledgement message was received at STREAM head.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 20562 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

20563 [EAGAIN] or [ENOSR]  
 20564 Unable to allocate buffers for the acknowledgement  
 20565 message.  
 20566 [EINVAL] Invalid multiplexer ID number.  
 20567 An I\_PUNLINK can also fail while waiting for the multiplexing driver to  
 20568 acknowledge the request if a message indicating an error or a hangup is  
 20569 received at the STREAM head of *fildev*. In addition, an error code can be  
 20570 returned in the positive or negative acknowledgement message. For these  
 20571 cases, I\_PUNLINK shall fail with *errno* set to the value in the message.

20572 **RETURN VALUE**

20573 Upon successful completion, *ioctl()* shall return a value other than  $-1$  that depends upon the  
 20574 STREAMS device control function. Otherwise, it shall return  $-1$  and set *errno* to indicate the  
 20575 error.

20576 **ERRORS**

20577 Under the following general conditions, *ioctl()* shall fail if:

20578 [EBADF] The *fildev* argument is not a valid open file descriptor.  
 20579 [EINTR] A signal was caught during the *ioctl()* operation.  
 20580 [EINVAL] The STREAM or multiplexer referenced by *fildev* is linked (directly or  
 20581 indirectly) downstream from a multiplexer.

20582 If an underlying device driver detects an error, then *ioctl()* shall fail if:

20583 [EINVAL] The *request* or *arg* argument is not valid for this device.  
 20584 [EIO] Some physical I/O error has occurred.  
 20585 [ENOTTY] The *fildev* argument is not associated with a STREAMS device that accepts  
 20586 control functions.  
 20587 [ENXIO] The *request* and *arg* arguments are valid for this device driver, but the service  
 20588 requested cannot be performed on this particular sub-device.  
 20589 [ENODEV] The *fildev* argument refers to a valid STREAMS device, but the corresponding  
 20590 device driver does not support the *ioctl()* function.

20591 If a STREAM is connected downstream from a multiplexer, any *ioctl()* command except  
 20592 I\_UNLINK and I\_PUNLINK shall set *errno* to [EINVAL].

20593 **EXAMPLES**

20594 None.

20595 **APPLICATION USAGE**

20596 The implementation-defined timeout interval for STREAMS has historically been 15 seconds.

20597 **RATIONALE**

20598 None.

20599 **FUTURE DIRECTIONS**

20600 None.

20601 **SEE ALSO**

20602 Section 2.6 (on page 38), *close()*, *fcntl()*, *getmsg()*, *open()*, *pipe()*, *poll()*, *putmsg()*, *read()*,  
 20603 *sigaction()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**stropts.h**>

20604 **CHANGE HISTORY**

20605 First released in Issue 4, Version 2.

20606 **Issue 5**

20607 Moved from X/OPEN UNIX extension to BASE.

20608 **Issue 6**

20609 The Open Group Corrigendum U028/4 is applied, correcting text in the L\_FDINSERT [EINVAL]  
20610 case to refer to *ctlbuf*.

20611 This function is marked as part of the XSI STREAMS Option Group.

20612 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20613 **NAME**

20614           isalnum — test for an alphanumeric character

20615 **SYNOPSIS**

20616           #include <ctype.h>

20617           int isalnum(int c);

20618 **DESCRIPTION**

20619 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
20620 conflict between the requirements described here and the ISO C standard is unintentional. This  
20621 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20622       The *isalnum()* function shall test whether *c* is a character of class **alpha** or **digit** in the program's  
20623 current locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

20624       The *c* argument is an **int**, the value of which the application shall ensure is representable as an  
20625 **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the  
20626 behavior is undefined.

20627 **RETURN VALUE**

20628       The *isalnum()* function shall return non-zero if *c* is an alphanumeric character; otherwise, it shall  
20629 return 0.

20630 **ERRORS**

20631       No errors are defined.

20632 **EXAMPLES**

20633       None.

20634 **APPLICATION USAGE**

20635       To ensure applications portability, especially across natural languages, only this function and  
20636 those listed in the SEE ALSO section should be used for character classification.

20637 **RATIONALE**

20638       None.

20639 **FUTURE DIRECTIONS**

20640       None.

20641 **SEE ALSO**

20642       *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*,  
20643 *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <ctype.h>,  
20644 <stdio.h>

20645 **CHANGE HISTORY**

20646       First released in Issue 1. Derived from Issue 1 of the SVID.

20647 **Issue 6**

20648       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20649 **NAME**

20650 isalpha — test for an alphabetic character

20651 **SYNOPSIS**

20652 #include <ctype.h>

20653 int isalpha(int c);

20654 **DESCRIPTION**

20655 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
20656 conflict between the requirements described here and the ISO C standard is unintentional. This  
20657 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20658 The *isalpha()* function shall test whether *c* is a character of class **alpha** in the program's current  
20659 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

20660 The *c* argument is an **int**, the value of which the application shall ensure is representable as an  
20661 **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the  
20662 behavior is undefined.

20663 **RETURN VALUE**

20664 The *isalpha()* function shall return non-zero if *c* is an alphabetic character; otherwise, it shall  
20665 return 0.

20666 **ERRORS**

20667 No errors are defined.

20668 **EXAMPLES**

20669 None.

20670 **APPLICATION USAGE**

20671 To ensure applications portability, especially across natural languages, only this function and  
20672 those listed in the SEE ALSO section should be used for character classification.

20673 **RATIONALE**

20674 None.

20675 **FUTURE DIRECTIONS**

20676 None.

20677 **SEE ALSO**

20678 *isalnum()*, *isctrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
20679 *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,  
20680 <ctype.h>, <stdio.h>

20681 **CHANGE HISTORY**

20682 First released in Issue 1. Derived from Issue 1 of the SVID.

20683 **Issue 6**

20684 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20685 **NAME**

20686        isascii — test for a 7-bit US-ASCII character

20687 **SYNOPSIS**

20688 xSI        #include &lt;ctype.h&gt;

20689        int isascii(int c);

20690

20691 **DESCRIPTION**20692        The *isascii()* function shall test whether *c* is a 7-bit US-ASCII character code.20693        The *isascii()* function is defined on all integer values.20694 **RETURN VALUE**20695        The *isascii()* function shall return non-zero if *c* is a 7-bit US-ASCII character code between 0 and octal 0177 inclusive; otherwise, it shall return 0.20697 **ERRORS**

20698        No errors are defined.

20699 **EXAMPLES**

20700        None.

20701 **APPLICATION USAGE**

20702        None.

20703 **RATIONALE**

20704        None.

20705 **FUTURE DIRECTIONS**

20706        None.

20707 **SEE ALSO**

20708        The Base Definitions volume of IEEE Std 1003.1-2001, &lt;ctype.h&gt;

20709 **CHANGE HISTORY**

20710        First released in Issue 1. Derived from Issue 1 of the SVID.

20711 **NAME**20712 isastream — test a file descriptor (**STREAMS**)20713 **SYNOPSIS**

20714 XSR #include &lt;stropts.h&gt;

20715 int isastream(int *fildev*);

20716

20717 **DESCRIPTION**20718 The *isastream()* function shall test whether *fildev*, an open file descriptor, is associated with a  
20719 STREAMS-based file.20720 **RETURN VALUE**20721 Upon successful completion, *isastream()* shall return 1 if *fildev* refers to a STREAMS-based file  
20722 and 0 if not. Otherwise, *isastream()* shall return -1 and set *errno* to indicate the error.20723 **ERRORS**20724 The *isastream()* function shall fail if:20725 [EBADF] The *fildev* argument is not a valid open file descriptor.20726 **EXAMPLES**

20727 None.

20728 **APPLICATION USAGE**

20729 None.

20730 **RATIONALE**

20731 None.

20732 **FUTURE DIRECTIONS**

20733 None.

20734 **SEE ALSO**

20735 The Base Definitions volume of IEEE Std 1003.1-2001, &lt;stropts.h&gt;

20736 **CHANGE HISTORY**

20737 First released in Issue 4, Version 2.

20738 **Issue 5**

20739 Moved from X/OPEN UNIX extension to BASE.

20740 **NAME**

20741           isatty — test for a terminal device

20742 **SYNOPSIS**

20743           #include <unistd.h>

20744           int isatty(int *fildev*);

20745 **DESCRIPTION**

20746           The *isatty()* function shall test whether *fildev*, an open file descriptor, is associated with a  
20747           terminal device.

20748 **RETURN VALUE**

20749           The *isatty()* function shall return 1 if *fildev* is associated with a terminal; otherwise, it shall return  
20750           0 and may set *errno* to indicate the error.

20751 **ERRORS**

20752           The *isatty()* function may fail if:

20753           [EBADF]           The *fildev* argument is not a valid open file descriptor.

20754           [ENOTTY]          The *fildev* argument is not associated with a terminal.

20755 **EXAMPLES**

20756           None.

20757 **APPLICATION USAGE**

20758           The *isatty()* function does not necessarily indicate that a human being is available for interaction  
20759           via *fildev*. It is quite possible that non-terminal devices are connected to the communications  
20760           line.

20761 **RATIONALE**

20762           None.

20763 **FUTURE DIRECTIONS**

20764           None.

20765 **SEE ALSO**

20766           The Base Definitions volume of IEEE Std 1003.1-2001, <unistd.h>

20767 **CHANGE HISTORY**

20768           First released in Issue 1. Derived from Issue 1 of the SVID.

20769 **Issue 6**

20770           The following new requirements on POSIX implementations derive from alignment with the  
20771           Single UNIX Specification:

- 20772           • The optional setting of *errno* to indicate an error is added.
- 20773           • The [EBADF] and [ENOTTY] optional error conditions are added.

20774 **NAME**

20775 isblank — test for a blank character

20776 **SYNOPSIS**

20777 #include &lt;ctype.h&gt;

20778 int isblank(int c);

20779 **DESCRIPTION**

20780 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
20781 conflict between the requirements described here and the ISO C standard is unintentional. This  
20782 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20783 The *isblank()* function shall test whether *c* is a character of class **blank** in the program's current  
20784 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

20785 The *c* argument is a type **int**, the value of which the application shall ensure is a character  
20786 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
20787 any other value, the behavior is undefined.

20788 **RETURN VALUE**20789 The *isblank()* function shall return non-zero if *c* is a <blank>; otherwise, it shall return 0.20790 **ERRORS**

20791 No errors are defined.

20792 **EXAMPLES**

20793 None.

20794 **APPLICATION USAGE**

20795 To ensure applications portability, especially across natural languages, only this function and  
20796 those listed in the SEE ALSO section should be used for character classification.

20797 **RATIONALE**

20798 None.

20799 **FUTURE DIRECTIONS**

20800 None.

20801 **SEE ALSO**

20802 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
20803 *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,  
20804 <ctype.h>

20805 **CHANGE HISTORY**

20806 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

20807 **NAME**

20808           isctrl — test for a control character

20809 **SYNOPSIS**

20810           #include &lt;ctype.h&gt;

20811           int isctrl(int c);

20812 **DESCRIPTION**

20813 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
20814       conflict between the requirements described here and the ISO C standard is unintentional. This  
20815       volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20816       The *isctrl()* function shall test whether *c* is a character of class **cntrl** in the program's current  
20817       locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

20818       The *c* argument is a type **int**, the value of which the application shall ensure is a character  
20819       representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
20820       any other value, the behavior is undefined.

20821 **RETURN VALUE**20822       The *isctrl()* function shall return non-zero if *c* is a control character; otherwise, it shall return 0.20823 **ERRORS**

20824       No errors are defined.

20825 **EXAMPLES**

20826       None.

20827 **APPLICATION USAGE**

20828       To ensure applications portability, especially across natural languages, only this function and  
20829       those listed in the SEE ALSO section should be used for character classification.

20830 **RATIONALE**

20831       None.

20832 **FUTURE DIRECTIONS**

20833       None.

20834 **SEE ALSO**

20835       *isalnum()*, *isalpha()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
20836       *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,  
20837       <ctype.h>

20838 **CHANGE HISTORY**

20839       First released in Issue 1. Derived from Issue 1 of the SVID.

20840 **Issue 6**

20841       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20842 **NAME**

20843            isdigit — test for a decimal digit

20844 **SYNOPSIS**

20845            #include <ctype.h>

20846            int isdigit(int c);

20847 **DESCRIPTION**

20848 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
20849 conflict between the requirements described here and the ISO C standard is unintentional. This  
20850 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20851            The *isdigit()* function shall test whether *c* is a character of class **digit** in the program's current  
20852 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

20853            The *c* argument is an **int**, the value of which the application shall ensure is a character  
20854 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
20855 any other value, the behavior is undefined.

20856 **RETURN VALUE**

20857            The *isdigit()* function shall return non-zero if *c* is a decimal digit; otherwise, it shall return 0.

20858 **ERRORS**

20859            No errors are defined.

20860 **EXAMPLES**

20861            None.

20862 **APPLICATION USAGE**

20863            To ensure applications portability, especially across natural languages, only this function and  
20864 those listed in the SEE ALSO section should be used for character classification.

20865 **RATIONALE**

20866            None.

20867 **FUTURE DIRECTIONS**

20868            None.

20869 **SEE ALSO**

20870            *isalnum()*, *isalpha()*, *iscntrl()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
20871 *isxdigit()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <ctype.h>

20872 **CHANGE HISTORY**

20873            First released in Issue 1. Derived from Issue 1 of the SVID.

20874 **Issue 6**

20875            The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20876 **NAME**

20877           isfinite — test for finite value

20878 **SYNOPSIS**

20879           #include &lt;math.h&gt;

20880           int isfinite(real-floating x);

20881 **DESCRIPTION**

20882 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
20883       conflict between the requirements described here and the ISO C standard is unintentional. This  
20884       volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20885       The *isfinite()* macro shall determine whether its argument has a finite value (zero, subnormal, or  
20886       normal, and not infinite or NaN). First, an argument represented in a format wider than its  
20887       semantic type is converted to its semantic type. Then determination is based on the type of the  
20888       argument.

20889 **RETURN VALUE**20890       The *isfinite()* macro shall return a non-zero value if and only if its argument has a finite value.20891 **ERRORS**

20892       No errors are defined.

20893 **EXAMPLES**

20894       None.

20895 **APPLICATION USAGE**

20896       None.

20897 **RATIONALE**

20898       None.

20899 **FUTURE DIRECTIONS**

20900       None.

20901 **SEE ALSO**

20902       *fpclassify()*, *isinf()*, *isnan()*, *isnormal()*, *signbit()*, the Base Definitions volume of  
20903       IEEE Std 1003.1-2001 <math.h>

20904 **CHANGE HISTORY**

20905       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

20906 **NAME**

20907 isgraph — test for a visible character

20908 **SYNOPSIS**

20909 #include &lt;ctype.h&gt;

20910 int isgraph(int c);

20911 **DESCRIPTION**

20912 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
20913 conflict between the requirements described here and the ISO C standard is unintentional. This  
20914 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20915 The *isgraph()* function shall test whether *c* is a character of class **graph** in the program's current  
20916 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

20917 The *c* argument is an **int**, the value of which the application shall ensure is a character  
20918 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
20919 any other value, the behavior is undefined.

20920 **RETURN VALUE**

20921 The *isgraph()* function shall return non-zero if *c* is a character with a visible representation;  
20922 otherwise, it shall return 0.

20923 **ERRORS**

20924 No errors are defined.

20925 **EXAMPLES**

20926 None.

20927 **APPLICATION USAGE**

20928 To ensure applications portability, especially across natural languages, only this function and  
20929 those listed in the SEE ALSO section should be used for character classification.

20930 **RATIONALE**

20931 None.

20932 **FUTURE DIRECTIONS**

20933 None.

20934 **SEE ALSO**

20935 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*,  
20936 *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <ctype.h>

20937 **CHANGE HISTORY**

20938 First released in Issue 1. Derived from Issue 1 of the SVID.

20939 **Issue 6**

20940 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20941 **NAME**

20942           isgreater — test if x greater than y

20943 **SYNOPSIS**

20944           #include &lt;math.h&gt;

20945           int isgreater(real-floating x, real-floating y);

20946 **DESCRIPTION**

20947 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
20948       conflict between the requirements described here and the ISO C standard is unintentional. This  
20949       volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20950       The *isgreater()* macro shall determine whether its first argument is greater than its second  
20951       argument. The value of *isgreater(x, y)* shall be equal to  $(x) > (y)$ ; however, unlike  $(x) > (y)$ ,  
20952       *isgreater(x, y)* shall not raise the invalid floating-point exception when x and y are unordered.

20953 **RETURN VALUE**20954       Upon successful completion, the *isgreater()* macro shall return the value of  $(x) > (y)$ .

20955       If x or y is NaN, 0 shall be returned.

20956 **ERRORS**

20957       No errors are defined.

20958 **EXAMPLES**

20959       None.

20960 **APPLICATION USAGE**

20961       The relational and equality operators support the usual mathematical relationships between  
20962       numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
20963       greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
20964       when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
20965       unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
20966       version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
20967       without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
20968       indicates that the argument shall be an expression of **real-floating** type.

20969 **RATIONALE**

20970       None.

20971 **FUTURE DIRECTIONS**

20972       None.

20973 **SEE ALSO**

20974       *isgreaterequal()*, *isless()*, *islessequal()*, *islessgreater()*, *isunordered()*, the Base Definitions volume of  
20975       IEEE Std 1003.1-2001 <math.h>

20976 **CHANGE HISTORY**

20977       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

20978 **NAME**

20979           isgreaterequal — test if *x* is greater than or equal to *y*

20980 **SYNOPSIS**

20981           #include <math.h>

20982           int isgreaterequal(real-floating *x*, real-floating *y*);

20983 **DESCRIPTION**

20984 *CX*       The functionality described on this reference page is aligned with the ISO C standard. Any  
20985 conflict between the requirements described here and the ISO C standard is unintentional. This  
20986 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

20987       The *isgreaterequal()* macro shall determine whether its first argument is greater than or equal to  
20988 its second argument. The value of *isgreaterequal(x, y)* shall be equal to  $(x) \geq (y)$ ; however, unlike  
20989  $(x) \geq (y)$ , *isgreaterequal(x, y)* shall not raise the invalid floating-point exception when *x* and *y* are  
20990 unordered.

20991 **RETURN VALUE**

20992       Upon successful completion, the *isgreaterequal()* macro shall return the value of  $(x) \geq (y)$ .

20993       If *x* or *y* is NaN, 0 shall be returned.

20994 **ERRORS**

20995       No errors are defined.

20996 **EXAMPLES**

20997       None.

20998 **APPLICATION USAGE**

20999       The relational and equality operators support the usual mathematical relationships between  
21000 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
21001 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
21002 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
21003 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
21004 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
21005 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
21006 indicates that the argument shall be an expression of **real-floating** type.

21007 **RATIONALE**

21008       None.

21009 **FUTURE DIRECTIONS**

21010       None.

21011 **SEE ALSO**

21012       *isgreater()*, *isless()*, *islessequal()*, *islessgreater()*, *isunordered()*, the Base Definitions volume of  
21013 IEEE Std 1003.1-2001 <math.h>

21014 **CHANGE HISTORY**

21015       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21016 **NAME**

21017        isinf — test for infinity

21018 **SYNOPSIS**

21019        #include &lt;math.h&gt;

21020        int isinf(real-floating x);

21021 **DESCRIPTION**

21022 *cx*        The functionality described on this reference page is aligned with the ISO C standard. Any  
21023        conflict between the requirements described here and the ISO C standard is unintentional. This  
21024        volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21025        The *isinf()* macro shall determine whether its argument value is an infinity (positive or  
21026        negative). First, an argument represented in a format wider than its semantic type is converted  
21027        to its semantic type. Then determination is based on the type of the argument.

21028 **RETURN VALUE**21029        The *isinf()* macro shall return a non-zero value if and only if its argument has an infinite value.21030 **ERRORS**

21031        No errors are defined.

21032 **EXAMPLES**

21033        None.

21034 **APPLICATION USAGE**

21035        None.

21036 **RATIONALE**

21037        None.

21038 **FUTURE DIRECTIONS**

21039        None.

21040 **SEE ALSO**

21041        *fpclassify()*, *isfinite()*, *isnan()*, *isnormal()*, *signbit()*, the Base Definitions volume of  
21042        IEEE Std 1003.1-2001 <math.h>

21043 **CHANGE HISTORY**

21044        First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21045 **NAME**

21046           isless — test if *x* is less than *y*

21047 **SYNOPSIS**

21048           #include <math.h>

21049           int isless(real-floating *x*, real-floating *y*);

21050 **DESCRIPTION**

21051 *cx*       The functionality described on this reference page is aligned with the ISO C standard. Any  
21052 conflict between the requirements described here and the ISO C standard is unintentional. This  
21053 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21054       The *isless()* macro shall determine whether its first argument is less than its second argument.  
21055       The value of *isless(x, y)* shall be equal to  $(x) < (y)$ ; however, unlike  $(x) < (y)$ , *isless(x, y)* shall not  
21056 raise the invalid floating-point exception when *x* and *y* are unordered.

21057 **RETURN VALUE**

21058       Upon successful completion, the *isless()* macro shall return the value of  $(x) < (y)$ .

21059       If *x* or *y* is NaN, 0 shall be returned.

21060 **ERRORS**

21061       No errors are defined.

21062 **EXAMPLES**

21063       None.

21064 **APPLICATION USAGE**

21065       The relational and equality operators support the usual mathematical relationships between  
21066 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
21067 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
21068 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
21069 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
21070 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
21071 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
21072 indicates that the argument shall be an expression of **real-floating** type.

21073 **RATIONALE**

21074       None.

21075 **FUTURE DIRECTIONS**

21076       None.

21077 **SEE ALSO**

21078       *isgreater()*, *isgreaterequal()*, *islessequal()*, *islessgreater()*, *isunordered()*, the Base Definitions volume  
21079 of IEEE Std 1003.1-2001, <math.h>

21080 **CHANGE HISTORY**

21081       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21082 **NAME**

21083 islessequal — test if *x* is less than or equal to *y*

21084 **SYNOPSIS**

21085 #include <math.h>

21086 int islessequal(real-floating *x*, real-floating *y*);

21087 **DESCRIPTION**

21088 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
21089 conflict between the requirements described here and the ISO C standard is unintentional. This  
21090 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21091 The *islessequal()* macro shall determine whether its first argument is less than or equal to its  
21092 second argument. The value of *islessequal(x, y)* shall be equal to  $(x) \leq (y)$ ; however, unlike  
21093  $(x) \leq (y)$ , *islessequal(x, y)* shall not raise the invalid floating-point exception when *x* and *y* are  
21094 unordered.

21095 **RETURN VALUE**

21096 Upon successful completion, the *islessequal()* macro shall return the value of  $(x) \leq (y)$ .

21097 If *x* or *y* is NaN, 0 shall be returned.

21098 **ERRORS**

21099 No errors are defined.

21100 **EXAMPLES**

21101 None.

21102 **APPLICATION USAGE**

21103 The relational and equality operators support the usual mathematical relationships between  
21104 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
21105 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
21106 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
21107 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
21108 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
21109 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
21110 indicates that the argument shall be an expression of **real-floating** type.

21111 **RATIONALE**

21112 None.

21113 **FUTURE DIRECTIONS**

21114 None.

21115 **SEE ALSO**

21116 *isgreater()*, *isgreaterequal()*, *isless()*, *islessgreater()*, *isunordered()*, the Base Definitions volume of  
21117 IEEE Std 1003.1-2001 <math.h>

21118 **CHANGE HISTORY**

21119 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

## 21120 NAME

21121 islessgreater — test if  $x$  is less than or greater than  $y$

## 21122 SYNOPSIS

21123 #include <math.h>

21124 int islessgreater(real-floating  $x$ , real-floating  $y$ );

## 21125 DESCRIPTION

21126 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
21127 conflict between the requirements described here and the ISO C standard is unintentional. This  
21128 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21129 The *islessgreater()* macro shall determine whether its first argument is less than or greater than  
21130 its second argument. The *islessgreater*( $x$ ,  $y$ ) macro is similar to  $(x) < (y) \mid \mid (x) > (y)$ ; however,  
21131 *islessgreater*( $x$ ,  $y$ ) shall not raise the invalid floating-point exception when  $x$  and  $y$  are unordered  
21132 (nor shall it evaluate  $x$  and  $y$  twice).

## 21133 RETURN VALUE

21134 Upon successful completion, the *islessgreater()* macro shall return the value of  
21135  $(x) < (y) \mid \mid (x) > (y)$ .

21136 If  $x$  or  $y$  is NaN, 0 shall be returned.

## 21137 ERRORS

21138 No errors are defined.

## 21139 EXAMPLES

21140 None.

## 21141 APPLICATION USAGE

21142 The relational and equality operators support the usual mathematical relationships between  
21143 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
21144 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
21145 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
21146 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
21147 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
21148 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
21149 indicates that the argument shall be an expression of **real-floating** type.

## 21150 RATIONALE

21151 None.

## 21152 FUTURE DIRECTIONS

21153 None.

## 21154 SEE ALSO

21155 *isgreater()*, *isgreaterequal()*, *isless()*, *islessequal()*, *isunordered()*, the Base Definitions volume of  
21156 IEEE Std 1003.1-2001 <math.h>

## 21157 CHANGE HISTORY

21158 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21159 **NAME**

21160           islower — test for a lowercase letter

21161 **SYNOPSIS**

21162           #include &lt;ctype.h&gt;

21163           int islower(int c);

21164 **DESCRIPTION**

21165 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 21166 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21167 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21168       The *islower()* function shall test whether *c* is a character of class **lower** in the program's current  
 21169 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

21170       The *c* argument is an **int**, the value of which the application shall ensure is a character  
 21171 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
 21172 any other value, the behavior is undefined.

21173 **RETURN VALUE**21174       The *islower()* function shall return non-zero if *c* is a lowercase letter; otherwise, it shall return 0.21175 **ERRORS**

21176       No errors are defined.

21177 **EXAMPLES**21178           **Testing for a Lowercase Letter**

21179       The following example tests whether the value is a lowercase letter, based on the locale of the  
 21180 user, then uses it as part of a key value.

```

21181 #include <ctype.h>
21182 #include <stdlib.h>
21183 #include <locale.h>
21184 ...
21185 char *keyst;
21186 int elementlen, len;
21187 char c;
21188 ...
21189 setlocale(LC_ALL, "");
21190 ...
21191 len = 0;
21192 while (len < elementlen) {
21193 c = (char) (rand() % 256);
21194 ...
21195 if (islower(c))
21196 keyst[len++] = c;
21197 }
21198 ...
```

21199 **APPLICATION USAGE**

21200       To ensure applications portability, especially across natural languages, only this function and  
 21201 those listed in the SEE ALSO section should be used for character classification.

21202 **RATIONALE**

21203           None.

21204 **FUTURE DIRECTIONS**

21205           None.

21206 **SEE ALSO**

21207           *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
21208           *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,  
21209           <ctype.h>

21210 **CHANGE HISTORY**

21211           First released in Issue 1. Derived from Issue 1 of the SVID.

21212 **Issue 6**

21213           The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21214           An example is added.

21215 **NAME**21216        **isnan** — test for a NaN21217 **SYNOPSIS**

21218        #include &lt;math.h&gt;

21219        int isnan(real-floating x);

21220 **DESCRIPTION**

21221 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
21222        conflict between the requirements described here and the ISO C standard is unintentional. This  
21223        volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21224        The *isnan()* macro shall determine whether its argument value is a NaN. First, an argument  
21225        represented in a format wider than its semantic type is converted to its semantic type. Then  
21226        determination is based on the type of the argument.

21227 **RETURN VALUE**21228        The *isnan()* macro shall return a non-zero value if and only if its argument has a NaN value.21229 **ERRORS**

21230        No errors are defined.

21231 **EXAMPLES**

21232        None.

21233 **APPLICATION USAGE**

21234        None.

21235 **RATIONALE**

21236        None.

21237 **FUTURE DIRECTIONS**

21238        None.

21239 **SEE ALSO**

21240        *fpclassify()*, *isfinite()*, *isinf()*, *isnormal()*, *signbit()*, the Base Definitions volume of  
21241        IEEE Std 1003.1-2001, <math.h>

21242 **CHANGE HISTORY**

21243        First released in Issue 3.

21244 **Issue 5**

21245        The DESCRIPTION is updated to indicate the return value when NaN is not supported. This  
21246        text was previously published in the APPLICATION USAGE section.

21247 **Issue 6**

21248        Re-written for alignment with the ISO/IEC 9899:1999 standard.

21249 **NAME**

21250           isnormal — test for a normal value

21251 **SYNOPSIS**

21252           #include &lt;math.h&gt;

21253           int isnormal(real-floating x);

21254 **DESCRIPTION**

21255 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
21256       conflict between the requirements described here and the ISO C standard is unintentional. This  
21257       volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21258       The *isnormal()* macro shall determine whether its argument value is normal (neither zero,  
21259       subnormal, infinite, nor NaN). First, an argument represented in a format wider than its  
21260       semantic type is converted to its semantic type. Then determination is based on the type of the  
21261       argument.

21262 **RETURN VALUE**

21263       The *isnormal()* macro shall return a non-zero value if and only if its argument has a normal  
21264       value.

21265 **ERRORS**

21266       No errors are defined.

21267 **EXAMPLES**

21268       None.

21269 **APPLICATION USAGE**

21270       None.

21271 **RATIONALE**

21272       None.

21273 **FUTURE DIRECTIONS**

21274       None.

21275 **SEE ALSO**

21276       *fpclassify()*, *isfinite()*, *isinf()*, *isnan()*, *signbit()*, the Base Definitions volume of  
21277       IEEE Std 1003.1-2001, <math.h>

21278 **CHANGE HISTORY**

21279       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21280 **NAME**

21281 isprint — test for a printable character

21282 **SYNOPSIS**

21283 #include &lt;ctype.h&gt;

21284 int isprint(int c);

21285 **DESCRIPTION**

21286 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
21287 conflict between the requirements described here and the ISO C standard is unintentional. This  
21288 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21289 The *isprint()* function shall test whether *c* is a character of class **print** in the program's current  
21290 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

21291 The *c* argument is an **int**, the value of which the application shall ensure is a character  
21292 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
21293 any other value, the behavior is undefined.

21294 **RETURN VALUE**

21295 The *isprint()* function shall return non-zero if *c* is a printable character; otherwise, it shall return  
21296 0.

21297 **ERRORS**

21298 No errors are defined.

21299 **EXAMPLES**

21300 None.

21301 **APPLICATION USAGE**

21302 To ensure applications portability, especially across natural languages, only this function and  
21303 those listed in the SEE ALSO section should be used for character classification.

21304 **RATIONALE**

21305 None.

21306 **FUTURE DIRECTIONS**

21307 None.

21308 **SEE ALSO**

21309 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *ispunct()*, *isspace()*, *isupper()*,  
21310 *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,  
21311 <ctype.h>

21312 **CHANGE HISTORY**

21313 First released in Issue 1. Derived from Issue 1 of the SVID.

21314 **Issue 6**

21315 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21316 **NAME**

21317           ispunct — test for a punctuation character

21318 **SYNOPSIS**

21319           #include <ctype.h>

21320           int ispunct(int c);

21321 **DESCRIPTION**

21322 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
21323 conflict between the requirements described here and the ISO C standard is unintentional. This  
21324 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21325       The *ispunct()* function shall test whether *c* is a character of class **punct** in the program's current  
21326 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

21327       The *c* argument is an **int**, the value of which the application shall ensure is a character  
21328 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
21329 any other value, the behavior is undefined.

21330 **RETURN VALUE**

21331       The *ispunct()* function shall return non-zero if *c* is a punctuation character; otherwise, it shall  
21332 return 0.

21333 **ERRORS**

21334       No errors are defined.

21335 **EXAMPLES**

21336       None.

21337 **APPLICATION USAGE**

21338       To ensure applications portability, especially across natural languages, only this function and  
21339 those listed in the SEE ALSO section should be used for character classification.

21340 **RATIONALE**

21341       None.

21342 **FUTURE DIRECTIONS**

21343       None.

21344 **SEE ALSO**

21345       *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *isspace()*, *isupper()*, *isxdigit()*,  
21346 *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <ctype.h>

21347 **CHANGE HISTORY**

21348       First released in Issue 1. Derived from Issue 1 of the SVID.

21349 **Issue 6**

21350       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21351 **NAME**

21352           isspace — test for a white-space character

21353 **SYNOPSIS**

21354           #include <ctype.h>

21355           int isspace(int c);

21356 **DESCRIPTION**

21357 *cx*       The functionality described on this reference page is aligned with the ISO C standard. Any  
21358           conflict between the requirements described here and the ISO C standard is unintentional. This  
21359           volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21360           The *isspace()* function shall test whether *c* is a character of class **space** in the program's current  
21361           locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

21362           The *c* argument is an **int**, the value of which the application shall ensure is a character  
21363           representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
21364           any other value, the behavior is undefined.

21365 **RETURN VALUE**

21366           The *isspace()* function shall return non-zero if *c* is a white-space character; otherwise, it shall  
21367           return 0.

21368 **ERRORS**

21369           No errors are defined.

21370 **EXAMPLES**

21371           None.

21372 **APPLICATION USAGE**

21373           To ensure applications portability, especially across natural languages, only this function and  
21374           those listed in the SEE ALSO section should be used for character classification.

21375 **RATIONALE**

21376           None.

21377 **FUTURE DIRECTIONS**

21378           None.

21379 **SEE ALSO**

21380           *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isupper()*,  
21381           *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale,  
21382           <ctype.h>

21383 **CHANGE HISTORY**

21384           First released in Issue 1. Derived from Issue 1 of the SVID.

21385 **Issue 6**

21386           The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21387 **NAME**

21388 isunordered — test if arguments are unordered

21389 **SYNOPSIS**

21390 #include <math.h>

21391 int isunordered(real-floating x, real-floating y);

21392 **DESCRIPTION**

21393 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
21394 conflict between the requirements described here and the ISO C standard is unintentional. This  
21395 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21396 The *isunordered()* macro shall determine whether its arguments are unordered.

21397 **RETURN VALUE**

21398 Upon successful completion, the *isunordered()* macro shall return 1 if its arguments are  
21399 unordered, and 0 otherwise.

21400 If *x* or *y* is NaN, 0 shall be returned.

21401 **ERRORS**

21402 No errors are defined.

21403 **EXAMPLES**

21404 None.

21405 **APPLICATION USAGE**

21406 The relational and equality operators support the usual mathematical relationships between  
21407 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
21408 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
21409 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
21410 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
21411 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
21412 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
21413 indicates that the argument shall be an expression of **real-floating** type.

21414 **RATIONALE**

21415 None.

21416 **FUTURE DIRECTIONS**

21417 None.

21418 **SEE ALSO**

21419 *isgreater()*, *isgreaterequal()*, *isless()*, *islessequal()*, *islessgreater()*, the Base Definitions volume of  
21420 IEEE Std 1003.1-2001, <math.h>

21421 **CHANGE HISTORY**

21422 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21423 **NAME**

21424 isupper — test for an uppercase letter

21425 **SYNOPSIS**

21426 #include &lt;ctype.h&gt;

21427 int isupper(int c);

21428 **DESCRIPTION**

21429 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
21430 conflict between the requirements described here and the ISO C standard is unintentional. This  
21431 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21432 The *isupper()* function shall test whether *c* is a character of class **upper** in the program's current  
21433 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

21434 The *c* argument is an **int**, the value of which the application shall ensure is a character  
21435 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
21436 any other value, the behavior is undefined.

21437 **RETURN VALUE**21438 The *isupper()* function shall return non-zero if *c* is an uppercase letter; otherwise, it shall return 0.21439 **ERRORS**

21440 No errors are defined.

21441 **EXAMPLES**

21442 None.

21443 **APPLICATION USAGE**

21444 To ensure applications portability, especially across natural languages, only this function and  
21445 those listed in the SEE ALSO section should be used for character classification.

21446 **RATIONALE**

21447 None.

21448 **FUTURE DIRECTIONS**

21449 None.

21450 **SEE ALSO**

21451 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isxdigit()*,  
21452 *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <ctype.h>

21453 **CHANGE HISTORY**

21454 First released in Issue 1. Derived from Issue 1 of the SVID.

21455 **Issue 6**

21456 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21457 **NAME**

21458           iswalnum — test for an alphanumeric wide-character code

21459 **SYNOPSIS**

21460           #include &lt;wctype.h&gt;

21461           int iswalnum(wint\_t wc);

21462 **DESCRIPTION**

21463 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 21464 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21465 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21466       The *iswalnum()* function shall test whether *wc* is a wide-character code representing a character  
 21467 of class **alpha** or **digit** in the program's current locale; see the Base Definitions volume of  
 21468 IEEE Std 1003.1-2001, Chapter 7, Locale.

21469       The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
 21470 code corresponding to a valid character in the current locale, or equal to the value of the macro  
 21471 WEOF. If the argument has any other value, the behavior is undefined.

21472 **RETURN VALUE**

21473       The *iswalnum()* function shall return non-zero if *wc* is an alphanumeric wide-character code;  
 21474 otherwise, it shall return 0.

21475 **ERRORS**

21476       No errors are defined.

21477 **EXAMPLES**

21478       None.

21479 **APPLICATION USAGE**

21480       To ensure applications portability, especially across natural languages, only this function and  
 21481 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21482 **RATIONALE**

21483       None.

21484 **FUTURE DIRECTIONS**

21485       None.

21486 **SEE ALSO**

21487       *iswalphabet()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
 21488 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
 21489 IEEE Std 1003.1-2001, Chapter 7, Locale, <stdio.h>, <wchar.h>, <wctype.h>

21490 **CHANGE HISTORY**

21491       First released as a World-wide Portability Interface in Issue 4.

21492 **Issue 5**

21493       The following change has been made in this issue for alignment with  
 21494 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21495       • The SYNOPSIS has been changed to indicate that this function and associated data types are  
 21496 now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

21497 **Issue 6**

21498       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21499 **NAME**

21500 iswalpha — test for an alphabetic wide-character code

21501 **SYNOPSIS**

21502 #include &lt;wctype.h&gt;

21503 int iswalpha(wint\_t wc);

21504 **DESCRIPTION**

21505 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 21506 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21507 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21508 The *iswalpha()* function shall test whether *wc* is a wide-character code representing a character of  
 21509 class **alpha** in the program's current locale; see the Base Definitions volume of  
 21510 IEEE Std 1003.1-2001, Chapter 7, Locale.

21511 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
 21512 code corresponding to a valid character in the current locale, or equal to the value of the macro  
 21513 WEOF. If the argument has any other value, the behavior is undefined.

21514 **RETURN VALUE**

21515 The *iswalpha()* function shall return non-zero if *wc* is an alphabetic wide-character code;  
 21516 otherwise, it shall return 0.

21517 **ERRORS**

21518 No errors are defined.

21519 **EXAMPLES**

21520 None.

21521 **APPLICATION USAGE**

21522 To ensure applications portability, especially across natural languages, only this function and  
 21523 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21524 **RATIONALE**

21525 None.

21526 **FUTURE DIRECTIONS**

21527 None.

21528 **SEE ALSO**

21529 *iswalnum()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
 21530 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
 21531 IEEE Std 1003.1-2001, Chapter 7, Locale, <stdio.h>, <wchar.h>, <wctype.h>

21532 **CHANGE HISTORY**

21533 First released in Issue 4.

21534 **Issue 5**

21535 The following change has been made in this issue for alignment with  
 21536 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21537 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
 21538 now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

21539 **Issue 6**

21540 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21541 **NAME**

21542           iswblank — test for a blank wide-character code

21543 **SYNOPSIS**

21544           #include &lt;wctype.h&gt;

21545           int iswblank(wint\_t wc);

21546 **DESCRIPTION**

21547 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
21548       conflict between the requirements described here and the ISO C standard is unintentional. This  
21549       volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21550       The *iswblank()* function shall test whether *wc* is a wide-character code representing a character of  
21551       class **blank** in the program's current locale; see the Base Definitions volume of  
21552       IEEE Std 1003.1-2001, Chapter 7, Locale.

21553       The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
21554       code corresponding to a valid character in the current locale, or equal to the value of the macro  
21555       WEOF. If the argument has any other value, the behavior is undefined.

21556 **RETURN VALUE**

21557       The *iswblank()* function shall return non-zero if *wc* is a blank wide-character code; otherwise, it  
21558       shall return 0.

21559 **ERRORS**

21560       No errors are defined.

21561 **EXAMPLES**

21562       None.

21563 **APPLICATION USAGE**

21564       To ensure applications portability, especially across natural languages, only this function and  
21565       those listed in the SEE ALSO section should be used for classification of wide-character codes.

21566 **RATIONALE**

21567       None.

21568 **FUTURE DIRECTIONS**

21569       None.

21570 **SEE ALSO**

21571       *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
21572       *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
21573       IEEE Std 1003.1-2001, Chapter 7, Locale, <stdio.h>, <wchar.h>, <wctype.h>

21574 **CHANGE HISTORY**

21575       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21576 **NAME**

21577 iswcntrl — test for a control wide-character code

21578 **SYNOPSIS**

21579 #include &lt;wctype.h&gt;

21580 int iswcntrl(wint\_t wc);

21581 **DESCRIPTION**

21582 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 21583 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21584 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21585 The *iswcntrl()* function shall test whether *wc* is a wide-character code representing a character of  
 21586 class **cntrl** in the program's current locale; see the Base Definitions volume of  
 21587 IEEE Std 1003.1-2001, Chapter 7, Locale.

21588 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
 21589 code corresponding to a valid character in the current locale, or equal to the value of the macro  
 21590 WEOF. If the argument has any other value, the behavior is undefined.

21591 **RETURN VALUE**

21592 The *iswcntrl()* function shall return non-zero if *wc* is a control wide-character code; otherwise, it  
 21593 shall return 0.

21594 **ERRORS**

21595 No errors are defined.

21596 **EXAMPLES**

21597 None.

21598 **APPLICATION USAGE**

21599 To ensure applications portability, especially across natural languages, only this function and  
 21600 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21601 **RATIONALE**

21602 None.

21603 **FUTURE DIRECTIONS**

21604 None.

21605 **SEE ALSO**

21606 *iswalnum()*, *iswalpha()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
 21607 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
 21608 IEEE Std 1003.1-2001, Chapter 7, Locale, <**wchar.h**>, <**wctype.h**>

21609 **CHANGE HISTORY**

21610 First released in Issue 4.

21611 **Issue 5**

21612 The following change has been made in this issue for alignment with  
 21613 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21614 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
 21615 now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

21616 **Issue 6**

21617 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

## 21618 NAME

21619 iswctype — test character for a specified class

## 21620 SYNOPSIS

21621 #include &lt;wctype.h&gt;

21622 int iswctype(wint\_t *wc*, wctype\_t *charclass*);

## 21623 DESCRIPTION

21624 *CX* The functionality described on this reference page is aligned with the ISO C standard. Any  
 21625 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21626 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21627 The *iswctype()* function shall determine whether the wide-character code *wc* has the character  
 21628 class *charclass*, returning true or false. The *iswctype()* function is defined on WEOF and wide-  
 21629 character codes corresponding to the valid character encodings in the current locale. If the *wc*  
 21630 argument is not in the domain of the function, the result is undefined. If the value of *charclass* is  
 21631 invalid (that is, not obtained by a call to *wctype()* or *charclass* is invalidated by a subsequent call  
 21632 to *setlocale()* that has affected category *LC\_CTYPE*) the result is unspecified.

## 21633 RETURN VALUE

21634 The *iswctype()* function shall return non-zero (true) if and only if *wc* has the property described  
 21635 *CX* by *charclass*. If *charclass* is 0, *iswctype()* shall return 0.

## 21636 ERRORS

21637 No errors are defined.

## 21638 EXAMPLES

## 21639 Testing for a Valid Character

```
21640 #include <wctype.h>
21641 ...
21642 int yes_or_no;
21643 wint_t wc;
21644 wctype_t valid_class;
21645 ...
21646 if ((valid_class=wctype("vowel")) == (wctype_t)0)
21647 /* Invalid character class. */
21648 yes_or_no=iswctype(wc,valid_class);
```

## 21649 APPLICATION USAGE

21650 The twelve strings "alnum", "alpha", "blank", "cntrl", "digit", "graph", "lower",  
 21651 "print", "punct", "space", "upper", and "xdigit" are reserved for the standard  
 21652 character classes. In the table below, the functions in the left column are equivalent to the  
 21653 functions in the right column.

|       |                     |                                      |
|-------|---------------------|--------------------------------------|
| 21654 | <i>iswalnum(wc)</i> | <i>iswctype(wc, wctype("alnum"))</i> |
| 21655 | <i>iswalpha(wc)</i> | <i>iswctype(wc, wctype("alpha"))</i> |
| 21656 | <i>iswblank(wc)</i> | <i>iswctype(wc, wctype("blank"))</i> |
| 21657 | <i>iswcntrl(wc)</i> | <i>iswctype(wc, wctype("cntrl"))</i> |
| 21658 | <i>iswdigit(wc)</i> | <i>iswctype(wc, wctype("digit"))</i> |
| 21659 | <i>iswgraph(wc)</i> | <i>iswctype(wc, wctype("graph"))</i> |
| 21660 | <i>iswlower(wc)</i> | <i>iswctype(wc, wctype("lower"))</i> |
| 21661 | <i>iswprint(wc)</i> | <i>iswctype(wc, wctype("print"))</i> |
| 21662 | <i>iswpunct(wc)</i> | <i>iswctype(wc, wctype("punct"))</i> |
| 21663 | <i>iswspace(wc)</i> | <i>iswctype(wc, wctype("space"))</i> |

21664            `iswupper(wc)`      `iswctype(wc, wctype("upper"))`  
21665            `iswxdigit(wc)`      `iswctype(wc, wctype("xdigit"))`

21666 **RATIONALE**

21667            None.

21668 **FUTURE DIRECTIONS**

21669            None.

21670 **SEE ALSO**

21671            `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`,  
21672            `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `wctype()`, the Base Definitions volume of  
21673            IEEE Std 1003.1-2001, `<wchar.h>`, `<wctype.h>`

21674 **CHANGE HISTORY**

21675            First released as World-wide Portability Interfaces in Issue 4.

21676 **Issue 5**

21677            The following change has been made in this issue for alignment with  
21678            ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21679            • The SYNOPSIS has been changed to indicate that this function and associated data types are  
21680            now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

21681 **Issue 6**

21682            The behavior of  $n=0$  is now described.

21683            An example is added.

21684            A new function, `iswblank()`, is added to the list in the APPLICATION USAGE.

21685 **NAME**

21686           iswdigit — test for a decimal digit wide-character code

21687 **SYNOPSIS**

21688           #include <wctype.h>

21689           int iswdigit(wint\_t wc);

21690 **DESCRIPTION**

21691 *cx*       The functionality described on this reference page is aligned with the ISO C standard. Any  
21692 conflict between the requirements described here and the ISO C standard is unintentional. This  
21693 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21694       The *iswdigit()* function shall test whether *wc* is a wide-character code representing a character of  
21695 class **digit** in the program's current locale; see the Base Definitions volume of  
21696 IEEE Std 1003.1-2001, Chapter 7, Locale.

21697       The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
21698 code corresponding to a valid character in the current locale, or equal to the value of the macro  
21699 WEOF. If the argument has any other value, the behavior is undefined.

21700 **RETURN VALUE**

21701       The *iswdigit()* function shall return non-zero if *wc* is a decimal digit wide-character code;  
21702 otherwise, it shall return 0.

21703 **ERRORS**

21704       No errors are defined.

21705 **EXAMPLES**

21706       None.

21707 **APPLICATION USAGE**

21708       To ensure applications portability, especially across natural languages, only this function and  
21709 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21710 **RATIONALE**

21711       None.

21712 **FUTURE DIRECTIONS**

21713       None.

21714 **SEE ALSO**

21715       *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
21716 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
21717 IEEE Std 1003.1-2001, Chapter 7, Locale, <**wchar.h**>, <**wctype.h**>

21718 **CHANGE HISTORY**

21719       First released in Issue 4.

21720 **Issue 5**

21721       The following change has been made in this issue for alignment with  
21722 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21723       • The SYNOPSIS has been changed to indicate that this function and associated data types are  
21724       now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

21725 **Issue 6**

21726       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21727 **NAME**

21728           iswgraph — test for a visible wide-character code

21729 **SYNOPSIS**

21730           #include &lt;wctype.h&gt;

21731           int iswgraph(wint\_t wc);

21732 **DESCRIPTION**

21733 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 21734 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21735 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21736       The *iswgraph()* function shall test whether *wc* is a wide-character code representing a character  
 21737 of class **graph** in the program's current locale; see the Base Definitions volume of  
 21738 IEEE Std 1003.1-2001, Chapter 7, Locale.

21739       The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
 21740 code corresponding to a valid character in the current locale, or equal to the value of the macro  
 21741 WEOF. If the argument has any other value, the behavior is undefined.

21742 **RETURN VALUE**

21743       The *iswgraph()* function shall return non-zero if *wc* is a wide-character code with a visible  
 21744 representation; otherwise, it shall return 0.

21745 **ERRORS**

21746       No errors are defined.

21747 **EXAMPLES**

21748       None.

21749 **APPLICATION USAGE**

21750       To ensure applications portability, especially across natural languages, only this function and  
 21751 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21752 **RATIONALE**

21753       None.

21754 **FUTURE DIRECTIONS**

21755       None.

21756 **SEE ALSO**

21757       *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
 21758 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
 21759 IEEE Std 1003.1-2001, Chapter 7, Locale, <**wchar.h**>, <**wctype.h**>

21760 **CHANGE HISTORY**

21761       First released in Issue 4.

21762 **Issue 5**

21763       The following change has been made in this issue for alignment with  
 21764 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21765       • The SYNOPSIS has been changed to indicate that this function and associated data types are  
 21766       now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

21767 **Issue 6**

21768       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21769 **NAME**

21770 iswlower — test for a lowercase letter wide-character code

21771 **SYNOPSIS**

21772 #include <wctype.h>

21773 int iswlower(wint\_t wc);

21774 **DESCRIPTION**

21775 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
21776 conflict between the requirements described here and the ISO C standard is unintentional. This  
21777 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21778 The *iswlower()* function shall test whether *wc* is a wide-character code representing a character  
21779 of class **lower** in the program's current locale; see the Base Definitions volume of  
21780 IEEE Std 1003.1-2001, Chapter 7, Locale.

21781 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
21782 code corresponding to a valid character in the current locale, or equal to the value of the macro  
21783 WEOF. If the argument has any other value, the behavior is undefined.

21784 **RETURN VALUE**

21785 The *iswlower()* function shall return non-zero if *wc* is a lowercase letter wide-character code;  
21786 otherwise, it shall return 0.

21787 **ERRORS**

21788 No errors are defined.

21789 **EXAMPLES**

21790 None.

21791 **APPLICATION USAGE**

21792 To ensure applications portability, especially across natural languages, only this function and  
21793 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21794 **RATIONALE**

21795 None.

21796 **FUTURE DIRECTIONS**

21797 None.

21798 **SEE ALSO**

21799 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswprint()*, *iswpunct()*,  
21800 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
21801 IEEE Std 1003.1-2001, Chapter 7, Locale, <**wchar.h**>, <**wctype.h**>

21802 **CHANGE HISTORY**

21803 First released in Issue 4.

21804 **Issue 5**

21805 The following change has been made in this issue for alignment with  
21806 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21807 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
21808 now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

21809 **Issue 6**

21810 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21811 **NAME**

21812           iswprint — test for a printable wide-character code

21813 **SYNOPSIS**

21814           #include &lt;wctype.h&gt;

21815           int iswprint(wint\_t wc);

21816 **DESCRIPTION**

21817 *cx*       The functionality described on this reference page is aligned with the ISO C standard. Any  
 21818 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21819 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21820       The *iswprint()* function shall test whether *wc* is a wide-character code representing a character of  
 21821 class **print** in the program's current locale; see the Base Definitions volume of  
 21822 IEEE Std 1003.1-2001, Chapter 7, Locale.

21823       The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
 21824 code corresponding to a valid character in the current locale, or equal to the value of the macro  
 21825 WEOF. If the argument has any other value, the behavior is undefined.

21826 **RETURN VALUE**

21827       The *iswprint()* function shall return non-zero if *wc* is a printable wide-character code; otherwise,  
 21828 it shall return 0.

21829 **ERRORS**

21830       No errors are defined.

21831 **EXAMPLES**

21832       None.

21833 **APPLICATION USAGE**

21834       To ensure applications portability, especially across natural languages, only this function and  
 21835 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21836 **RATIONALE**

21837       None.

21838 **FUTURE DIRECTIONS**

21839       None.

21840 **SEE ALSO**

21841       *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswpunct()*,  
 21842 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
 21843 IEEE Std 1003.1-2001, Chapter 7, Locale, <**wchar.h**>, <**wctype.h**>

21844 **CHANGE HISTORY**

21845       First released in Issue 4.

21846 **Issue 5**

21847       The following change has been made in this issue for alignment with  
 21848 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21849       • The SYNOPSIS has been changed to indicate that this function and associated data types are  
 21850 now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

21851 **Issue 6**

21852       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21853 **NAME**

21854 iswpunct — test for a punctuation wide-character code

21855 **SYNOPSIS**

21856 #include <wctype.h>

21857 int iswpunct(wint\_t wc);

21858 **DESCRIPTION**

21859 *cx* The functionality described on this reference page is aligned with the ISO C standard. Any  
21860 conflict between the requirements described here and the ISO C standard is unintentional. This  
21861 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21862 The *iswpunct()* function shall test whether *wc* is a wide-character code representing a character  
21863 of class **punct** in the program's current locale; see the Base Definitions volume of  
21864 IEEE Std 1003.1-2001, Chapter 7, Locale.

21865 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
21866 code corresponding to a valid character in the current locale, or equal to the value of the macro  
21867 WEOF. If the argument has any other value, the behavior is undefined.

21868 **RETURN VALUE**

21869 The *iswpunct()* function shall return non-zero if *wc* is a punctuation wide-character code;  
21870 otherwise, it shall return 0.

21871 **ERRORS**

21872 No errors are defined.

21873 **EXAMPLES**

21874 None.

21875 **APPLICATION USAGE**

21876 To ensure applications portability, especially across natural languages, only this function and  
21877 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21878 **RATIONALE**

21879 None.

21880 **FUTURE DIRECTIONS**

21881 None.

21882 **SEE ALSO**

21883 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
21884 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
21885 IEEE Std 1003.1-2001, Chapter 7, Locale, <**wchar.h**>, <**wctype.h**>

21886 **CHANGE HISTORY**

21887 First released in Issue 4.

21888 **Issue 5**

21889 The following change has been made in this issue for alignment with  
21890 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21891 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
21892 now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

21893 **Issue 6**

21894 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21895 **NAME**

21896           iswspace — test for a white-space wide-character code

21897 **SYNOPSIS**

21898           #include &lt;wctype.h&gt;

21899           int iswspace(wint\_t wc);

21900 **DESCRIPTION**

21901 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
 21902       conflict between the requirements described here and the ISO C standard is unintentional. This  
 21903       volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21904       The *iswspace()* function shall test whether *wc* is a wide-character code representing a character of  
 21905       class **space** in the program's current locale; see the Base Definitions volume of  
 21906       IEEE Std 1003.1-2001, Chapter 7, Locale.

21907       The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
 21908       code corresponding to a valid character in the current locale, or equal to the value of the macro  
 21909       WEOF. If the argument has any other value, the behavior is undefined.

21910 **RETURN VALUE**

21911       The *iswspace()* function shall return non-zero if *wc* is a white-space wide-character code;  
 21912       otherwise, it shall return 0.

21913 **ERRORS**

21914       No errors are defined.

21915 **EXAMPLES**

21916       None.

21917 **APPLICATION USAGE**

21918       To ensure applications portability, especially across natural languages, only this function and  
 21919       those listed in the SEE ALSO section should be used for classification of wide-character codes.

21920 **RATIONALE**

21921       None.

21922 **FUTURE DIRECTIONS**

21923       None.

21924 **SEE ALSO**

21925       *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
 21926       *iswpunct()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
 21927       IEEE Std 1003.1-2001, Chapter 7, Locale, <**wchar.h**>, <**wctype.h**>

21928 **CHANGE HISTORY**

21929       First released in Issue 4.

21930 **Issue 5**

21931       The following change has been made in this issue for alignment with  
 21932       ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21933           • The SYNOPSIS has been changed to indicate that this function and associated data types are  
 21934           now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

21935 **Issue 6**

21936       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21937 **NAME**

21938 iswupper — test for an uppercase letter wide-character code

21939 **SYNOPSIS**

21940 #include <wctype.h>

21941 int iswupper(wint\_t wc);

21942 **DESCRIPTION**

21943 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
21944 conflict between the requirements described here and the ISO C standard is unintentional. This  
21945 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

21946 The *iswupper()* function shall test whether *wc* is a wide-character code representing a character  
21947 of class **upper** in the program's current locale; see the Base Definitions volume of  
21948 IEEE Std 1003.1-2001, Chapter 7, Locale.

21949 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
21950 code corresponding to a valid character in the current locale, or equal to the value of the macro  
21951 WEOF. If the argument has any other value, the behavior is undefined.

21952 **RETURN VALUE**

21953 The *iswupper()* function shall return non-zero if *wc* is an uppercase letter wide-character code;  
21954 otherwise, it shall return 0.

21955 **ERRORS**

21956 No errors are defined.

21957 **EXAMPLES**

21958 None.

21959 **APPLICATION USAGE**

21960 To ensure applications portability, especially across natural languages, only this function and  
21961 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21962 **RATIONALE**

21963 None.

21964 **FUTURE DIRECTIONS**

21965 None.

21966 **SEE ALSO**

21967 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
21968 *iswpunct()*, *iswspace()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
21969 IEEE Std 1003.1-2001, Chapter 7, Locale, <**wchar.h**>, <**wctype.h**>

21970 **CHANGE HISTORY**

21971 First released in Issue 4.

21972 **Issue 5**

21973 The following change has been made in this issue for alignment with  
21974 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21975 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
21976 now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

21977 **Issue 6**

21978 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21979 **NAME**

21980 iswxdigit — test for a hexadecimal digit wide-character code

21981 **SYNOPSIS**

21982 #include &lt;wctype.h&gt;

21983 int iswxdigit(wint\_t wc);

21984 **DESCRIPTION**21985 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
21986 conflict between the requirements described here and the ISO C standard is unintentional. This  
21987 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.21988 The *iswxdigit()* function shall test whether *wc* is a wide-character code representing a character  
21989 of class **xdigit** in the program's current locale; see the Base Definitions volume of  
21990 IEEE Std 1003.1-2001, Chapter 7, Locale.21991 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
21992 code corresponding to a valid character in the current locale, or equal to the value of the macro  
21993 WEOF. If the argument has any other value, the behavior is undefined.21994 **RETURN VALUE**21995 The *iswxdigit()* function shall return non-zero if *wc* is a hexadecimal digit wide-character code;  
21996 otherwise, it shall return 0.21997 **ERRORS**

21998 No errors are defined.

21999 **EXAMPLES**

22000 None.

22001 **APPLICATION USAGE**22002 To ensure applications portability, especially across natural languages, only this function and  
22003 those listed in the SEE ALSO section should be used for classification of wide-character codes.22004 **RATIONALE**

22005 None.

22006 **FUTURE DIRECTIONS**

22007 None.

22008 **SEE ALSO**22009 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
22010 *iswpunct()*, *iswspace()*, *iswupper()*, *setlocale()*, the Base Definitions volume of  
22011 IEEE Std 1003.1-2001, Chapter 7, Locale, <**wchar.h**>, <**wctype.h**>22012 **CHANGE HISTORY**

22013 First released in Issue 4.

22014 **Issue 5**22015 The following change has been made in this issue for alignment with  
22016 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 22017
- The SYNOPSIS has been changed to indicate that this function and associated data types are  
22018 now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

22019 **Issue 6**

22020 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

22021 **NAME**

22022 isxdigit — test for a hexadecimal digit

22023 **SYNOPSIS**

22024 #include &lt;ctype.h&gt;

22025 int isxdigit(int c);

22026 **DESCRIPTION**

22027 **cx** The functionality described on this reference page is aligned with the ISO C standard. Any  
22028 conflict between the requirements described here and the ISO C standard is unintentional. This  
22029 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22030 The *isxdigit()* function shall test whether *c* is a character of class **xdigit** in the program's current  
22031 locale; see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale.

22032 The *c* argument is an **int**, the value of which the application shall ensure is a character  
22033 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
22034 any other value, the behavior is undefined.

22035 **RETURN VALUE**

22036 The *isxdigit()* function shall return non-zero if *c* is a hexadecimal digit; otherwise, it shall return  
22037 0.

22038 **ERRORS**

22039 No errors are defined.

22040 **EXAMPLES**

22041 None.

22042 **APPLICATION USAGE**

22043 To ensure applications portability, especially across natural languages, only this function and  
22044 those listed in the SEE ALSO section should be used for character classification.

22045 **RATIONALE**

22046 None.

22047 **FUTURE DIRECTIONS**

22048 None.

22049 **SEE ALSO**

22050 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
22051 the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <ctype.h>

22052 **CHANGE HISTORY**

22053 First released in Issue 1. Derived from Issue 1 of the SVID.

22054 **Issue 6**

22055 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

22056 **NAME**

22057       j0, j1, jn — Bessel functions of the first kind

22058 **SYNOPSIS**

```
22059 xSI #include <math.h>
22060 double j0(double x);
22061 double j1(double x);
22062 double jn(int n, double x);
22063
```

22064 **DESCRIPTION**

22065       The *j0()*, *j1()*, and *jn()* functions shall compute Bessel functions of *x* of the first kind of orders 0, 1, and *n*, respectively.

22067       An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(*FE\_ALL\_EXCEPT*) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(*FE\_INVALID* | *FE\_DIVBYZERO* | *FE\_OVERFLOW* | *FE\_UNDERFLOW*) is non-zero, an error has occurred.

22071 **RETURN VALUE**

22072       Upon successful completion, these functions shall return the relevant Bessel value of *x* of the first kind.

22074       If the *x* argument is too large in magnitude, or the correct result would cause underflow, 0 shall be returned and a range error may occur.

22076       If *x* is NaN, a NaN shall be returned.

22077 **ERRORS**

22078       These functions may fail if:

22079       Range Error       The value of *x* was too large in magnitude, or an underflow occurred.

22080       If the integer expression (*math\_errhandling* & *MATH\_ERRNO*) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math\_errhandling* & *MATH\_ERREXCEPT*) is non-zero, then the underflow floating-point exception shall be raised.

22084       No other errors shall occur.

22085 **EXAMPLES**

22086       None.

22087 **APPLICATION USAGE**

22088       On error, the expressions (*math\_errhandling* & *MATH\_ERRNO*) and (*math\_errhandling* & *MATH\_ERREXCEPT*) are independent of each other, but at least one of them must be non-zero.

22090 **RATIONALE**

22091       None.

22092 **FUTURE DIRECTIONS**

22093       None.

22094 **SEE ALSO**

22095       *feclearexcept()*, *fetestexcept()*, *isnan()*, *y0()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions, <**math.h**>

22097 **CHANGE HISTORY**

22098 First released in Issue 1. Derived from Issue 1 of the SVID.

22099 **Issue 5**

22100 The DESCRIPTION is updated to indicate how an application should check for an error. This  
22101 text was previously published in the APPLICATION USAGE section.

22102 **Issue 6**

22103 The may fail [EDOM] error is removed for the case for NaN.

22104 The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling  
22105 with the ISO/IEC 9899:1999 standard.

22106 **NAME**

22107           jrand48 — generate a uniformly distributed pseudo-random long signed integer

22108 **SYNOPSIS**

22109 xSI       #include &lt;stdlib.h&gt;

22110           long jrand48(unsigned short xsubi[3]);

22111

22112 **DESCRIPTION**22113           Refer to *drand48()*.

## 22114 NAME

22115 kill — send a signal to a process or a group of processes

## 22116 SYNOPSIS

22117 cx #include <signal.h>

22118 int kill(pid\_t pid, int sig);

22119

## 22120 DESCRIPTION

22121 The *kill()* function shall send a signal to a process or a group of processes specified by *pid*. The  
 22122 signal to be sent is specified by *sig* and is either one from the list given in <signal.h> or 0. If *sig* is  
 22123 0 (the null signal), error checking is performed but no signal is actually sent. The null signal can  
 22124 be used to check the validity of *pid*.

22125 For a process to have permission to send a signal to a process designated by *pid*, unless the  
 22126 sending process has appropriate privileges, the real or effective user ID of the sending process  
 22127 shall match the real or saved set-user-ID of the receiving process.

22128 If *pid* is greater than 0, *sig* shall be sent to the process whose process ID is equal to *pid*.

22129 If *pid* is 0, *sig* shall be sent to all processes (excluding an unspecified set of system processes)  
 22130 whose process group ID is equal to the process group ID of the sender, and for which the  
 22131 process has permission to send a signal.

22132 If *pid* is  $-1$ , *sig* shall be sent to all processes (excluding an unspecified set of system processes) for  
 22133 which the process has permission to send that signal.

22134 If *pid* is negative, but not  $-1$ , *sig* shall be sent to all processes (excluding an unspecified set of  
 22135 system processes) whose process group ID is equal to the absolute value of *pid*, and for which  
 22136 the process has permission to send a signal.

22137 If the value of *pid* causes *sig* to be generated for the sending process, and if *sig* is not blocked for  
 22138 the calling thread and if no other thread has *sig* unblocked or is waiting in a *sigwait()* function  
 22139 for *sig*, either *sig* or at least one pending unblocked signal shall be delivered to the sending  
 22140 thread before *kill()* returns.

22141 The user ID tests described above shall not be applied when sending SIGCONT to a process that  
 22142 is a member of the same session as the sending process.

22143 An implementation that provides extended security controls may impose further  
 22144 implementation-defined restrictions on the sending of signals, including the null signal. In  
 22145 particular, the system may deny the existence of some or all of the processes specified by *pid*.

22146 The *kill()* function is successful if the process has permission to send *sig* to any of the processes  
 22147 specified by *pid*. If *kill()* fails, no signal shall be sent.

## 22148 RETURN VALUE

22149 Upon successful completion, 0 shall be returned. Otherwise,  $-1$  shall be returned and *errno* set to  
 22150 indicate the error.

## 22151 ERRORS

22152 The *kill()* function shall fail if:

22153 [EINVAL] The value of the *sig* argument is an invalid or unsupported signal number.

22154 [EPERM] The process does not have permission to send the signal to any receiving  
 22155 process.

22156 [ESRCH] No process or process group can be found corresponding to that specified by  
 22157 *pid*.

22158 **EXAMPLES**

22159 None.

22160 **APPLICATION USAGE**

22161 None.

22162 **RATIONALE**

22163 The semantics for permission checking for *kill()* differed between System V and most other  
 22164 implementations, such as Version 7 or 4.3 BSD. The semantics chosen for this volume of  
 22165 IEEE Std 1003.1-2001 agree with System V. Specifically, a set-user-ID process cannot protect  
 22166 itself against signals (or at least not against SIGKILL) unless it changes its real user ID. This  
 22167 choice allows the user who starts an application to send it signals even if it changes its effective  
 22168 user ID. The other semantics give more power to an application that wants to protect itself from  
 22169 the user who ran it.

22170 Some implementations provide semantic extensions to the *kill()* function when the absolute  
 22171 value of *pid* is greater than some maximum, or otherwise special, value. Negative values are a  
 22172 flag to *kill()*. Since most implementations return [ESRCH] in this case, this behavior is not  
 22173 included in this volume of IEEE Std 1003.1-2001, although a conforming implementation could  
 22174 provide such an extension.

22175 The implementation-defined processes to which a signal cannot be sent may include the  
 22176 scheduler or *init*.

22177 There was initially strong sentiment to specify that, if *pid* specifies that a signal be sent to the  
 22178 calling process and that signal is not blocked, that signal would be delivered before *kill()*  
 22179 returns. This would permit a process to call *kill()* and be guaranteed that the call never return.  
 22180 However, historical implementations that provide only the *signal()* function make only the  
 22181 weaker guarantee in this volume of IEEE Std 1003.1-2001, because they only deliver one signal  
 22182 each time a process enters the kernel. Modifications to such implementations to support the  
 22183 *sigaction()* function generally require entry to the kernel following return from a signal-catching  
 22184 function, in order to restore the signal mask. Such modifications have the effect of satisfying the  
 22185 stronger requirement, at least when *sigaction()* is used, but not necessarily when *signal()* is used.  
 22186 The developers of this volume of IEEE Std 1003.1-2001 considered making the stronger  
 22187 requirement except when *signal()* is used, but felt this would be unnecessarily complex.  
 22188 Implementors are encouraged to meet the stronger requirement whenever possible. In practice,  
 22189 the weaker requirement is the same, except in the rare case when two signals arrive during a  
 22190 very short window. This reasoning also applies to a similar requirement for *sigprocmask()*.

22191 In 4.2 BSD, the SIGCONT signal can be sent to any descendant process regardless of user-ID  
 22192 security checks. This allows a job control shell to continue a job even if processes in the job have  
 22193 altered their user IDs (as in the *su* command). In keeping with the addition of the concept of  
 22194 sessions, similar functionality is provided by allowing the SIGCONT signal to be sent to any  
 22195 process in the same session regardless of user ID security checks. This is less restrictive than BSD  
 22196 in the sense that ancestor processes (in the same session) can now be the recipient. It is more  
 22197 restrictive than BSD in the sense that descendant processes that form new sessions are now  
 22198 subject to the user ID checks. A similar relaxation of security is not necessary for the other job  
 22199 control signals since those signals are typically sent by the terminal driver in recognition of  
 22200 special characters being typed; the terminal driver bypasses all security checks.

22201 In secure implementations, a process may be restricted from sending a signal to a process having  
 22202 a different security label. In order to prevent the existence or nonexistence of a process from  
 22203 being used as a covert channel, such processes should appear nonexistent to the sender; that is,  
 22204 [ESRCH] should be returned, rather than [EPERM], if *pid* refers only to such processes.

- 22205 Existing implementations vary on the result of a *kill()* with *pid* indicating an inactive process (a  
 22206 terminated process that has not been waited for by its parent). Some indicate success on such a  
 22207 call (subject to permission checking), while others give an error of [ESRCH]. Since the definition  
 22208 of process lifetime in this volume of IEEE Std 1003.1-2001 covers inactive processes, the  
 22209 [ESRCH] error as described is inappropriate in this case. In particular, this means that an  
 22210 application cannot have a parent process check for termination of a particular child with *kill()*.  
 22211 (Usually this is done with the null signal; this can be done reliably with *waitpid()*.)
- 22212 There is some belief that the name *kill()* is misleading, since the function is not always intended  
 22213 to cause process termination. However, the name is common to all historical implementations,  
 22214 and any change would be in conflict with the goal of minimal changes to existing application  
 22215 code.
- 22216 **FUTURE DIRECTIONS**
- 22217 None.
- 22218 **SEE ALSO**
- 22219 *getpid()*, *raise()*, *setsid()*, *sigaction()*, *sigqueue()*, the Base Definitions volume of  
 22220 IEEE Std 1003.1-2001, <signal.h>, <sys/types.h>
- 22221 **CHANGE HISTORY**
- 22222 First released in Issue 1. Derived from Issue 1 of the SVID.
- 22223 **Issue 5**
- 22224 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.
- 22225 **Issue 6**
- 22226 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.
- 22227 The following new requirements on POSIX implementations derive from alignment with the  
 22228 Single UNIX Specification:
- 22229 • In the DESCRIPTION, the second paragraph is reworded to indicate that the saved set-user-  
 22230 ID of the calling process is checked in place of its effective user ID. This is a FIPS  
 22231 requirement.
  - 22232 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
 22233 required for conforming implementations of previous POSIX specifications, it was not  
 22234 required for UNIX applications.
  - 22235 • The behavior when *pid* is  $-1$  is now specified. It was previously explicitly unspecified in the  
 22236 POSIX.1-1988 standard.
- 22237 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

22238 **NAME**

22239 killpg — send a signal to a process group

22240 **SYNOPSIS**

22241 XSI #include <signal.h>

22242 int killpg(pid\_t pgrp, int sig);

22243

22244 **DESCRIPTION**

22245 The *killpg()* function shall send the signal specified by *sig* to the process group specified by *pgrp*.

22246 If *pgrp* is greater than 1, *killpg(pgrp, sig)* shall be equivalent to *kill(-pgrp, sig)*. If *pgrp* is less than or  
22247 equal to 1, the behavior of *killpg()* is undefined.

22248 **RETURN VALUE**

22249 Refer to *kill()*.

22250 **ERRORS**

22251 Refer to *kill()*.

22252 **EXAMPLES**

22253 None.

22254 **APPLICATION USAGE**

22255 None.

22256 **RATIONALE**

22257 None.

22258 **FUTURE DIRECTIONS**

22259 None.

22260 **SEE ALSO**

22261 *getpgid()*, *getpid()*, *kill()*, *raise()*, the Base Definitions volume of IEEE Std 1003.1-2001, <signal.h>

22262 **CHANGE HISTORY**

22263 First released in Issue 4, Version 2.

22264 **Issue 5**

22265 Moved from X/OPEN UNIX extension to BASE.

22266 **NAME**22267       **l64a** — convert a 32-bit integer to a radix-64 ASCII string22268 **SYNOPSIS**

22269 XSI       #include &lt;stdlib.h&gt;

22270       char \*l64a(long value);

22271

22272 **DESCRIPTION**22273       Refer to *a64l()*.

22274 **NAME**

22275 labs, llabs — return a long integer absolute value

22276 **SYNOPSIS**

22277 #include <stdlib.h>

22278 long labs(long i);

22279 long long llabs(long long i);

22280 **DESCRIPTION**

22281 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
22282 conflict between the requirements described here and the ISO C standard is unintentional. This  
22283 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22284 The *labs()* function shall compute the absolute value of the **long** integer operand *i*. The *llabs()*  
22285 function shall compute the absolute value of the **long long** integer operand *i*. If the result cannot  
22286 be represented, the behavior is undefined.

22287 **RETURN VALUE**

22288 The *labs()* function shall return the absolute value of the **long** integer operand. The *llabs()*  
22289 function shall return the absolute value of the **long long** integer operand.

22290 **ERRORS**

22291 No errors are defined.

22292 **EXAMPLES**

22293 None.

22294 **APPLICATION USAGE**

22295 None.

22296 **RATIONALE**

22297 None.

22298 **FUTURE DIRECTIONS**

22299 None.

22300 **SEE ALSO**

22301 *abs()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>

22302 **CHANGE HISTORY**

22303 First released in Issue 4. Derived from the ISO C standard.

22304 **Issue 6**

22305 The *llabs()* function is added for alignment with the ISO/IEC 9899:1999 standard.

## 22306 NAME

22307 lchown — change the owner and group of a symbolic link

## 22308 SYNOPSIS

22309 XSI #include &lt;unistd.h&gt;

22310 int lchown(const char \*path, uid\_t owner, gid\_t group);

22311

## 22312 DESCRIPTION

22313 The *lchown()* function shall be equivalent to *chown()*, except in the case where the named file is a  
 22314 symbolic link. In this case, *lchown()* shall change the ownership of the symbolic link file itself,  
 22315 while *chown()* changes the ownership of the file or directory to which the symbolic link refers.

## 22316 RETURN VALUE

22317 Upon successful completion, *lchown()* shall return 0. Otherwise, it shall return -1 and set *errno* to  
 22318 indicate an error.

## 22319 ERRORS

22320 The *lchown()* function shall fail if:

- 22321 [EACCES] Search permission is denied on a component of the path prefix of *path*.
- 22322 [EINVAL] The owner or group ID is not a value supported by the implementation.
- 22323 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 22324 argument.
- 22325 [ENAMETOOLONG]  
 22326 The length of a pathname exceeds {PATH\_MAX} or a pathname component is  
 22327 longer than {NAME\_MAX}.
- 22328 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- 22329 [ENOTDIR] A component of the path prefix of *path* is not a directory.
- 22330 [EOPNOTSUPP] The *path* argument names a symbolic link and the implementation does not  
 22331 support setting the owner or group of a symbolic link.
- 22332 [EPERM] The effective user ID does not match the owner of the file and the process  
 22333 does not have appropriate privileges.
- 22334 [EROFS] The file resides on a read-only file system.

22335 The *lchown()* function may fail if:

- 22336 [EIO] An I/O error occurred while reading or writing to the file system.
- 22337 [EINTR] A signal was caught during execution of the function.
- 22338 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 22339 resolution of the *path* argument.
- 22340 [ENAMETOOLONG]  
 22341 Pathname resolution of a symbolic link produced an intermediate result  
 22342 whose length exceeds {PATH\_MAX}.

22343 **EXAMPLES**22344 **Changing the Current Owner of a File**

22345 The following example shows how to change the ownership of the symbolic link named  
22346 **/modules/pass1** to the user ID associated with “jones” and the group ID associated with “cnd”.

22347 The numeric value for the user ID is obtained by using the *getpwnam()* function. The numeric  
22348 value for the group ID is obtained by using the *getgrnam()* function.

```
22349 #include <sys/types.h>
22350 #include <unistd.h>
22351 #include <pwd.h>
22352 #include <grp.h>

22353 struct passwd *pwd;
22354 struct group *grp;
22355 char *path = "/modules/pass1";
22356 ...
22357 pwd = getpwnam("jones");
22358 grp = getgrnam("cnd");
22359 lchown(path, pwd->pw_uid, grp->gr_gid);
```

22360 **APPLICATION USAGE**

22361 On implementations which support symbolic links as directory entries rather than files, *lchown()*  
22362 may fail.

22363 **RATIONALE**

22364 None.

22365 **FUTURE DIRECTIONS**

22366 None.

22367 **SEE ALSO**

22368 *chown()*, *symlink()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<unistd.h>**

22369 **CHANGE HISTORY**

22370 First released in Issue 4, Version 2.

22371 **Issue 5**

22372 Moved from X/OPEN UNIX extension to BASE.

22373 **Issue 6**

22374 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
22375 [ELOOP] error condition is added.

22376 The Open Group Base Resolution bwg2001-013 is applied, adding wording to the  
22377 APPLICATION USAGE.

22378 **NAME**

22379        lcong48 — seed a uniformly distributed pseudo-random signed long integer generator

22380 **SYNOPSIS**

22381 XSI        #include <stdlib.h>

22382        void lcong48(unsigned short param[7]);

22383

22384 **DESCRIPTION**

22385        Refer to *drand48()*.

22386 **NAME**

22387 ldexp, ldexpf, ldexpl — load exponent of a floating-point number

22388 **SYNOPSIS**

22389 #include &lt;math.h&gt;

22390 double ldexp(double x, int exp);

22391 float ldexpf(float x, int exp);

22392 long double ldexpl(long double x, int exp);

22393 **DESCRIPTION**

22394 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 22395 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22396 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22397 These functions shall compute the quantity  $x * 2^{exp}$ .

22398 An application wishing to check for error situations should set *errno* to zero and call  
 22399 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 22400 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 22401 zero, an error has occurred.

22402 **RETURN VALUE**22403 Upon successful completion, these functions shall return *x* multiplied by 2, raised to the power  
22404 *exp*.

22405 If these functions would cause overflow, a range error shall occur and *ldexp*(*x*), *ldexpf*(*x*), and  
 22406 *ldexpl*(*x*) shall return ±HUGE\_VAL, ±HUGE\_VALF, and ±HUGE\_VALL (according to the sign of  
 22407 *x*), respectively.

22408 If the correct value would cause underflow, and is not representable, a range error may occur,  
 22409 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.

22410 **MX** If *x* is NaN, a NaN shall be returned.22411 If *x* is ±0 or ±Inf, *x* shall be returned.22412 If *exp* is 0, *x* shall be returned.

22413 If the correct value would cause underflow, and is representable, a range error may occur and  
 22414 the correct value shall be returned.

22415 **ERRORS**

22416 These functions shall fail if:

22417 **Range Error** The result overflows.

22418 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 22419 then *errno* shall be set to [ERANGE]. If the integer expression  
 22420 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow  
 22421 floating-point exception shall be raised.

22422 These functions may fail if:

22423 **Range Error** The result underflows.

22424 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 22425 then *errno* shall be set to [ERANGE]. If the integer expression  
 22426 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow  
 22427 floating-point exception shall be raised.

22428 **EXAMPLES**

22429 None.

22430 **APPLICATION USAGE**

22431 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
22432 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

22433 **RATIONALE**

22434 None.

22435 **FUTURE DIRECTIONS**

22436 None.

22437 **SEE ALSO**

22438 *feclearexcept()*, *fetestexcept()*, *frexp()*, *isnan()*, the Base Definitions volume of  
22439 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,  
22440 <math.h>

22441 **CHANGE HISTORY**

22442 First released in Issue 1. Derived from Issue 1 of the SVID.

22443 **Issue 5**

22444 The DESCRIPTION is updated to indicate how an application should check for an error. This  
22445 text was previously published in the APPLICATION USAGE section.

22446 **Issue 6**

22447 The *ldexpf()* and *ldexpl()* functions are added for alignment with the ISO/IEC 9899:1999  
22448 standard.

22449 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
22450 revised to align with the ISO/IEC 9899:1999 standard.

22451 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
22452 marked.

22453 **NAME**

22454 ldiv, lldiv — compute quotient and remainder of a long division

22455 **SYNOPSIS**

22456 #include &lt;stdlib.h&gt;

22457 ldiv\_t ldiv(long *numer*, long *denom*);22458 lldiv\_t lldiv(long long *numer*, long long *denom*);22459 **DESCRIPTION**

22460 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 22461 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22462 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22463 These functions shall compute the quotient and remainder of the division of the numerator  
 22464 *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the **long**  
 22465 integer (for the *ldiv()* function) or **long long** integer (for the *lldiv()* function) of lesser magnitude  
 22466 that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is  
 22467 undefined; otherwise, *quot* \* *denom* + *rem* shall equal *numer*.

22468 **RETURN VALUE**

22469 The *ldiv()* function shall return a structure of type **ldiv\_t**, comprising both the quotient and the  
 22470 remainder. The structure shall include the following members, in any order:

22471 long quot; /\* Quotient \*/

22472 long rem; /\* Remainder \*/

22473 The *lldiv()* function shall return a structure of type **lldiv\_t**, comprising both the quotient and the  
 22474 remainder. The structure shall include the following members, in any order:

22475 long long quot; /\* Quotient \*/

22476 long long rem; /\* Remainder \*/

22477 **ERRORS**

22478 No errors are defined.

22479 **EXAMPLES**

22480 None.

22481 **APPLICATION USAGE**

22482 None.

22483 **RATIONALE**

22484 None.

22485 **FUTURE DIRECTIONS**

22486 None.

22487 **SEE ALSO**22488 *div()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>22489 **CHANGE HISTORY**

22490 First released in Issue 4. Derived from the ISO C standard.

22491 **Issue 6**22492 The *lldiv()* function is added for alignment with the ISO/IEC 9899:1999 standard.

22493 **NAME**

22494 **lfind** — find entry in a linear search table

22495 **SYNOPSIS**

```
22496 XSI #include <search.h>
```

```
22497 void *lfind(const void *key, const void *base, size_t *nelp,
22498 size_t width, int (*compar)(const void *, const void *));
22499
```

22500 **DESCRIPTION**

22501 Refer to *lsearch()*.

22502 **NAME**

22503 lgamma, lgammaf, lgammal — log gamma function

22504 **SYNOPSIS**

```
22505 #include <math.h>
22506 double lgamma(double x);
22507 float lgammaf(float x);
22508 long double lgammal(long double x);
22509 XSI extern int signgam;
22510
```

22511 **DESCRIPTION**

22512 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 22513 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22514 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22515 These functions shall compute  $\log_e|\Gamma(x)|$  where  $\Gamma(x)$  is defined as  $\int_0^{\infty} e^{-t}t^{x-1} dt$ . The argument  $x$   
 22516 need not be a non-positive integer ( $\Gamma(x)$  is defined over the reals, except the non-positive  
 22517 integers).  
 22518

22519 XSI The sign of  $\Gamma(x)$  is returned in the external integer *signgam*.

22520 CX These functions need not be reentrant. A function that is not required to be reentrant is not  
 22521 required to be thread-safe.

22522 An application wishing to check for error situations should set *errno* to zero and call  
 22523 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 22524 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 22525 zero, an error has occurred.

22526 **RETURN VALUE**

22527 Upon successful completion, these functions shall return the logarithmic gamma of  $x$ .

22528 If  $x$  is a non-positive integer, a pole error shall occur and *lgamma*(), *lgammaf*(), and *lgammal*()  
 22529 shall return +HUGE\_VAL, +HUGE\_VALF, and +HUGE\_VALL, respectively.

22530 If the correct value would cause overflow, a range error shall occur and *lgamma*(), *lgammaf*(),  
 22531 and *lgammal*() shall return ±HUGE\_VAL, ±HUGE\_VALF, and ±HUGE\_VALL (having the same  
 22532 sign as the correct value), respectively.

22533 MX If  $x$  is NaN, a NaN shall be returned.

22534 If  $x$  is 1 or 2, +0 shall be returned.

22535 If  $x$  is ±Inf, +Inf shall be returned.

22536 **ERRORS**

22537 These functions shall fail if:

22538 Pole Error The  $x$  argument is a negative integer or zero.

22539 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 22540 then *errno* shall be set to [ERANGE]. If the integer expression  
 22541 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the divide-by-  
 22542 zero floating-point exception shall be raised.

22543 Range Error The result overflows.

22544 If the integer expression (`math_errhandling & MATH_ERRNO`) is non-zero,  
22545 then *errno* shall be set to [ERANGE]. If the integer expression  
22546 (`math_errhandling & MATH_ERREXCEPT`) is non-zero, then the overflow  
22547 floating-point exception shall be raised.

**22548 EXAMPLES**

22549 None.

**22550 APPLICATION USAGE**

22551 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`  
22552 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

**22553 RATIONALE**

22554 None.

**22555 FUTURE DIRECTIONS**

22556 None.

**22557 SEE ALSO**

22558 *exp()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
22559 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <**math.h**>

**22560 CHANGE HISTORY**

22561 First released in Issue 3.

**22562 Issue 5**

22563 The DESCRIPTION is updated to indicate how an application should check for an error. This  
22564 text was previously published in the APPLICATION USAGE section.

22565 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

**22566 Issue 6**

22567 The *lgamma()* function is no longer marked as an extension.

22568 The *lgammaf()* and *lgammal()* functions are added for alignment with the ISO/IEC 9899:1999  
22569 standard.

22570 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
22571 revised to align with the ISO/IEC 9899:1999 standard.

22572 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
22573 marked.

22574 XSI extensions are marked.

22575 **NAME**

22576 link — link to a file

22577 **SYNOPSIS**

22578 #include &lt;unistd.h&gt;

22579 int link(const char \*path1, const char \*path2);

22580 **DESCRIPTION**22581 The *link()* function shall create a new link (directory entry) for the existing file, *path1*.

22582 The *path1* argument points to a pathname naming an existing file. The *path2* argument points to  
 22583 a pathname naming the new directory entry to be created. The *link()* function shall atomically  
 22584 create a new link for the existing file and the link count of the file shall be incremented by one.

22585 If *path1* names a directory, *link()* shall fail unless the process has appropriate privileges and the  
 22586 implementation supports using *link()* on directories.

22587 Upon successful completion, *link()* shall mark for update the *st\_ctime* field of the file. Also, the  
 22588 *st\_ctime* and *st\_mtime* fields of the directory that contains the new entry shall be marked for  
 22589 update.

22590 If *link()* fails, no link shall be created and the link count of the file shall remain unchanged.

22591 The implementation may require that the calling process has permission to access the existing  
 22592 file.

22593 **RETURN VALUE**

22594 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 22595 indicate the error.

22596 **ERRORS**22597 The *link()* function shall fail if:

22598 [EACCES] A component of either path prefix denies search permission, or the requested  
 22599 link requires writing in a directory that denies write permission, or the calling  
 22600 process does not have permission to access the existing file and this is  
 22601 required by the implementation.

22602 [EEXIST] The *path2* argument resolves to an existing file or refers to a symbolic link.

22603 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path1* or  
 22604 *path2* argument.

22605 [EMLINK] The number of links to the file named by *path1* would exceed {LINK\_MAX}.

22606 [ENAMETOOLONG]

22607 The length of the *path1* or *path2* argument exceeds {PATH\_MAX} or a  
 22608 pathname component is longer than {NAME\_MAX}.

22609 [ENOENT] A component of either path prefix does not exist; the file named by *path1* does  
 22610 not exist; or *path1* or *path2* points to an empty string.

22611 [ENOSPC] The directory to contain the link cannot be extended.

22612 [ENOTDIR] A component of either path prefix is not a directory.

22613 [EPERM] The file named by *path1* is a directory and either the calling process does not  
 22614 have appropriate privileges or the implementation prohibits using *link()* on  
 22615 directories.

- 22616 [EROFS] The requested link requires writing in a directory on a read-only file system.
- 22617 [EXDEV] The link named by *path2* and the file named by *path1* are on different file  
22618 systems and the implementation does not support links between file systems.
- 22619 XSR [EXDEV] *path1* refers to a named STREAM.
- 22620 The *link()* function may fail if:
- 22621 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
22622 resolution of the *path1* or *path2* argument.
- 22623 [ENAMETOOLONG]  
22624 As a result of encountering a symbolic link in resolution of the *path1* or *path2*  
22625 argument, the length of the substituted pathname string exceeded  
22626 {PATH\_MAX}.

22627 **EXAMPLES**22628 **Creating a Link to a File**

22629 The following example shows how to create a link to a file named `/home/cnd/mod1` by creating a  
22630 new directory entry named `/modules/pass1`.

```
22631 #include <unistd.h>
22632 char *path1 = "/home/cnd/mod1";
22633 char *path2 = "/modules/pass1";
22634 int status;
22635 ...
22636 status = link (path1, path2);
```

22637 **Creating a Link to a File Within a Program**

22638 In the following program example, the *link()* function links the `/etc/passwd` file (defined as  
22639 **PASSWDFILE**) to a file named `/etc/opasswd` (defined as **SAVEFILE**), which is used to save the  
22640 current password file. Then, after removing the current password file (defined as  
22641 **PASSWDFILE**), the new password file is saved as the current password file using the *link()*  
22642 function again.

```
22643 #include <unistd.h>
22644 #define LOCKFILE "/etc/ptmp"
22645 #define PASSWDFILE "/etc/passwd"
22646 #define SAVEFILE "/etc/opasswd"
22647 ...
22648 /* Save current password file */
22649 link (PASSWDFILE, SAVEFILE);
22650 /* Remove current password file. */
22651 unlink (PASSWDFILE);
22652 /* Save new password file as current password file. */
22653 link (LOCKFILE, PASSWDFILE);
```

**22654 APPLICATION USAGE**

22655           Some implementations do allow links between file systems.

**22656 RATIONALE**

22657           Linking to a directory is restricted to the superuser in most historical implementations because  
22658           this capability may produce loops in the file hierarchy or otherwise corrupt the file system. This  
22659           volume of IEEE Std 1003.1-2001 continues that philosophy by prohibiting *link()* and *unlink()*  
22660           from doing this. Other functions could do it if the implementor designed such an extension.

22661           Some historical implementations allow linking of files on different file systems. Wording was  
22662           added to explicitly allow this optional behavior.

22663           The exception for cross-file system links is intended to apply only to links that are  
22664           programmatically indistinguishable from “hard” links.

**22665 FUTURE DIRECTIONS**

22666           None.

**22667 SEE ALSO**

22668           *symlink()*, *unlink()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

**22669 CHANGE HISTORY**

22670           First released in Issue 1. Derived from Issue 1 of the SVID.

**22671 Issue 6**

22672           The following new requirements on POSIX implementations derive from alignment with the  
22673           Single UNIX Specification:

- 22674           • The [ELOOP] mandatory error condition is added.
- 22675           • A second [ENAMETOOLONG] is added as an optional error condition.

22676           The following changes were made to align with the IEEE P1003.1a draft standard:

- 22677           • An explanation is added of the action when *path2* refers to a symbolic link.
- 22678           • The [ELOOP] optional error condition is added.

## 22679 NAME

22680 lio\_listio — list directed I/O (**REALTIME**)

## 22681 SYNOPSIS

22682 AIO #include &lt;aio.h&gt;

```
22683 int lio_listio(int mode, struct aiocb *restrict const list[restrict],
22684 int nent, struct sigevent *restrict sig);
22685
```

## 22686 DESCRIPTION

22687 The *lio\_listio()* function shall initiate a list of I/O requests with a single function call.

22688 The *mode* argument takes one of the values LIO\_WAIT or LIO\_NOWAIT declared in <aio.h> and  
 22689 determines whether the function returns when the I/O operations have been completed, or as  
 22690 soon as the operations have been queued. If the *mode* argument is LIO\_WAIT, the function shall  
 22691 wait until all I/O is complete and the *sig* argument shall be ignored.

22692 If the *mode* argument is LIO\_NOWAIT, the function shall return immediately, and asynchronous  
 22693 notification shall occur, according to the *sig* argument, when all the I/O operations complete. If  
 22694 *sig* is NULL, then no asynchronous notification shall occur. If *sig* is not NULL, asynchronous  
 22695 notification occurs as specified in Section 2.4.1 (on page 28) when all the requests in *list* have  
 22696 completed.

22697 The I/O requests enumerated by *list* are submitted in an unspecified order.

22698 The *list* argument is an array of pointers to **aiocb** structures. The array contains *nent* elements.  
 22699 The array may contain NULL elements, which shall be ignored.

22700 The *aio\_lio\_opcode* field of each **aiocb** structure specifies the operation to be performed. The  
 22701 supported operations are LIO\_READ, LIO\_WRITE, and LIO\_NOP; these symbols are defined in  
 22702 <aio.h>. The LIO\_NOP operation causes the list entry to be ignored. If the *aio\_lio\_opcode*  
 22703 element is equal to LIO\_READ, then an I/O operation is submitted as if by a call to *aio\_read()*  
 22704 with the *aiocbp* equal to the address of the **aiocb** structure. If the *aio\_lio\_opcode* element is equal  
 22705 to LIO\_WRITE, then an I/O operation is submitted as if by a call to *aio\_write()* with the *aiocbp*  
 22706 equal to the address of the **aiocb** structure.

22707 The *aio\_fildes* member specifies the file descriptor on which the operation is to be performed.22708 The *aio\_buf* member specifies the address of the buffer to or from which the data is transferred.22709 The *aio\_nbytes* member specifies the number of bytes of data to be transferred.

22710 The members of the **aiocb** structure further describe the I/O operation to be performed, in a  
 22711 manner identical to that of the corresponding **aiocb** structure when used by the *aio\_read()* and  
 22712 *aio\_write()* functions.

22713 The *nent* argument specifies how many elements are members of the list; that is, the length of the  
 22714 array.

22715 The behavior of this function is altered according to the definitions of synchronized I/O data  
 22716 integrity completion and synchronized I/O file integrity completion if synchronized I/O is  
 22717 enabled on the file associated with *aio\_fildes*.

22718 For regular files, no data transfer shall occur past the offset maximum established in the open  
 22719 file description associated with *aiocbp->aio\_fildes*.

## 22720 RETURN VALUE

22721 If the *mode* argument has the value LIO\_NOWAIT, the *lio\_listio()* function shall return the value  
 22722 zero if the I/O operations are successfully queued; otherwise, the function shall return the value  
 22723 -1 and set *errno* to indicate the error.

22724 If the *mode* argument has the value LIO\_WAIT, the *lio\_listio()* function shall return the value  
 22725 zero when all the indicated I/O has completed successfully. Otherwise, *lio\_listio()* shall return a  
 22726 value of -1 and set *errno* to indicate the error.

22727 In either case, the return value only indicates the success or failure of the *lio\_listio()* call itself,  
 22728 not the status of the individual I/O requests. In some cases one or more of the I/O requests  
 22729 contained in the list may fail. Failure of an individual request does not prevent completion of  
 22730 any other individual request. To determine the outcome of each I/O request, the application  
 22731 shall examine the error status associated with each **aiocb** control block. The error statuses so  
 22732 returned are identical to those returned as the result of an *aio\_read()* or *aio\_write()* function.

## 22733 ERRORS

22734 The *lio\_listio()* function shall fail if:

22735 [EAGAIN] The resources necessary to queue all the I/O requests were not available. The  
 22736 application may check the error status for each **aiocb** to determine the  
 22737 individual request(s) that failed.

22738 [EAGAIN] The number of entries indicated by *nent* would cause the system-wide limit  
 22739 {AIO\_MAX} to be exceeded.

22740 [EINVAL] The *mode* argument is not a proper value, or the value of *nent* was greater than  
 22741 {AIO\_LISTIO\_MAX}.

22742 [EINTR] A signal was delivered while waiting for all I/O requests to complete during  
 22743 an LIO\_WAIT operation. Note that, since each I/O operation invoked by  
 22744 *lio\_listio()* may possibly provoke a signal when it completes, this error return  
 22745 may be caused by the completion of one (or more) of the very I/O operations  
 22746 being awaited. Outstanding I/O requests are not canceled, and the application  
 22747 shall examine each list element to determine whether the request was  
 22748 initiated, canceled, or completed.

22749 [EIO] One or more of the individual I/O operations failed. The application may  
 22750 check the error status for each **aiocb** structure to determine the individual  
 22751 request(s) that failed.

22752 In addition to the errors returned by the *lio\_listio()* function, if the *lio\_listio()* function succeeds  
 22753 or fails with errors of [EAGAIN], [EINTR], or [EIO], then some of the I/O specified by the list  
 22754 may have been initiated. If the *lio\_listio()* function fails with an error code other than [EAGAIN],  
 22755 [EINTR], or [EIO], no operations from the list shall have been initiated. The I/O operation  
 22756 indicated by each list element can encounter errors specific to the individual read or write  
 22757 function being performed. In this event, the error status for each **aiocb** control block contains the  
 22758 associated error code. The error codes that can be set are the same as would be set by a *read()* or  
 22759 *write()* function, with the following additional error codes possible:

22760 [EAGAIN] The requested I/O operation was not queued due to resource limitations.

22761 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit  
 22762 *aio\_cancel()* request.

22763 [EFBIG] The *aiocbp->aio\_lio\_opcode* is LIO\_WRITE, the file is a regular file,  
 22764 *aiocbp->aio\_nbytes* is greater than 0, and the *aiocbp->aio\_offset* is greater than or  
 22765 equal to the offset maximum in the open file description associated with

22766 *aiocbp->aio\_fildes*.

22767 [EINPROGRESS] The requested I/O is in progress.

22768 [EOVERFLOW] The *aiocbp->aio\_lio\_opcode* is LIO\_READ, the file is a regular file,  
 22769 *aiocbp->aio\_nbytes* is greater than 0, and the *aiocbp->aio\_offset* is before the  
 22770 end-of-file and is greater than or equal to the offset maximum in the open file  
 22771 description associated with *aiocbp->aio\_fildes*.

22772 **EXAMPLES**

22773 None.

22774 **APPLICATION USAGE**

22775 None.

22776 **RATIONALE**

22777 Although it may appear that there are inconsistencies in the specified circumstances for error  
 22778 codes, the [EIO] error condition applies when any circumstance relating to an individual  
 22779 operation makes that operation fail. This might be due to a badly formulated request (for  
 22780 example, the *aio\_lio\_opcode* field is invalid, and *aio\_error()* returns [EINVAL]) or might arise from  
 22781 application behavior (for example, the file descriptor is closed before the operation is initiated,  
 22782 and *aio\_error()* returns [EBADF]).

22783 The limitation on the set of error codes returned when operations from the list shall have been  
 22784 initiated enables applications to know when operations have been started and whether  
 22785 *aio\_error()* is valid for a specific operation.

22786 **FUTURE DIRECTIONS**

22787 None.

22788 **SEE ALSO**

22789 *aio\_read()*, *aio\_write()*, *aio\_error()*, *aio\_return()*, *aio\_cancel()*, *close()*, *exec*, *exit()*, *fork()*, *lseek()*,  
 22790 *read()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**aio.h**>

22791 **CHANGE HISTORY**

22792 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

22793 Large File Summit extensions are added.

22794 **Issue 6**

22795 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 22796 implementation does not support the Asynchronous Input and Output option.

22797 The *lio\_listio()* function is marked as part of the Asynchronous Input and Output option.

22798 The following new requirements on POSIX implementations derive from alignment with the  
 22799 Single UNIX Specification:

- 22800 • In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs  
 22801 past the offset maximum established in the open file description associated with  
 22802 *aiocbp->aio\_fildes*. This change is to support large files.

- 22803 • The [EBIG] and [EOVERFLOW] error conditions are defined. This change is to support large  
 22804 files.

22805 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

22806 The **restrict** keyword is added to the *lio\_listio()* prototype for alignment with the  
 22807 ISO/IEC 9899:1999 standard.

22808 **NAME**

22809       listen — listen for socket connections and limit the queue of incoming connections

22810 **SYNOPSIS**

22811       #include <sys/socket.h>  
22812       int listen(int *socket*, int *backlog*);

22813 **DESCRIPTION**

22814       The *listen()* function shall mark a connection-mode socket, specified by the *socket* argument, as  
22815       accepting connections.

22816       The *backlog* argument provides a hint to the implementation which the implementation shall use  
22817       to limit the number of outstanding connections in the socket's listen queue. Implementations  
22818       may impose a limit on *backlog* and silently reduce the specified value. Normally, a larger *backlog*  
22819       argument value shall result in a larger or equal length of the listen queue. Implementations shall  
22820       support values of *backlog* up to SOMAXCONN, defined in <sys/socket.h>.

22821       The implementation may include incomplete connections in its listen queue. The limits on the  
22822       number of incomplete connections and completed connections queued may be different.

22823       The implementation may have an upper limit on the length of the listen queue—either global or  
22824       per accepting socket. If *backlog* exceeds this limit, the length of the listen queue is set to the limit.

22825       If *listen()* is called with a *backlog* argument value that is less than 0, the function behaves as if it  
22826       had been called with a *backlog* argument value of 0.

22827       A *backlog* argument of 0 may allow the socket to accept connections, in which case the length of  
22828       the listen queue may be set to an implementation-defined minimum value.

22829       The socket in use may require the process to have appropriate privileges to use the *listen()*  
22830       function.

22831 **RETURN VALUE**

22832       Upon successful completions, *listen()* shall return 0; otherwise, -1 shall be returned and *errno* set  
22833       to indicate the error.

22834 **ERRORS**

22835       The *listen()* function shall fail if:

22836       [EBADF]       The *socket* argument is not a valid file descriptor.

22837       [EDESTADDRREQ]

22838       The socket is not bound to a local address, and the protocol does not support  
22839       listening on an unbound socket.

22840       [EINVAL]       The *socket* is already connected.

22841       [ENOTSOCK]    The *socket* argument does not refer to a socket.

22842       [EOPNOTSUPP] The socket protocol does not support *listen()*.

22843       The *listen()* function may fail if:

22844       [EACCES]       The calling process does not have the appropriate privileges.

22845       [EINVAL]       The *socket* has been shut down.

22846       [ENOBUFS]      Insufficient resources are available in the system to complete the call.

22847 **EXAMPLES**

22848           None.

22849 **APPLICATION USAGE**

22850           None.

22851 **RATIONALE**

22852           None.

22853 **FUTURE DIRECTIONS**

22854           None.

22855 **SEE ALSO**

22856           *accept()*, *connect()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**sys/socket.h**>

22857 **CHANGE HISTORY**

22858           First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

22859           The DESCRIPTION is updated to describe the relationship of SOMAXCONN and the *backlog*  
22860           argument.

22861 **NAME**

22862 llabs — return a long integer absolute value

22863 **SYNOPSIS**

22864 #include &lt;stdlib.h&gt;

22865 long long llabs(long long *i*);22866 **DESCRIPTION**22867 Refer to *labs()*.

22868 **NAME**

22869        `lldiv` — compute quotient and remainder of a long division

22870 **SYNOPSIS**

22871        `#include <stdlib.h>`

22872        `lldiv_t lldiv(long long numer, long long denom);`

22873 **DESCRIPTION**

22874        Refer to *ldiv()*.

22875 **NAME**

22876 llrint, llrintf, llrintl — round to the nearest integer value using current rounding direction

22877 **SYNOPSIS**

22878 #include <math.h>

22879 long long llrint(double x);

22880 long long llrintf(float x);

22881 long long llrintl(long double x);

22882 **DESCRIPTION**

22883 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 22884 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22885 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22886 These functions shall round their argument to the nearest integer value, rounding according to  
 22887 the current rounding direction.

22888 An application wishing to check for error situations should set *errno* to zero and call  
 22889 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 22890 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 22891 zero, an error has occurred.

22892 **RETURN VALUE**

22893 Upon successful completion, these functions shall return the rounded integer value.

22894 **MX** If *x* is NaN, a domain error shall occur, and an unspecified value is returned.

22895 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

22896 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

22897 If the correct value is positive and too large to represent as a **long long**, a domain error shall  
 22898 occur and an unspecified value is returned.

22899 If the correct value is negative and too large to represent as a **long long**, a domain error shall  
 22900 occur and an unspecified value is returned.

22901 **ERRORS**

22902 These functions shall fail if:

22903 **MX** Domain Error The *x* argument is NaN or ±Inf, or the correct value is not representable as an  
 22904 integer.

22905 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 22906 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling  
 22907 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 22908 shall be raised.

22909 **EXAMPLES**

22910 None.

22911 **APPLICATION USAGE**

22912 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 22913 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

22914 **RATIONALE**

22915 These functions provide floating-to-integer conversions. They round according to the current  
 22916 rounding direction. If the rounded value is outside the range of the return type, the numeric  
 22917 result is unspecified and the invalid floating-point exception is raised. When they raise no other  
 22918 floating-point exception and the result differs from the argument, they raise the inexact

22919 floating-point exception.

22920 **FUTURE DIRECTIONS**

22921 None.

22922 **SEE ALSO**

22923 *feclearexcept()*, *fetestexcept()*, *lrint()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section  
22924 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

22925 **CHANGE HISTORY**

22926 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

22927 **NAME**

22928 llround, llroundf, llroundl — round to nearest integer value

22929 **SYNOPSIS**

22930 #include &lt;math.h&gt;

22931 long long llround(double x);

22932 long long llroundf(float x);

22933 long long llroundl(long double x);

22934 **DESCRIPTION**

22935 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 22936 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22937 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22938 These functions shall round their argument to the nearest integer value, rounding halfway cases  
 22939 away from zero, regardless of the current rounding direction.

22940 An application wishing to check for error situations should set *errno* to zero and call  
 22941 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 22942 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 22943 zero, an error has occurred.

22944 **RETURN VALUE**

22945 Upon successful completion, these functions shall return the rounded integer value.

22946 **MX** If *x* is NaN, a domain error shall occur, and an unspecified value is returned.22947 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.22948 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

22949 If the correct value is positive and too large to represent as a **long long**, a domain error shall  
 22950 occur and an unspecified value is returned.

22951 If the correct value is negative and too large to represent as a **long long**, a domain error shall  
 22952 occur and an unspecified value is returned.

22953 **ERRORS**

22954 These functions shall fail if:

22955 **MX** Domain Error The *x* argument is NaN or ±Inf, or the correct value is not representable as an  
 22956 integer.

22957 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 22958 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling  
 22959 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 22960 shall be raised.

22961 **EXAMPLES**

22962 None.

22963 **APPLICATION USAGE**

22964 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 22965 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

22966 **RATIONALE**

22967 These functions differ from the *llrint*() functions in that the default rounding direction for the  
 22968 *llround*() functions round halfway cases away from zero and need not raise the inexact floating-  
 22969 point exception for non-integer arguments that round to within the range of the return type.

22970 **FUTURE DIRECTIONS**

22971           None.

22972 **SEE ALSO**

22973           *feclearexcept()*, *fetestexcept()*, *lround()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
22974           Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

22975 **CHANGE HISTORY**

22976           First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

22977 **NAME**

22978 localeconv — return locale-specific information

22979 **SYNOPSIS**

22980 #include &lt;locale.h&gt;

22981 struct lconv \*localeconv(void);

22982 **DESCRIPTION**

22983 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 22984 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22985 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

22986 The *localeconv()* function shall set the components of an object with the type **struct lconv** with  
 22987 the values appropriate for the formatting of numeric quantities (monetary and otherwise)  
 22988 according to the rules of the current locale.

22989 The members of the structure with type **char \*** are pointers to strings, any of which (except  
 22990 **decimal\_point**) can point to " ", to indicate that the value is not available in the current locale or  
 22991 is of zero length. The members with type **char** are non-negative numbers, any of which can be  
 22992 {CHAR\_MAX} to indicate that the value is not available in the current locale.

22993 The members include the following:

22994 **char \*decimal\_point**

22995 The radix character used to format non-monetary quantities.

22996 **char \*thousands\_sep**

22997 The character used to separate groups of digits before the decimal-point character in  
 22998 formatted non-monetary quantities.

22999 **char \*grouping**

23000 A string whose elements taken as one-byte integer values indicate the size of each group of  
 23001 digits in formatted non-monetary quantities.

23002 **char \*int\_curr\_symbol**

23003 The international currency symbol applicable to the current locale. The first three  
 23004 characters contain the alphabetic international currency symbol in accordance with those |  
 23005 specified in the ISO 4217:2001 standard. The fourth character (immediately preceding the |  
 23006 null byte) is the character used to separate the international currency symbol from the  
 23007 monetary quantity.

23008 **char \*currency\_symbol**

23009 The local currency symbol applicable to the current locale.

23010 **char \*mon\_decimal\_point**

23011 The radix character used to format monetary quantities.

23012 **char \*mon\_thousands\_sep**

23013 The separator for groups of digits before the decimal-point in formatted monetary  
 23014 quantities.

23015 **char \*mon\_grouping**

23016 A string whose elements taken as one-byte integer values indicate the size of each group of  
 23017 digits in formatted monetary quantities.

23018 **char \*positive\_sign**

23019 The string used to indicate a non-negative valued formatted monetary quantity.

23020        **char \*negative\_sign**  
23021            The string used to indicate a negative valued formatted monetary quantity.

23022        **char int\_frac\_digits**  
23023            The number of fractional digits (those after the decimal-point) to be displayed in an  
23024            internationally formatted monetary quantity.

23025        **char frac\_digits**  
23026            The number of fractional digits (those after the decimal-point) to be displayed in a  
23027            formatted monetary quantity.

23028        **char p\_cs\_precedes**  
23029            Set to 1 if the **currency\_symbol** precedes the value for a non-negative formatted monetary  
23030            quantity. Set to 0 if the symbol succeeds the value.

23031        **char p\_sep\_by\_space**  
23032            Set to a value indicating the separation of the **currency\_symbol**, the sign string, and the  
23033            value for a non-negative formatted monetary quantity.

23034        **char n\_cs\_precedes**  
23035            Set to 1 if the **currency\_symbol** precedes the value for a negative formatted monetary  
23036            quantity. Set to 0 if the symbol succeeds the value.

23037        **char n\_sep\_by\_space**  
23038            Set to a value indicating the separation of the **currency\_symbol**, the sign string, and the  
23039            value for a negative formatted monetary quantity.

23040        **char p\_sign\_posn**  
23041            Set to a value indicating the positioning of the **positive\_sign** for a non-negative formatted  
23042            monetary quantity.

23043        **char n\_sign\_posn**  
23044            Set to a value indicating the positioning of the **negative\_sign** for a negative formatted  
23045            monetary quantity.

23046        **char int\_p\_cs\_precedes**  
23047            Set to 1 or 0 if the **int\_curr\_symbol** respectively precedes or succeeds the value for a non-  
23048            negative internationally formatted monetary quantity.

23049        **char int\_n\_cs\_precedes**  
23050            Set to 1 or 0 if the **int\_curr\_symbol** respectively precedes or succeeds the value for a  
23051            negative internationally formatted monetary quantity.

23052        **char int\_p\_sep\_by\_space**  
23053            Set to a value indicating the separation of the **int\_curr\_symbol**, the sign string, and the  
23054            value for a non-negative internationally formatted monetary quantity.

23055        **char int\_n\_sep\_by\_space**  
23056            Set to a value indicating the separation of the **int\_curr\_symbol**, the sign string, and the  
23057            value for a negative internationally formatted monetary quantity.

23058        **char int\_p\_sign\_posn**  
23059            Set to a value indicating the positioning of the **positive\_sign** for a non-negative  
23060            internationally formatted monetary quantity.

23061        **char int\_n\_sign\_posn**  
23062            Set to a value indicating the positioning of the **negative\_sign** for a negative internationally  
23063            formatted monetary quantity.

- 23064 The elements of **grouping** and **mon\_grouping** are interpreted according to the following:
- 23065 {CHAR\_MAX} No further grouping is to be performed.
- 23066 0 The previous element is to be repeatedly used for the remainder of the digits.
- 23067 *other* The integer value is the number of digits that comprise the current group. The  
23068 next element is examined to determine the size of the next group of digits  
23069 before the current group.
- 23070 The values of **p\_sep\_by\_space**, **n\_sep\_by\_space**, **int\_p\_sep\_by\_space**, and **int\_n\_sep\_by\_space**  
23071 are interpreted according to the following:
- 23072 0 No space separates the currency symbol and value.
- 23073 1 If the currency symbol and sign string are adjacent, a space separates them from the value;  
23074 otherwise, a space separates the currency symbol from the value.
- 23075 2 If the currency symbol and sign string are adjacent, a space separates them; otherwise, a  
23076 space separates the sign string from the value.
- 23077 For **int\_p\_sep\_by\_space** and **int\_n\_sep\_by\_space**, the fourth character of **int\_curr\_symbol** is  
23078 used instead of a space.
- 23079 The values of **p\_sign\_posn**, **n\_sign\_posn**, **int\_p\_sign\_posn**, and **int\_n\_sign\_posn** are  
23080 interpreted according to the following:
- 23081 0 Parentheses surround the quantity and **currency\_symbol** or **int\_curr\_symbol**.
- 23082 1 The sign string precedes the quantity and **currency\_symbol** or **int\_curr\_symbol**.
- 23083 2 The sign string succeeds the quantity and **currency\_symbol** or **int\_curr\_symbol**.
- 23084 3 The sign string immediately precedes the **currency\_symbol** or **int\_curr\_symbol**.
- 23085 4 The sign string immediately succeeds the **currency\_symbol** or **int\_curr\_symbol**.
- 23086 The implementation shall behave as if no function in this volume of IEEE Std 1003.1-2001 calls  
23087 *localeconv()*.
- 23088 cx The *localeconv()* function need not be reentrant. A function that is not required to be reentrant is  
23089 not required to be thread-safe.

#### 23090 RETURN VALUE

- 23091 The *localeconv()* function shall return a pointer to the filled-in object. The application shall not  
23092 modify the structure pointed to by the return value which may be overwritten by a subsequent  
23093 call to *localeconv()*. In addition, calls to *setlocale()* with the categories *LC\_ALL*, *LC\_MONETARY*,  
23094 or *LC\_NUMERIC* may overwrite the contents of the structure.

#### 23095 ERRORS

- 23096 No errors are defined.

#### 23097 EXAMPLES

- 23098 None.

#### 23099 APPLICATION USAGE

- 23100 The following table illustrates the rules which may be used by four countries to format monetary  
23101 quantities.

23102  
23103  
23104  
23105  
23106  
23107

| Country     | Positive Format            | Negative Format             | International Format |
|-------------|----------------------------|-----------------------------|----------------------|
| Italy       | L.1.230                    | -L.1.230                    | ITL.1.230            |
| Netherlands | F 1.234,56                 | F -1.234,56                 | NLG 1.234,56         |
| Norway      | kr1.234,56                 | kr1.234,56-                 | NOK 1.234,56         |
| Switzerland | SFr <sub>s</sub> .1,234.56 | SFr <sub>s</sub> .1,234.56C | CHF 1,234.56         |

23108 For these four countries, the respective values for the monetary members of the structure  
23109 returned by *localeconv()* are:

23110  
23111  
23112  
23113  
23114  
23115  
23116  
23117  
23118  
23119  
23120  
23121  
23122  
23123  
23124  
23125  
23126  
23127  
23128  
23129  
23130  
23131

|                           | Italy   | Netherlands | Norway | Switzerland          |
|---------------------------|---------|-------------|--------|----------------------|
| <b>int_curr_symbol</b>    | "ITL. " | "NLG "      | "NOK " | "CHF "               |
| <b>currency_symbol</b>    | "L. "   | "F"         | "kr"   | "SFr <sub>s</sub> ." |
| <b>mon_decimal_point</b>  | " "     | ","         | ","    | ."                   |
| <b>mon_thousands_sep</b>  | ". "    | ". "        | ". "   | "/ "                 |
| <b>mon_grouping</b>       | "\3 "   | "\3 "       | "\3 "  | "\3 "                |
| <b>positive_sign</b>      | " "     | " "         | " "    | " "                  |
| <b>negative_sign</b>      | " - "   | " - "       | " - "  | "C"                  |
| <b>int_frac_digits</b>    | 0       | 2           | 2      | 2                    |
| <b>frac_digits</b>        | 0       | 2           | 2      | 2                    |
| <b>p_cs_precedes</b>      | 1       | 1           | 1      | 1                    |
| <b>p_sep_by_space</b>     | 0       | 1           | 0      | 0                    |
| <b>n_cs_precedes</b>      | 1       | 1           | 1      | 1                    |
| <b>n_sep_by_space</b>     | 0       | 1           | 0      | 0                    |
| <b>p_sign_posn</b>        | 1       | 1           | 1      | 1                    |
| <b>n_sign_posn</b>        | 1       | 4           | 2      | 2                    |
| <b>int_p_cs_precedes</b>  | 1       | 1           | 1      | 1                    |
| <b>int_n_cs_precedes</b>  | 1       | 1           | 1      | 1                    |
| <b>int_p_sep_by_space</b> | 0       | 0           | 0      | 0                    |
| <b>int_n_sep_by_space</b> | 0       | 0           | 0      | 0                    |
| <b>int_p_sign_posn</b>    | 1       | 1           | 1      | 1                    |
| <b>int_n_sign_posn</b>    | 1       | 4           | 4      | 2                    |

23132 **RATIONALE**  
23133 None.

23134 **FUTURE DIRECTIONS**  
23135 None.

23136 **SEE ALSO**  
23137 *isalpha()*, *isascii()*, *nl\_langinfo()*, *printf()*, *scanf()*, *setlocale()*, *strcat()*, *strchr()*, *strcmp()*, *strcoll()*,  
23138 *strcpy()*, *strftime()*, *strlen()*, *strpbrk()*, *strspn()*, *strtok()*, *strxfrm()*, *strtod()*, the Base Definitions  
23139 volume of IEEE Std 1003.1-2001, <langinfo.h>, <locale.h>

23140 **CHANGE HISTORY**  
23141 First released in Issue 4. Derived from the ANSI C standard.

23142 **Issue 6**  
23143 A note indicating that this function need not be reentrant is added to the DESCRIPTION.  
23144 The RETURN VALUE section is rewritten to avoid use of the term ‘‘must’’.  
23145 This reference page is updated for alignment with the ISO/IEC 9899:1999 standard.  
23146 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

23147 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/31 is applied, removing references to  
23148 **int\_curr\_symbol** and updating the descriptions of **p\_sep\_by\_space** and **n\_sep\_by\_space**. These  
23149 changes are for alignment with the ISO C standard.

## 23150 NAME

23151 localtime, localtime\_r — convert a time value to a broken-down local time

## 23152 SYNOPSIS

23153 #include <time.h>

23154 struct tm \*localtime(const time\_t \*timer);

23155 TSF struct tm \*localtime\_r(const time\_t \*restrict timer,

23156 struct tm \*restrict result);

23157

## 23158 DESCRIPTION

23159 CX For *localtime()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23162 The *localtime()* function shall convert the time in seconds since the Epoch pointed to by *timer* into a broken-down time, expressed as a local time. The function corrects for the timezone and any seasonal time adjustments. Local timezone information is used as though *localtime()* calls *tzset()*.

23166 The relationship between a time in seconds since the Epoch used as an argument to *localtime()* and the **tm** structure (defined in the <time.h> header) is that the result shall be as specified in the expression given in the definition of seconds since the Epoch (see the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.14, Seconds Since the Epoch) corrected for timezone and any seasonal time adjustments, where the names in the structure and in the expression correspond.

23171 TSF The same relationship shall apply for *localtime\_r()*.

23172 CX The *localtime()* function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

23174 The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static objects: a broken-down time structure and an array of type **char**. Execution of any of the functions may overwrite the information returned in either of these objects by any of the other functions.

23178 TSF The *localtime\_r()* function shall convert the time in seconds since the Epoch pointed to by *timer* into a broken-down time stored in the structure to which *result* points. The *localtime\_r()* function shall also return a pointer to that same structure.

23181 Unlike *localtime()*, the reentrant version is not required to set *tzname*.

## 23182 RETURN VALUE

23183 Upon successful completion, the *localtime()* function shall return a pointer to the broken-down time structure. If an error is detected, *localtime()* shall return a null pointer and set *errno* to indicate the error.

23186 TSF Upon successful completion, *localtime\_r()* shall return a pointer to the structure pointed to by the argument *result*.

## 23188 ERRORS

23189 The *localtime()* function shall fail if:

23190 CX [EOVERFLOW] The result cannot be represented.

23191 **EXAMPLES**23192 **Getting the Local Date and Time**

23193 The following example uses the *time()* function to calculate the time elapsed, in seconds, since  
 23194 January 1, 1970 0:00 UTC (the Epoch), *localtime()* to convert that value to a broken-down time,  
 23195 and *asctime()* to convert the broken-down time values into a printable string.

```
23196 #include <stdio.h>
23197 #include <time.h>
23198 int main(void)
23199 {
23200 time_t result;
23201 result = time(NULL);
23202 printf("%s%ju secs since the Epoch\n",
23203 asctime(localtime(&result)),
23204 (uintmax_t)result);
23205 return(0);
23206 }
```

23207 This example writes the current time to *stdout* in a form like this:

```
23208 Wed Jun 26 10:32:15 1996
23209 835810335 secs since the Epoch
```

23210 **Getting the Modification Time for a File**

23211 The following example gets the modification time for a file. The *localtime()* function converts the  
 23212 **time\_t** value of the last modification date, obtained by a previous call to *stat()*, into a **tm**  
 23213 structure that contains the year, month, day, and so on.

```
23214 #include <time.h>
23215 ...
23216 struct stat statbuf;
23217 ...
23218 tm = localtime(&statbuf.st_mtime);
23219 ...
```

23220 **Timing an Event**

23221 The following example gets the current time, converts it to a string using *localtime()* and  
 23222 *asctime()*, and prints it to standard output using *fputs()*. It then prints the number of minutes to  
 23223 an event being timed.

```
23224 #include <time.h>
23225 #include <stdio.h>
23226 ...
23227 time_t now;
23228 int minutes_to_event;
23229 ...
23230 time(&now);
23231 printf("The time is ");
23232 fputs(asctime(localtime(&now)), stdout);
23233 printf("There are still %d minutes to the event.\n",
```

23234                   minutes\_to\_event);

23235                   ...

23236 **APPLICATION USAGE**

23237                   The *localtime\_r()* function is thread-safe and returns values in a user-supplied buffer instead of  
23238                   possibly using a static data area that may be overwritten by each call.

23239 **RATIONALE**

23240                   None.

23241 **FUTURE DIRECTIONS**

23242                   None.

23243 **SEE ALSO**

23244                   *asctime()*, *clock()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*,  
23245                   *utime()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**time.h**>

23246 **CHANGE HISTORY**

23247                   First released in Issue 1. Derived from Issue 1 of the SVID.

23248 **Issue 5**

23249                   A note indicating that the *localtime()* function need not be reentrant is added to the  
23250                   DESCRIPTION.

23251                   The *localtime\_r()* function is included for alignment with the POSIX Threads Extension.

23252 **Issue 6**

23253                   The *localtime\_r()* function is marked as part of the Thread-Safe Functions option.

23254                   Extensions beyond the ISO C standard are marked.

23255                   The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
23256                   its avoidance of possibly using a static data area.

23257                   The **restrict** keyword is added to the *localtime\_r()* prototype for alignment with the  
23258                   ISO/IEC 9899:1999 standard.

23259                   Examples are added.

23260                   IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/32 is applied, adding the  
23261                   [EOverflow] error.

23262 **NAME**

23263 lockf — record locking on files

23264 **SYNOPSIS**

23265 xSI #include &lt;unistd.h&gt;

23266 int lockf(int *fildes*, int *function*, off\_t *size*);

23267

23268 **DESCRIPTION**

23269 The *lockf()* function shall lock sections of a file with advisory-mode locks. Calls to *lockf()* from  
 23270 other threads which attempt to lock the locked file section shall either return an error value or  
 23271 block until the section becomes unlocked. All the locks for a process are removed when the  
 23272 process terminates. Record locking with *lockf()* shall be supported for regular files and may be  
 23273 supported for other files.

23274 The *fildes* argument is an open file descriptor. To establish a lock with this function, the file  
 23275 descriptor shall be opened with write-only permission (O\_WRONLY) or with read/write  
 23276 permission (O\_RDWR).

23277 The *function* argument is a control value which specifies the action to be taken. The permissible  
 23278 values for *function* are defined in <unistd.h> as follows:

23279

23280

23281

23282

23283

23284

| Function | Description                                  |
|----------|----------------------------------------------|
| F_ULOCK  | Unlock locked sections.                      |
| F_LOCK   | Lock a section for exclusive use.            |
| F_TLOCK  | Test and lock a section for exclusive use.   |
| F_TEST   | Test a section for locks by other processes. |

23285 F\_TEST shall detect if a lock by another process is present on the specified section.

23286 F\_LOCK and F\_TLOCK shall both lock a section of a file if the section is available.

23287 F\_ULOCK shall remove locks from a section of the file.

23288 The *size* argument is the number of contiguous bytes to be locked or unlocked. The section to be  
 23289 locked or unlocked starts at the current offset in the file and extends forward for a positive *size*  
 23290 or backward for a negative *size* (the preceding bytes up to but not including the current offset).  
 23291 If *size* is 0, the section from the current offset through the largest possible file offset shall be  
 23292 locked (that is, from the current offset through the present or any future end-of-file). An area  
 23293 need not be allocated to the file to be locked because locks may exist past the end-of-file.

23294 The sections locked with F\_LOCK or F\_TLOCK may, in whole or in part, contain or be contained  
 23295 by a previously locked section for the same process. When this occurs, or if adjacent locked  
 23296 sections would occur, the sections shall be combined into a single locked section. If the request  
 23297 would cause the number of locks to exceed a system-imposed limit, the request shall fail.

23298 F\_LOCK and F\_TLOCK requests differ only by the action taken if the section is not available.  
 23299 F\_LOCK shall block the calling thread until the section is available. F\_TLOCK shall cause the  
 23300 function to fail if the section is already locked by another process.

23301 File locks shall be released on first close by the locking process of any file descriptor for the file.

23302 F\_ULOCK requests may release (wholly or in part) one or more locked sections controlled by the  
 23303 process. Locked sections shall be unlocked starting at the current file offset through *size* bytes or  
 23304 to the end-of-file if *size* is (off\_t)0. When all of a locked section is not released (that is, when the  
 23305 beginning or end of the area to be unlocked falls within a locked section), the remaining portions  
 23306 of that section shall remain locked by the process. Releasing the center portion of a locked

- 23307 section shall cause the remaining locked beginning and end portions to become two separate  
 23308 locked sections. If the request would cause the number of locks in the system to exceed a  
 23309 system-imposed limit, the request shall fail.
- 23310 A potential for deadlock occurs if the threads of a process controlling a locked section are  
 23311 blocked by accessing another process' locked section. If the system detects that deadlock would  
 23312 occur, *lockf()* shall fail with an [EDEADLK] error.
- 23313 The interaction between *fcntl()* and *lockf()* locks is unspecified.
- 23314 Blocking on a section shall be interrupted by any signal.
- 23315 An F\_ULOCK request in which *size* is non-zero and the offset of the last byte of the requested  
 23316 section is the maximum value for an object of type *off\_t*, when the process has an existing lock  
 23317 in which *size* is 0 and which includes the last byte of the requested section, shall be treated as a  
 23318 request to unlock from the start of the requested section with a size equal to 0. Otherwise, an  
 23319 F\_ULOCK request shall attempt to unlock only the requested section.
- 23320 Attempting to lock a section of a file that is associated with a buffered stream produces  
 23321 unspecified results.
- 23322 **RETURN VALUE**
- 23323 Upon successful completion, *lockf()* shall return 0. Otherwise, it shall return -1, set *errno* to  
 23324 indicate an error, and existing locks shall not be changed.
- 23325 **ERRORS**
- 23326 The *lockf()* function shall fail if:
- 23327 [EBADF] The *fildev* argument is not a valid open file descriptor; or *function* is F\_LOCK  
 23328 or F\_TLOCK and *fildev* is not a valid file descriptor open for writing.
- 23329 [EACCES] or [EAGAIN]  
 23330 The *function* argument is F\_TLOCK or F\_TEST and the section is already  
 23331 locked by another process.
- 23332 [EDEADLK] The *function* argument is F\_LOCK and a deadlock is detected.
- 23333 [EINTR] A signal was caught during execution of the function.
- 23334 [EINVAL] The *function* argument is not one of F\_LOCK, F\_TLOCK, F\_TEST, or  
 23335 F\_ULOCK; or *size* plus the current file offset is less than 0.
- 23336 [EOVERFLOW] The offset of the first, or if *size* is not 0 then the last, byte in the requested  
 23337 section cannot be represented correctly in an object of type *off\_t*.
- 23338 The *lockf()* function may fail if:
- 23339 [EAGAIN] The *function* argument is F\_LOCK or F\_TLOCK and the file is mapped with  
 23340 *mmap()*.
- 23341 [EDEADLK] or [ENOLCK]  
 23342 The *function* argument is F\_LOCK, F\_TLOCK, or F\_ULOCK, and the request  
 23343 would cause the number of locks to exceed a system-imposed limit.
- 23344 [EOPNOTSUPP] or [EINVAL]  
 23345 The implementation does not support the locking of files of the type indicated  
 23346 by the *fildev* argument.

23347 **EXAMPLES**23348 **Locking a Portion of a File**

23349 In the following example, a file named `/home/cnd/mod1` is being modified. Other processes that  
 23350 use locking are prevented from changing it during this process. Only the first 10 000 bytes are  
 23351 locked, and the lock call fails if another process has any part of this area locked already.

```
23352 #include <fcntl.h>
23353 #include <unistd.h>
23354 int fildes;
23355 int status;
23356 ...
23357 fildes = open("/home/cnd/mod1", O_RDWR);
23358 status = lockf(fildes, F_TLOCK, (off_t)10000);
```

23359 **APPLICATION USAGE**

23360 Record-locking should not be used in combination with the *fopen()*, *fread()*, *fwrite()*, and other  
 23361 *stdio* functions. Instead, the more primitive, non-buffered functions (such as *open()*) should be  
 23362 used. Unexpected results may occur in processes that do buffering in the user address space. The  
 23363 process may later read/write data which is/was locked. The *stdio* functions are the most  
 23364 common source of unexpected buffering.

23365 The *alarm()* function may be used to provide a timeout facility in applications requiring it.

23366 **RATIONALE**

23367 None.

23368 **FUTURE DIRECTIONS**

23369 None.

23370 **SEE ALSO**

23371 *alarm()*, *chmod()*, *close()*, *creat()*, *fcntl()*, *fopen()*, *mmap()*, *open()*, *read()*, *write()*, the Base  
 23372 Definitions volume of IEEE Std 1003.1-2001, `<unistd.h>`

23373 **CHANGE HISTORY**

23374 First released in Issue 4, Version 2.

23375 **Issue 5**

23376 Moved from X/OPEN UNIX extension to BASE.

23377 Large File Summit extensions are added. In particular, the description of [EINVAL] is clarified  
 23378 and moved from optional to mandatory status.

23379 A note is added to the DESCRIPTION indicating the effects of attempting to lock a section of a  
 23380 file that is associated with a buffered stream.

23381 **Issue 6**

23382 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

## 23383 NAME

23384 log, logf, logl — natural logarithm function

## 23385 SYNOPSIS

23386 #include &lt;math.h&gt;

23387 double log(double x);

23388 float logf(float x);

23389 long double logl(long double x);

## 23390 DESCRIPTION

23391 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 23392 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23393 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23394 These functions shall compute the natural logarithm of their argument  $x$ ,  $\log_e(x)$ .

23395 An application wishing to check for error situations should set *errno* to zero and call  
 23396 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 23397 *fetetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 23398 zero, an error has occurred.

## 23399 RETURN VALUE

23400 Upon successful completion, these functions shall return the natural logarithm of  $x$ .

23401 If  $x$  is  $\pm 0$ , a pole error shall occur and *log()*, *logf()*, and *logl()* shall return  $-\text{HUGE\_VAL}$ ,  
 23402  $-\text{HUGE\_VALF}$ , and  $-\text{HUGE\_VALL}$ , respectively.

23403 MX For finite values of  $x$  that are less than 0, or if  $x$  is  $-\text{Inf}$ , a domain error shall occur, and either a  
 23404 NaN (if supported), or an implementation-defined value shall be returned.

23405 MX If  $x$  is NaN, a NaN shall be returned.23406 If  $x$  is 1,  $+0$  shall be returned.23407 If  $x$  is  $+\text{Inf}$ ,  $x$  shall be returned.

## 23408 ERRORS

23409 These functions shall fail if:

23410 MX Domain Error The finite value of  $x$  is negative, or  $x$  is  $-\text{Inf}$ .

23411 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 23412 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling  
 23413 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 23414 shall be raised.

23415 Pole Error The value of  $x$  is zero.

23416 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 23417 then *errno* shall be set to [ERANGE]. If the integer expression  
 23418 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the divide-by-  
 23419 zero floating-point exception shall be raised.

23420 **EXAMPLES**

23421 None.

23422 **APPLICATION USAGE**

23423 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
23424 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23425 **RATIONALE**

23426 None.

23427 **FUTURE DIRECTIONS**

23428 None.

23429 **SEE ALSO**

23430 *exp()*, *feclearexcept()*, *fetetestexcept()*, *isnan()*, *log10()*, *log1p()*, the Base Definitions volume of  
23431 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,  
23432 <math.h>

23433 **CHANGE HISTORY**

23434 First released in Issue 1. Derived from Issue 1 of the SVID.

23435 **Issue 5**

23436 The DESCRIPTION is updated to indicate how an application should check for an error. This  
23437 text was previously published in the APPLICATION USAGE section.

23438 **Issue 6**

23439 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

23440 The *logf()* and *logl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

23441 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
23442 revised to align with the ISO/IEC 9899:1999 standard.

23443 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
23444 marked.

23445 **NAME**

23446 log10, log10f, log10l — base 10 logarithm function

23447 **SYNOPSIS**

23448 #include &lt;math.h&gt;

23449 double log10(double x);

23450 float log10f(float x);

23451 long double log10l(long double x);

23452 **DESCRIPTION**

23453 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 23454 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23455 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23456 These functions shall compute the base 10 logarithm of their argument  $x$ ,  $\log_{10}(x)$ .

23457 An application wishing to check for error situations should set *errno* to zero and call  
 23458 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 23459 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 23460 zero, an error has occurred.

23461 **RETURN VALUE**23462 Upon successful completion, these functions shall return the base 10 logarithm of  $x$ .

23463 If  $x$  is  $\pm 0$ , a pole error shall occur and *log10()*, *log10f()*, and *log10l()* shall return  $-\text{HUGE\_VAL}$ ,  
 23464  $-\text{HUGE\_VALF}$ , and  $-\text{HUGE\_VALL}$ , respectively.

23465 **MX** For finite values of  $x$  that are less than 0, or if  $x$  is  $-\text{Inf}$ , a domain error shall occur, and either a  
 23466 NaN (if supported), or an implementation-defined value shall be returned.

23467 **MX** If  $x$  is NaN, a NaN shall be returned.23468 If  $x$  is 1,  $+0$  shall be returned.23469 If  $x$  is  $+\text{Inf}$ ,  $+\text{Inf}$  shall be returned.23470 **ERRORS**

23471 These functions shall fail if:

23472 **MX** Domain Error The finite value of  $x$  is negative, or  $x$  is  $-\text{Inf}$ .

23473 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 23474 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling  
 23475 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 23476 shall be raised.

23477 Pole Error The value of  $x$  is zero.

23478 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 23479 then *errno* shall be set to [ERANGE]. If the integer expression  
 23480 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the divide-by-  
 23481 zero floating-point exception shall be raised.

23482 **EXAMPLES**

23483 None.

23484 **APPLICATION USAGE**

23485 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`  
23486 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

23487 **RATIONALE**

23488 None.

23489 **FUTURE DIRECTIONS**

23490 None.

23491 **SEE ALSO**

23492 `feclearexcept()`, `fetestexcept()`, `isnan()`, `log()`, `pow()`, the Base Definitions volume of  
23493 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,  
23494 `<math.h>`

23495 **CHANGE HISTORY**

23496 First released in Issue 1. Derived from Issue 1 of the SVID.

23497 **Issue 5**

23498 The DESCRIPTION is updated to indicate how an application should check for an error. This  
23499 text was previously published in the APPLICATION USAGE section.

23500 **Issue 6**

23501 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

23502 The `log10f()` and `log10l()` functions are added for alignment with the ISO/IEC 9899:1999  
23503 standard.

23504 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
23505 revised to align with the ISO/IEC 9899:1999 standard.

23506 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
23507 marked.

## 23508 NAME

23509 log1p, log1pf, log1pl — compute a natural logarithm

## 23510 SYNOPSIS

23511 #include &lt;math.h&gt;

23512 double log1p(double x);

23513 float log1pf(float x);

23514 long double log1pl(long double x);

## 23515 DESCRIPTION

23516 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 23517 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23518 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23519 These functions shall compute  $\log_e(1.0 + x)$ .

23520 An application wishing to check for error situations should set *errno* to zero and call  
 23521 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 23522 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 23523 zero, an error has occurred.

## 23524 RETURN VALUE

23525 Upon successful completion, these functions shall return the natural logarithm of  $1.0 + x$ .

23526 If  $x$  is  $-1$ , a pole error shall occur and *log1p()*, *log1pf()*, and *log1pl()* shall return  $-\text{HUGE\_VAL}$ ,  
 23527  $-\text{HUGE\_VALF}$ , and  $-\text{HUGE\_VALL}$ , respectively.

23528 MX For finite values of  $x$  that are less than  $-1$ , or if  $x$  is  $-\text{Inf}$ , a domain error shall occur, and either a  
 23529 NaN (if supported), or an implementation-defined value shall be returned.

23530 MX If  $x$  is NaN, a NaN shall be returned.23531 If  $x$  is  $\pm 0$ , or  $+\text{Inf}$ ,  $x$  shall be returned.23532 If  $x$  is subnormal, a range error may occur and  $x$  should be returned.

## 23533 ERRORS

23534 These functions shall fail if:

23535 MX Domain Error The finite value of  $x$  is less than  $-1$ , or  $x$  is  $-\text{Inf}$ .

23536 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 23537 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling  
 23538 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 23539 shall be raised.

23540 Pole Error The value of  $x$  is  $-1$ .

23541 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 23542 then *errno* shall be set to [ERANGE]. If the integer expression  
 23543 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the divide-by-  
 23544 zero floating-point exception shall be raised.

23545 These functions may fail if:

23546 MX Range Error The value of  $x$  is subnormal.

23547 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 23548 then *errno* shall be set to [ERANGE]. If the integer expression  
 23549 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow  
 23550 floating-point exception shall be raised.

23551 **EXAMPLES**

23552 None.

23553 **APPLICATION USAGE**

23554 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`  
23555 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

23556 **RATIONALE**

23557 None.

23558 **FUTURE DIRECTIONS**

23559 None.

23560 **SEE ALSO**

23561 *feclearexcept()*, *fetestexcept()*, *log()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section  
23562 4.18, Treatment of Error Conditions for Mathematical Functions, <**math.h**>

23563 **CHANGE HISTORY**

23564 First released in Issue 4, Version 2.

23565 **Issue 5**

23566 Moved from X/OPEN UNIX extension to BASE.

23567 **Issue 6**

23568 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

23569 The *log1p()* function is no longer marked as an extension.

23570 The *log1pf()* and *log1pl()* functions are added for alignment with the ISO/IEC 9899:1999  
23571 standard.

23572 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
23573 revised to align with the ISO/IEC 9899:1999 standard.

23574 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
23575 marked.

23576 **NAME**

23577 log2, log2f, log2l — compute base 2 logarithm functions

23578 **SYNOPSIS**

23579 #include &lt;math.h&gt;

23580 double log2(double x);

23581 float log2f(float x);

23582 long double log2l(long double x);

23583 **DESCRIPTION**

23584 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 23585 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23586 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23587 These functions shall compute the base 2 logarithm of their argument  $x$ ,  $\log_2(x)$ .

23588 An application wishing to check for error situations should set *errno* to zero and call  
 23589 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 23590 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 23591 zero, an error has occurred.

23592 **RETURN VALUE**23593 Upon successful completion, these functions shall return the base 2 logarithm of  $x$ .

23594 If  $x$  is  $\pm 0$ , a pole error shall occur and *log2()*, *log2f()*, and *log2l()* shall return `-HUGE_VAL`,  
 23595 `-HUGE_VALF`, and `-HUGE_VALL`, respectively.

23596 **MX** For finite values of  $x$  that are less than 0, or if  $x$  is `-Inf`, a domain error shall occur, and either a  
 23597 NaN (if supported), or an implementation-defined value shall be returned.

23598 **MX** If  $x$  is NaN, a NaN shall be returned.23599 If  $x$  is 1, `+0` shall be returned.23600 If  $x$  is `+Inf`,  $x$  shall be returned.23601 **ERRORS**

23602 These functions shall fail if:

23603 **MX** Domain Error The finite value of  $x$  is less than zero, or  $x$  is `-Inf`.

23604 If the integer expression (`math_errhandling & MATH_ERRNO`) is non-zero,  
 23605 then *errno* shall be set to [EDOM]. If the integer expression (`math_errhandling`  
 23606 & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception  
 23607 shall be raised.

23608 Pole Error The value of  $x$  is zero.

23609 If the integer expression (`math_errhandling & MATH_ERRNO`) is non-zero,  
 23610 then *errno* shall be set to [ERANGE]. If the integer expression  
 23611 (`math_errhandling & MATH_ERREXCEPT`) is non-zero, then the divide-by-  
 23612 zero floating-point exception shall be raised.

23613 **EXAMPLES**

23614 None.

23615 **APPLICATION USAGE**

23616 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
23617 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23618 **RATIONALE**

23619 None.

23620 **FUTURE DIRECTIONS**

23621 None.

23622 **SEE ALSO**

23623 *feclearexcept()*, *fetestexcept()*, *log()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section  
23624 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

23625 **CHANGE HISTORY**

23626 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23627 **NAME**

23628 logb, logbf, logbl — radix-independent exponent

23629 **SYNOPSIS**

23630 #include &lt;math.h&gt;

23631 double logb(double x);

23632 float logbf(float x);

23633 long double logbl(long double x);

23634 **DESCRIPTION**

23635 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 23636 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23637 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23638 These functions shall compute the exponent of  $x$ , which is the integral part of  $\log_r |x|$ , as a  
 23639 signed floating-point value, for non-zero  $x$ , where  $r$  is the radix of the machine's floating-point  
 23640 arithmetic, which is the value of FLT\_RADIX defined in the <float.h> header.

23641 If  $x$  is subnormal it is treated as though it were normalized; thus for finite positive  $x$ :

23642  $1 \leq x * FLT\_RADIX^{-\log_b(x)} < FLT\_RADIX$

23643 An application wishing to check for error situations should set *errno* to zero and call  
 23644 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 23645 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 23646 zero, an error has occurred.

23647 **RETURN VALUE**

23648 Upon successful completion, these functions shall return the exponent of  $x$ .

23649 If  $x$  is  $\pm 0$ , a pole error shall occur and *logb*(), *logbf*(), and *logbl*() shall return  $-\text{HUGE\_VAL}$ ,  
 23650  $-\text{HUGE\_VALF}$ , and  $-\text{HUGE\_VALL}$ , respectively.

23651 **MX** If  $x$  is NaN, a NaN shall be returned.

23652 If  $x$  is  $\pm\text{Inf}$ ,  $+\text{Inf}$  shall be returned.

23653 **ERRORS**

23654 These functions shall fail if:

23655 Pole Error The value of  $x$  is  $\pm 0$ .

23656 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 23657 then *errno* shall be set to [ERANGE]. If the integer expression  
 23658 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the divide-by-  
 23659 zero floating-point exception shall be raised.

23660 **EXAMPLES**

23661 None.

23662 **APPLICATION USAGE**

23663 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 23664 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23665 **RATIONALE**

23666 None.

23667 **FUTURE DIRECTIONS**

23668 None.

23669 **SEE ALSO**23670 *feclearexcept()*, *fetestexcept()*, *ilogb()*, *scalb()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
23671 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <float.h>, <math.h>23672 **CHANGE HISTORY**

23673 First released in Issue 4, Version 2.

23674 **Issue 5**

23675 Moved from X/OPEN UNIX extension to BASE.

23676 **Issue 6**23677 The *logb()* function is no longer marked as an extension.23678 The *logbf()* and *logbl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.23679 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
23680 revised to align with the ISO/IEC 9899:1999 standard.23681 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
23682 marked.

**23683 NAME**

23684       logf, logl — natural logarithm function

**23685 SYNOPSIS**

23686       #include <math.h>

23687       float logf(float x);

23688       long double logl(long double x);

**23689 DESCRIPTION**

23690       Refer to *log()*.

23691 **NAME**

23692 longjmp — non-local goto

23693 **SYNOPSIS**

23694 #include &lt;setjmp.h&gt;

23695 void longjmp(jmp\_buf env, int val);

23696 **DESCRIPTION**

23697 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 23698 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23699 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23700 The *longjmp()* function shall restore the environment saved by the most recent invocation of  
 23701 *setjmp()* in the same thread, with the corresponding **jmp\_buf** argument. If there is no such  
 23702 invocation, or if the function containing the invocation of *setjmp()* has terminated execution in  
 23703 the interim, or if the invocation of *setjmp()* was within the scope of an identifier with variably  
 23704 **CX** modified type and execution has left that scope in the interim, the behavior is undefined. It is  
 23705 unspecified whether *longjmp()* restores the signal mask, leaves the signal mask unchanged, or  
 23706 restores it to its value at the time *setjmp()* was called.

23707 All accessible objects have values, and all other components of the abstract machine have state  
 23708 (for example, floating-point status flags and open files), as of the time *longjmp()* was called,  
 23709 except that the values of objects of automatic storage duration are unspecified if they meet all  
 23710 the following conditions:

- 23711 • They are local to the function containing the corresponding *setjmp()* invocation.
- 23712 • They do not have volatile-qualified type.
- 23713 • They are changed between the *setjmp()* invocation and *longjmp()* call.

23714 **CX** As it bypasses the usual function call and return mechanisms, *longjmp()* shall execute correctly  
 23715 in contexts of interrupts, signals, and any of their associated functions. However, if *longjmp()* is  
 23716 invoked from a nested signal handler (that is, from a function invoked as a result of a signal  
 23717 raised during the handling of another signal), the behavior is undefined.

23718 The effect of a call to *longjmp()* where initialization of the **jmp\_buf** structure was not performed  
 23719 in the calling thread is undefined.

23720 **RETURN VALUE**

23721 After *longjmp()* is completed, program execution continues as if the corresponding invocation of  
 23722 *setjmp()* had just returned the value specified by *val*. The *longjmp()* function shall not cause  
 23723 *setjmp()* to return 0; if *val* is 0, *setjmp()* shall return 1.

23724 **ERRORS**

23725 No errors are defined.

23726 **EXAMPLES**

23727 None.

23728 **APPLICATION USAGE**

23729 Applications whose behavior depends on the value of the signal mask should not use *longjmp()*  
 23730 and *setjmp()*, since their effect on the signal mask is unspecified, but should instead use the  
 23731 *siglongjmp()* and *sigsetjmp()* functions (which can save and restore the signal mask under  
 23732 application control).

23733 **RATIONALE**

23734 None.

23735 **FUTURE DIRECTIONS**

23736 None.

23737 **SEE ALSO**

23738 *setjmp()*, *sigaction()*, *siglongjmp()*, *sigsetjmp()*, the Base Definitions volume of  
23739 IEEE Std 1003.1-2001, <**setjmp.h**>

23740 **CHANGE HISTORY**

23741 First released in Issue 1. Derived from Issue 1 of the SVID.

23742 **Issue 5**

23743 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

23744 **Issue 6**

23745 Extensions beyond the ISO C standard are marked.

23746 The following new requirements on POSIX implementations derive from alignment with the  
23747 Single UNIX Specification:

- 23748 • The DESCRIPTION now explicitly makes *longjmp()*'s effect on the signal mask unspecified.

23749 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

23750 **NAME**

23751 lrand48 — generate uniformly distributed pseudo-random non-negative long integers

23752 **SYNOPSIS**

```
23753 xSI #include <stdlib.h>
```

```
23754 long lrand48(void);
```

23755

23756 **DESCRIPTION**

23757 Refer to *drand48()*.

23758 **NAME**

23759 lrint, lrintf, lrintl — round to nearest integer value using current rounding direction

23760 **SYNOPSIS**

```
23761 #include <math.h>
23762 long lrint(double x);
23763 long lrintf(float x);
23764 long lrintl(long double x);
```

23765 **DESCRIPTION**

23766 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 23767 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23768 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23769 These functions shall round their argument to the nearest integer value, rounding according to  
 23770 the current rounding direction.

23771 An application wishing to check for error situations should set *errno* to zero and call  
 23772 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 23773 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 23774 zero, an error has occurred.

23775 **RETURN VALUE**

23776 Upon successful completion, these functions shall return the rounded integer value.

23777 **MX** If *x* is NaN, a domain error shall occur and an unspecified value is returned.23778 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.23779 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

23780 If the correct value is positive and too large to represent as a **long**, a domain error shall occur  
 23781 and an unspecified value is returned.

23782 If the correct value is negative and too large to represent as a **long**, a domain error shall occur  
 23783 and an unspecified value is returned.

23784 **ERRORS**

23785 These functions shall fail if:

23786 **MX** Domain Error The *x* argument is NaN or ±Inf, or the correct value is not representable as an  
 23787 integer.

23788 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 23789 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling  
 23790 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 23791 shall be raised.

23792 **EXAMPLES**

23793 None.

23794 **APPLICATION USAGE**

23795 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 23796 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23797 **RATIONALE**

23798 These functions provide floating-to-integer conversions. They round according to the current  
 23799 rounding direction. If the rounded value is outside the range of the return type, the numeric  
 23800 result is unspecified and the invalid floating-point exception is raised. When they raise no other  
 23801 floating-point exception and the result differs from the argument, they raise the inexact

23802 floating-point exception.

23803 **FUTURE DIRECTIONS**

23804 None.

23805 **SEE ALSO**

23806 *feclearexcept()*, *fetetestexcept()*, *llrint()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
23807 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

23808 **CHANGE HISTORY**

23809 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

## 23810 NAME

23811 lround, lroundf, lroundl — round to nearest integer value

## 23812 SYNOPSIS

23813 #include <math.h>

23814 long lround(double x);

23815 long lroundf(float x);

23816 long lroundl(long double x);

## 23817 DESCRIPTION

23818 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
23819 conflict between the requirements described here and the ISO C standard is unintentional. This  
23820 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

23821 These functions shall round their argument to the nearest integer value, rounding halfway cases  
23822 away from zero, regardless of the current rounding direction.

23823 An application wishing to check for error situations should set *errno* to zero and call  
23824 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
23825 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
23826 zero, an error has occurred.

## 23827 RETURN VALUE

23828 Upon successful completion, these functions shall return the rounded integer value.

23829 MX If *x* is NaN, a domain error shall occur and an unspecified value is returned.

23830 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

23831 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

23832 If the correct value is positive and too large to represent as a **long**, a domain error shall occur  
23833 and an unspecified value is returned.

23834 If the correct value is negative and too large to represent as a **long**, a domain error shall occur  
23835 and an unspecified value is returned.

## 23836 ERRORS

23837 These functions shall fail if:

23838 MX Domain Error The *x* argument is NaN or ±Inf, or the correct value is not representable as an  
23839 integer.

23840 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
23841 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling  
23842 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
23843 shall be raised.

## 23844 EXAMPLES

23845 None.

## 23846 APPLICATION USAGE

23847 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
23848 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

## 23849 RATIONALE

23850 These functions differ from the *lrint*() functions in the default rounding direction, with the  
23851 *lround*() functions rounding halfway cases away from zero and needing not to raise the inexact  
23852 floating-point exception for non-integer arguments that round to within the range of the return  
23853 type.

23854 **FUTURE DIRECTIONS**

23855 None.

23856 **SEE ALSO**

23857 *feclearexcept()*, *fetetestexcept()*, *llround()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
23858 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

23859 **CHANGE HISTORY**

23860 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23861 **NAME**

23862 lsearch, lfind — linear search and update

23863 **SYNOPSIS**

23864 XSI #include &lt;search.h&gt;

23865 void \*lsearch(const void \*key, void \*base, size\_t \*nel, size\_t width,  
23866 int (\*compar)(const void \*, const void \*));23867 void \*lfind(const void \*key, const void \*base, size\_t \*nel,  
23868 size\_t width, int (\*compar)(const void \*, const void \*));

23869

23870 **DESCRIPTION**

23871 The *lsearch()* function shall linearly search the table and return a pointer into the table for the  
 23872 matching entry. If the entry does not occur, it shall be added at the end of the table. The *key*  
 23873 argument points to the entry to be sought in the table. The *base* argument points to the first  
 23874 element in the table. The *width* argument is the size of an element in bytes. The *nel* argument  
 23875 points to an integer containing the current number of elements in the table. The integer to which  
 23876 *nel* points shall be incremented if the entry is added to the table. The *compar* argument points to  
 23877 a comparison function which the application shall supply (for example, *strcmp()*). It is called  
 23878 with two arguments that point to the elements being compared. The application shall ensure  
 23879 that the function returns 0 if the elements are equal, and non-zero otherwise.

23880 The *lfind()* function shall be equivalent to *lsearch()*, except that if the entry is not found, it is not  
 23881 added to the table. Instead, a null pointer is returned.

23882 **RETURN VALUE**

23883 If the searched for entry is found, both *lsearch()* and *lfind()* shall return a pointer to it. Otherwise,  
 23884 *lfind()* shall return a null pointer and *lsearch()* shall return a pointer to the newly added element.

23885 Both functions shall return a null pointer in case of error.

23886 **ERRORS**

23887 No errors are defined.

23888 **EXAMPLES**23889 **Storing Strings in a Table**

23890 This fragment reads in less than or equal to TABSIZE strings of length less than or equal to  
 23891 ELSIZE and stores them in a table, eliminating duplicates.

23892 #include &lt;stdio.h&gt;

23893 #include &lt;string.h&gt;

23894 #include &lt;search.h&gt;

23895 #define TABSIZE 50

23896 #define ELSIZE 120

23897 ...

23898 char line[ELSIZE], tab[TABSIZE][ELSIZE];

23899 size\_t nel = 0;

23900 ...

23901 while (fgets(line, ELSIZE, stdin) != NULL &amp;&amp; nel &lt; TABSIZE)

23902 (void) lsearch(line, tab, &amp;nel,

23903 ELSIZE, (int (\*)(const void \*, const void \*)) strcmp);

23904 ...

**23905 Finding a Matching Entry**

23906 The following example finds any line that reads "This is a test."

```
23907 #include <search.h>
23908 #include <string.h>
23909 ...
23910 char line[ELSIZE], tab[TABSIZE][ELSIZE];
23911 size_t nel = 0;
23912 char *findline;
23913 void *entry;
23914
23915 findline = "This is a test.\n";
23916
23917 entry = lfind(findline, tab, &nel, ELSIZE, (
23918 int (*)(const void *, const void *)) strcmp);
```

**23917 APPLICATION USAGE**

23918 The comparison function need not compare every byte, so arbitrary data may be contained in  
23919 the elements in addition to the values being compared.

23920 Undefined results can occur if there is not enough room in the table to add a new item.

**23921 RATIONALE**

23922 None.

**23923 FUTURE DIRECTIONS**

23924 None.

**23925 SEE ALSO**

23926 *hcreate()*, *tsearch()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**search.h**>

**23927 CHANGE HISTORY**

23928 First released in Issue 1. Derived from Issue 1 of the SVID.

**23929 Issue 6**

23930 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

23931 **NAME**

23932 lseek — move the read/write file offset

23933 **SYNOPSIS**

23934 #include &lt;unistd.h&gt;

23935 off\_t lseek(int *fildev*, off\_t *offset*, int *whence*);23936 **DESCRIPTION**23937 The *lseek()* function shall set the file offset for the open file description associated with the file  
23938 descriptor *fildev*, as follows:

- 23939 • If *whence* is SEEK\_SET, the file offset shall be set to *offset* bytes.
- 23940 • If *whence* is SEEK\_CUR, the file offset shall be set to its current location plus *offset*.
- 23941 • If *whence* is SEEK\_END, the file offset shall be set to the size of the file plus *offset*.

23942 The symbolic constants SEEK\_SET, SEEK\_CUR, and SEEK\_END are defined in &lt;unistd.h&gt;.

23943 The behavior of *lseek()* on devices which are incapable of seeking is implementation-defined.  
23944 The value of the file offset associated with such a device is undefined.23945 The *lseek()* function shall allow the file offset to be set beyond the end of the existing data in the  
23946 file. If data is later written at this point, subsequent reads of data in the gap shall return bytes  
23947 with the value 0 until data is actually written into the gap.23948 The *lseek()* function shall not, by itself, extend the size of a file.23949 SHM If *fildev* refers to a shared memory object, the result of the *lseek()* function is unspecified.23950 TYM If *fildev* refers to a typed memory object, the result of the *lseek()* function is unspecified.23951 **RETURN VALUE**23952 Upon successful completion, the resulting offset, as measured in bytes from the beginning of the  
23953 file, shall be returned. Otherwise, (off\_t)-1 shall be returned, *errno* shall be set to indicate the  
23954 error, and the file offset shall remain unchanged.23955 **ERRORS**23956 The *lseek()* function shall fail if:

- |       |             |                                                                                                                                                        |
|-------|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| 23957 | [EBADF]     | The <i>fildev</i> argument is not an open file descriptor.                                                                                             |
| 23958 | [EINVAL]    | The <i>whence</i> argument is not a proper value, or the resulting file offset would be negative for a regular file, block special file, or directory. |
| 23959 |             |                                                                                                                                                        |
| 23960 | [EOVERFLOW] | The resulting file offset would be a value which cannot be represented correctly in an object of type <b>off_t</b> .                                   |
| 23961 |             |                                                                                                                                                        |
| 23962 | [ESPIPE]    | The <i>fildev</i> argument is associated with a pipe, FIFO, or socket.                                                                                 |

23963 **EXAMPLES**

23964 None.

23965 **APPLICATION USAGE**

23966 None.

23967 **RATIONALE**23968 The ISO C standard includes the functions *fgetpos()* and *fsetpos()*, which work on very large files  
23969 by use of a special positioning type.23970 Although *lseek()* may position the file offset beyond the end of the file, this function does not  
23971 itself extend the size of the file. While the only function in IEEE Std 1003.1-2001 that may directly

23972 extend the size of the file is *write()*, *truncate()*, and *ftruncate()*, several functions originally  
 23973 derived from the ISO C standard, such as *fwrite()*, *fprintf()*, and so on, may do so (by causing  
 23974 calls on *write()*).

23975 An invalid file offset that would cause [EINVAL] to be returned may be both implementation-  
 23976 defined and device-dependent (for example, memory may have few invalid values). A negative  
 23977 file offset may be valid for some devices in some implementations.

23978 The POSIX.1-1990 standard did not specifically prohibit *lseek()* from returning a negative offset.  
 23979 Therefore, an application was required to clear *errno* prior to the call and check *errno* upon return  
 23980 to determine whether a return value of (*off\_t*)-1 is a negative offset or an indication of an error  
 23981 condition. The standard developers did not wish to require this action on the part of a  
 23982 conforming application, and chose to require that *errno* be set to [EINVAL] when the resulting  
 23983 file offset would be negative for a regular file, block special file, or directory.

#### 23984 FUTURE DIRECTIONS

23985 None.

#### 23986 SEE ALSO

23987 *open()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**sys/types.h**>, <**unistd.h**>

#### 23988 CHANGE HISTORY

23989 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 23990 Issue 5

23991 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

23992 Large File Summit extensions are added.

#### 23993 Issue 6

23994 In the SYNOPSIS, the optional include of the <**sys/types.h**> header is removed.

23995 The following new requirements on POSIX implementations derive from alignment with the  
 23996 Single UNIX Specification:

23997 • The requirement to include <**sys/types.h**> has been removed. Although <**sys/types.h**> was  
 23998 required for conforming implementations of previous POSIX specifications, it was not  
 23999 required for UNIX applications.

24000 • The [EOVERFLOW] error condition is added. This change is to support large files.

24001 An additional [ESPIPE] error condition is added for sockets.

24002 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
 24003 *lseek()* results are unspecified for typed memory objects.

## 24004 NAME

24005 lstat — get symbolic link status

## 24006 SYNOPSIS

24007 #include &lt;sys/stat.h&gt;

24008 int lstat(const char \*restrict path, struct stat \*restrict buf);

## 24009 DESCRIPTION

24010 The *lstat()* function shall be equivalent to *stat()*, except when *path* refers to a symbolic link. In  
 24011 that case *lstat()* shall return information about the link, while *stat()* shall return information  
 24012 about the file the link references.

24013 For symbolic links, the *st\_mode* member shall contain meaningful information when used with  
 24014 the file type macros, and the *st\_size* member shall contain the length of the pathname contained  
 24015 in the symbolic link. File mode bits and the contents of the remaining members of the **stat**  
 24016 structure are unspecified. The value returned in the *st\_size* member is the length of the contents  
 24017 of the symbolic link, and does not count any trailing null.

## 24018 RETURN VALUE

24019 Upon successful completion, *lstat()* shall return 0. Otherwise, it shall return  $-1$  and set *errno* to  
 24020 indicate the error.

## 24021 ERRORS

24022 The *lstat()* function shall fail if:

24023 [EACCES] A component of the path prefix denies search permission.

24024 [EIO] An error occurred while reading from the file system.

24025 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 24026 argument.

24027 [ENAMETOOLONG]

24028 The length of a pathname exceeds {PATH\_MAX} or a pathname component is  
 24029 longer than {NAME\_MAX}.

24030 [ENOTDIR] A component of the path prefix is not a directory.

24031 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

24032 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file  
 24033 serial number cannot be represented correctly in the structure pointed to by  
 24034 *buf*.

24035 The *lstat()* function may fail if:

24036 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 24037 resolution of the *path* argument.

24038 [ENAMETOOLONG]

24039 As a result of encountering a symbolic link in resolution of the *path* argument,  
 24040 the length of the substituted pathname string exceeded {PATH\_MAX}.

24041 [EOVERFLOW] One of the members is too large to store into the structure pointed to by the  
 24042 *buf* argument.

24043 **EXAMPLES**24044 **Obtaining Symbolic Link Status Information**

24045 The following example shows how to obtain status information for a symbolic link named  
24046 **/modules/pass1**. The structure variable *buffer* is defined for the **stat** structure. If the *path*  
24047 argument specified the filename for the file pointed to by the symbolic link (**/home/cnd/mod1**),  
24048 the results of calling the function would be the same as those returned by a call to the *stat()*  
24049 function.

```
24050 #include <sys/stat.h>
24051 struct stat buffer;
24052 int status;
24053 ...
24054 status = lstat("/modules/pass1", &buffer);
```

24055 **APPLICATION USAGE**

24056 None.

24057 **RATIONALE**

24058 The *lstat()* function is not required to update the time-related fields if the named file is not a  
24059 symbolic link. While the *st\_uid*, *st\_gid*, *st\_atime*, *st\_mtime*, and *st\_ctime* members of the **stat**  
24060 structure may apply to a symbolic link, they are not required to do so. No functions in  
24061 IEEE Std 1003.1-2001 are required to maintain any of these time fields.

24062 **FUTURE DIRECTIONS**

24063 None.

24064 **SEE ALSO**

24065 *lstat()*, *readlink()*, *stat()*, *symlink()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
24066 **<sys/stat.h>**

24067 **CHANGE HISTORY**

24068 First released in Issue 4, Version 2.

24069 **Issue 5**

24070 Moved from X/OPEN UNIX extension to BASE.

24071 Large File Summit extensions are added.

24072 **Issue 6**

24073 The following changes were made to align with the IEEE P1003.1a draft standard:

- 24074 • This function is now mandatory.
- 24075 • The [ELOOP] optional error condition is added.

24076 The **restrict** keyword is added to the *lstat()* prototype for alignment with the ISO/IEC 9899:1999  
24077 standard.

## 24078 NAME

24079 makecontext, swapcontext — manipulate user contexts

## 24080 SYNOPSIS

```

24081 xSI #include <ucontext.h>
24082 void makecontext(ucontext_t *ucp, void (*func)(void),
24083 int argc, ...);
24084 int swapcontext(ucontext_t *restrict oucp,
24085 const ucontext_t *restrict ucp);
24086

```

## 24087 DESCRIPTION

24088 The *makecontext()* function shall modify the context specified by *ucp*, which has been initialized  
 24089 using *getcontext()*. When this context is resumed using *swapcontext()* or *setcontext()*, program  
 24090 execution shall continue by calling *func*, passing it the arguments that follow *argc* in the  
 24091 *makecontext()* call.

24092 Before a call is made to *makecontext()*, the application shall ensure that the context being  
 24093 modified has a stack allocated for it. The application shall ensure that the value of *argc* matches  
 24094 the number of arguments of type **int** passed to *func*; otherwise, the behavior is undefined.

24095 The *uc\_link* member is used to determine the context that shall be resumed when the context  
 24096 being modified by *makecontext()* returns. The application shall ensure that the *uc\_link* member is  
 24097 initialized prior to the call to *makecontext()*.

24098 The *swapcontext()* function shall save the current context in the context structure pointed to by  
 24099 *oucp* and shall set the context to the context structure pointed to by *ucp*.

## 24100 RETURN VALUE

24101 Upon successful completion, *swapcontext()* shall return 0. Otherwise,  $-1$  shall be returned and  
 24102 *errno* set to indicate the error.

## 24103 ERRORS

24104 The *swapcontext()* function shall fail if:

24105 [ENOMEM] The *ucp* argument does not have enough stack left to complete the operation.

## 24106 EXAMPLES

24107 The following example illustrates the use of *makecontext()*:

```

24108 #include <stdio.h>
24109 #include <ucontext.h>
24110 static ucontext_t ctx[3];
24111 static void
24112 f1 (void)
24113 {
24114 puts("start f1");
24115 swapcontext(&ctx[1], &ctx[2]);
24116 puts("finish f1");
24117 }
24118 static void
24119 f2 (void)
24120 {
24121 puts("start f2");
24122 swapcontext(&ctx[2], &ctx[1]);

```

```

24123 puts("finish f2");
24124 }
24125 int
24126 main (void)
24127 {
24128 char st1[8192];
24129 char st2[8192];

24130 getcontext(&ctx[1]);
24131 ctx[1].uc_stack.ss_sp = st1;
24132 ctx[1].uc_stack.ss_size = sizeof st1;
24133 ctx[1].uc_link = &ctx[0];
24134 makecontext(&ctx[1], f1, 0);

24135 getcontext(&ctx[2]);
24136 ctx[2].uc_stack.ss_sp = st2;
24137 ctx[2].uc_stack.ss_size = sizeof st2;
24138 ctx[2].uc_link = &ctx[1];
24139 makecontext(&ctx[2], f2, 0);

24140 swapcontext(&ctx[0], &ctx[2]);
24141 return 0;
24142 }

```

**24143 APPLICATION USAGE**

24144 None.

**24145 RATIONALE**

24146 None.

**24147 FUTURE DIRECTIONS**

24148 None.

**24149 SEE ALSO**

24150 *exit()*, *getcontext()*, *sigaction()*, *sigprocmask()*, the Base Definitions volume of  
 24151 IEEE Std 1003.1-2001, <**ucontext.h**>

**24152 CHANGE HISTORY**

24153 First released in Issue 4, Version 2.

**24154 Issue 5**

24155 Moved from X/OPEN UNIX extension to BASE.

24156 In the ERRORS section, the description of [ENOMEM] is changed to apply to *swapcontext()* only.

**24157 Issue 6**

24158 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

24159 The **restrict** keyword is added to the *swapcontext()* prototype for alignment with the  
 24160 ISO/IEC 9899:1999 standard.

24161 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/33 is applied, clarifying that the  
 24162 arguments passed to *func* are of type **int**.

24163 **NAME**

24164 malloc — a memory allocator

24165 **SYNOPSIS**

24166 #include &lt;stdlib.h&gt;

24167 void \*malloc(size\_t size);

24168 **DESCRIPTION**

24169 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
24170 conflict between the requirements described here and the ISO C standard is unintentional. This  
24171 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24172 The *malloc()* function shall allocate unused space for an object whose size in bytes is specified by  
24173 *size* and whose value is unspecified.

24174 The order and contiguity of storage allocated by successive calls to *malloc()* is unspecified. The  
24175 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to  
24176 a pointer to any type of object and then used to access such an object in the space allocated (until  
24177 the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object  
24178 disjoint from any other object. The pointer returned points to the start (lowest byte address) of  
24179 the allocated space. If the space cannot be allocated, a null pointer shall be returned. If the size of  
24180 the space requested is 0, the behavior is implementation-defined: the value returned shall be  
24181 either a null pointer or a unique pointer.

24182 **RETURN VALUE**

24183 Upon successful completion with *size* not equal to 0, *malloc()* shall return a pointer to the  
24184 allocated space. If *size* is 0, either a null pointer or a unique pointer that can be successfully  
24185 CX passed to *free()* shall be returned. Otherwise, it shall return a null pointer and set *errno* to  
24186 indicate the error.

24187 **ERRORS**24188 The *malloc()* function shall fail if:

24189 CX [ENOMEM] Insufficient storage space is available.

24190 **EXAMPLES**

24191 None.

24192 **APPLICATION USAGE**

24193 None.

24194 **RATIONALE**

24195 None.

24196 **FUTURE DIRECTIONS**

24197 None.

24198 **SEE ALSO**24199 *calloc()*, *free()*, *realloc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>24200 **CHANGE HISTORY**

24201 First released in Issue 1. Derived from Issue 1 of the SVID.

24202 **Issue 6**

24203 Extensions beyond the ISO C standard are marked.

24204 The following new requirements on POSIX implementations derive from alignment with the  
24205 Single UNIX Specification:

24206

- In the RETURN VALUE section, the requirement to set *errno* to indicate an error is added.

24207

- The [ENOMEM] error condition is added.

24208 **NAME**

24209           mblen — get number of bytes in a character

24210 **SYNOPSIS**

24211           #include &lt;stdlib.h&gt;

24212           int mblen(const char \*s, size\_t n);

24213 **DESCRIPTION**

24214 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 24215 conflict between the requirements described here and the ISO C standard is unintentional. This  
 24216 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24217       If *s* is not a null pointer, *mblen()* shall determine the number of bytes constituting the character  
 24218 pointed to by *s*. Except that the shift state of *mbtowc()* is not affected, it shall be equivalent to:

24219       mbtowc((wchar\_t \*)0, s, n);

24220       The implementation shall behave as if no function defined in this volume of  
 24221 IEEE Std 1003.1-2001 calls *mblen()*.

24222       The behavior of this function is affected by the *LC\_CTYPE* category of the current locale. For a  
 24223 state-dependent encoding, this function shall be placed into its initial state by a call for which its  
 24224 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null  
 24225 pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a  
 24226 null pointer shall cause this function to return a non-zero value if encodings have state  
 24227 dependency, and 0 otherwise. If the implementation employs special bytes to change the shift  
 24228 state, these bytes shall not produce separate wide-character codes, but shall be grouped with an  
 24229 adjacent character. Changing the *LC\_CTYPE* category causes the shift state of this function to be  
 24230 unspecified.

24231 **RETURN VALUE**

24232       If *s* is a null pointer, *mblen()* shall return a non-zero or 0 value, if character encodings,  
 24233 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mblen()* shall  
 24234 either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the  
 24235 character (if the next *n* or fewer bytes form a valid character), or return -1 (if they do not form a  
 24236 valid character) and may set *errno* to indicate the error. In no case shall the value returned be  
 24237 greater than *n* or the value of the {MB\_CUR\_MAX} macro.

24238 **ERRORS**24239       The *mblen()* function may fail if:24240 **XSI**       [EILSEQ]       Invalid character sequence is detected.24241 **EXAMPLES**

24242       None.

24243 **APPLICATION USAGE**

24244       None.

24245 **RATIONALE**

24246       None.

24247 **FUTURE DIRECTIONS**

24248       None.

24249 **SEE ALSO**

24250 *mbtowc()*, *mbstowcs()*, *wctomb()*, *wcstombs()*, the Base Definitions volume of  
24251 IEEE Std 1003.1-2001, <stdlib.h>

24252 **CHANGE HISTORY**

24253 First released in Issue 4. Aligned with the ISO C standard.

24254 **NAME**

24255 `mbrlen` — get number of bytes in a character (restartable)

24256 **SYNOPSIS**

24257 `#include <wchar.h>`

24258 `size_t mbrlen(const char *restrict s, size_t n,`  
 24259 `mbstate_t *restrict ps);`

24260 **DESCRIPTION**

24261 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 24262 conflict between the requirements described here and the ISO C standard is unintentional. This  
 24263 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24264 If *s* is not a null pointer, `mbrlen()` shall determine the number of bytes constituting the character  
 24265 pointed to by *s*. It shall be equivalent to:

```
24266 mbstate_t internal;
24267 mbrtowc(NULL, s, n, ps != NULL ? ps : &internal);
```

24268 If *ps* is a null pointer, the `mbrlen()` function shall use its own internal **mbstate\_t** object, which is  
 24269 initialized at program start-up to the initial conversion state. Otherwise, the **mbstate\_t** object  
 24270 pointed to by *ps* shall be used to completely describe the current conversion state of the  
 24271 associated character sequence. The implementation shall behave as if no function defined in this  
 24272 volume of IEEE Std 1003.1-2001 calls `mbrlen()`.

24273 The behavior of this function is affected by the `LC_CTYPE` category of the current locale.

24274 **RETURN VALUE**

24275 The `mbrlen()` function shall return the first of the following that applies:

24276 **0** If the next *n* or fewer bytes complete the character that corresponds to the null  
 24277 wide character.

24278 **positive** If the next *n* or fewer bytes complete a valid character; the value returned shall  
 24279 be the number of bytes that complete the character.

24280 **(size\_t)-2** If the next *n* bytes contribute to an incomplete but potentially valid character,  
 24281 and all *n* bytes have been processed. When *n* has at least the value of the  
 24282 {MB\_CUR\_MAX} macro, this case can only occur if *s* points at a sequence of  
 24283 redundant shift sequences (for implementations with state-dependent  
 24284 encodings).

24285 **(size\_t)-1** If an encoding error occurs, in which case the next *n* or fewer bytes do not  
 24286 contribute to a complete and valid character. In this case, [EILSEQ] shall be  
 24287 stored in `errno` and the conversion state is undefined.

24288 **ERRORS**

24289 The `mbrlen()` function may fail if:

24290 [EINVAL] *ps* points to an object that contains an invalid conversion state.

24291 [EILSEQ] Invalid character sequence is detected.

24292 **EXAMPLES**

24293 None.

24294 **APPLICATION USAGE**

24295 None.

24296 **RATIONALE**

24297 None.

24298 **FUTURE DIRECTIONS**

24299 None.

24300 **SEE ALSO**24301 *mbsinit()*, *mbrtowc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>24302 **CHANGE HISTORY**

24303 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995

24304 (E).

24305 **Issue 6**24306 The *mbrlen()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

## 24307 NAME

24308 mbrtowc — convert a character to a wide-character code (restartable)

## 24309 SYNOPSIS

24310 #include <wchar.h>

24311 size\_t mbrtowc(wchar\_t \*restrict pwc, const char \*restrict s,  
24312 size\_t n, mbstate\_t \*restrict ps);

## 24313 DESCRIPTION

24314 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
24315 conflict between the requirements described here and the ISO C standard is unintentional. This  
24316 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24317 If *s* is a null pointer, the *mbrtowc()* function shall be equivalent to the call:

24318 mbrtowc(NULL, "", 1, ps)

24319 In this case, the values of the arguments *pwc* and *n* are ignored.

24320 If *s* is not a null pointer, the *mbrtowc()* function shall inspect at most *n* bytes beginning at the  
24321 byte pointed to by *s* to determine the number of bytes needed to complete the next character  
24322 (including any shift sequences). If the function determines that the next character is completed, it  
24323 shall determine the value of the corresponding wide character and then, if *pwc* is not a null  
24324 pointer, shall store that value in the object pointed to by *pwc*. If the corresponding wide  
24325 character is the null wide character, the resulting state described shall be the initial conversion  
24326 state.

24327 If *ps* is a null pointer, the *mbrtowc()* function shall use its own internal **mbstate\_t** object, which  
24328 shall be initialized at program start-up to the initial conversion state. Otherwise, the **mbstate\_t**  
24329 object pointed to by *ps* shall be used to completely describe the current conversion state of the  
24330 associated character sequence. The implementation shall behave as if no function defined in this  
24331 volume of IEEE Std 1003.1-2001 calls *mbrtowc()*.

24332 The behavior of this function is affected by the *LC\_CTYPE* category of the current locale.

## 24333 RETURN VALUE

24334 The *mbrtowc()* function shall return the first of the following that applies:

24335 0 If the next *n* or fewer bytes complete the character that corresponds to the null  
24336 wide character (which is the value stored).

24337 between 1 and *n* inclusive

24338 If the next *n* or fewer bytes complete a valid character (which is the value  
24339 stored); the value returned shall be the number of bytes that complete the  
24340 character.

24341 (**size\_t**)−2 If the next *n* bytes contribute to an incomplete but potentially valid character,  
24342 and all *n* bytes have been processed (no value is stored). When *n* has at least  
24343 the value of the {*MB\_CUR\_MAX*} macro, this case can only occur if *s* points at  
24344 a sequence of redundant shift sequences (for implementations with state-  
24345 dependent encodings).

24346 (**size\_t**)−1 If an encoding error occurs, in which case the next *n* or fewer bytes do not  
24347 contribute to a complete and valid character (no value is stored). In this case,  
24348 [EILSEQ] shall be stored in *errno* and the conversion state is undefined.

24349 **ERRORS**

24350 The *mbrtowc()* function may fail if:

24351 **CX** [EINVAL] *ps* points to an object that contains an invalid conversion state.

24352 [EILSEQ] Invalid character sequence is detected.

24353 **EXAMPLES**

24354 None.

24355 **APPLICATION USAGE**

24356 None.

24357 **RATIONALE**

24358 None.

24359 **FUTURE DIRECTIONS**

24360 None.

24361 **SEE ALSO**

24362 *mbstowc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <*wchar.h*>

24363 **CHANGE HISTORY**

24364 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

24366 **Issue 6**

24367 The *mbrtowc()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

24368 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 24369 • The [EINVAL] error condition is added.

24371 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

24372 **NAME**

24373 mbsinit — determine conversion object status

24374 **SYNOPSIS**

24375 #include <wchar.h>

24376 int mbsinit(const mbstate\_t \*ps);

24377 **DESCRIPTION**

24378 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
24379 conflict between the requirements described here and the ISO C standard is unintentional. This  
24380 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24381 If *ps* is not a null pointer, the *mbsinit()* function shall determine whether the object pointed to by  
24382 *ps* describes an initial conversion state.

24383 **RETURN VALUE**

24384 The *mbsinit()* function shall return non-zero if *ps* is a null pointer, or if the pointed-to object  
24385 describes an initial conversion state; otherwise, it shall return zero.

24386 If an **mbstate\_t** object is altered by any of the functions described as “restartable”, and is then  
24387 used with a different character sequence, or in the other conversion direction, or with a different  
24388 *LC\_CTYPE* category setting than on earlier function calls, the behavior is undefined.

24389 **ERRORS**

24390 No errors are defined.

24391 **EXAMPLES**

24392 None.

24393 **APPLICATION USAGE**

24394 The **mbstate\_t** object is used to describe the current conversion state from a particular character  
24395 sequence to a wide-character sequence (or *vice versa*) under the rules of a particular setting of the  
24396 *LC\_CTYPE* category of the current locale.

24397 The initial conversion state corresponds, for a conversion in either direction, to the beginning of  
24398 a new character sequence in the initial shift state. A zero valued **mbstate\_t** object is at least one  
24399 way to describe an initial conversion state. A zero valued **mbstate\_t** object can be used to initiate  
24400 conversion involving any character sequence, in any *LC\_CTYPE* category setting.

24401 **RATIONALE**

24402 None.

24403 **FUTURE DIRECTIONS**

24404 None.

24405 **SEE ALSO**

24406 *mbrlen()*, *mbrtowc()*, *wcrtomb()*, *mbsrtowcs()*, *wcsrtombs()*, the Base Definitions volume of  
24407 IEEE Std 1003.1-2001, <**wchar.h**>

24408 **CHANGE HISTORY**

24409 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
24410 (E).

## 24411 NAME

24412 mbsrtowcs — convert a character string to a wide-character string (restartable)

## 24413 SYNOPSIS

24414 #include <wchar.h>

```
24415 size_t mbsrtowcs(wchar_t *restrict dst, const char **restrict src,
24416 size_t len, mbstate_t *restrict ps);
```

## 24417 DESCRIPTION

24418 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 24419 conflict between the requirements described here and the ISO C standard is unintentional. This  
 24420 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24421 The *mbsrtowcs()* function shall convert a sequence of characters, beginning in the conversion  
 24422 state described by the object pointed to by *ps*, from the array indirectly pointed to by *src* into a  
 24423 sequence of corresponding wide characters. If *dst* is not a null pointer, the converted characters  
 24424 shall be stored into the array pointed to by *dst*. Conversion continues up to and including a  
 24425 terminating null character, which shall also be stored. Conversion shall stop early in either of the  
 24426 following cases:

- 24427 • A sequence of bytes is encountered that does not form a valid character.
- 24428 • *len* codes have been stored into the array pointed to by *dst* (and *dst* is not a null pointer).

24429 Each conversion shall take place as if by a call to the *mbrtowc()* function.

24430 If *dst* is not a null pointer, the pointer object pointed to by *src* shall be assigned either a null  
 24431 pointer (if conversion stopped due to reaching a terminating null character) or the address just  
 24432 past the last character converted (if any). If conversion stopped due to reaching a terminating  
 24433 null character, and if *dst* is not a null pointer, the resulting state described shall be the initial  
 24434 conversion state.

24435 If *ps* is a null pointer, the *mbsrtowcs()* function shall use its own internal **mbstate\_t** object, which  
 24436 is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate\_t** object  
 24437 pointed to by *ps* shall be used to completely describe the current conversion state of the  
 24438 associated character sequence. The implementation behaves as if no function defined in this  
 24439 volume of IEEE Std 1003.1-2001 calls *mbsrtowcs()*.

24440 The behavior of this function shall be affected by the *LC\_CTYPE* category of the current locale.

## 24441 RETURN VALUE

24442 If the input conversion encounters a sequence of bytes that do not form a valid character, an  
 24443 encoding error occurs. In this case, the *mbsrtowcs()* function stores the value of the macro  
 24444 [EILSEQ] in *errno* and shall return (**size\_t**)-1; the conversion state is undefined. Otherwise, it  
 24445 shall return the number of characters successfully converted, not including the terminating null  
 24446 (if any).

## 24447 ERRORS

24448 The *mbsrtowcs()* function may fail if:

- 24449 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.
- 24450 [EILSEQ] Invalid character sequence is detected.

24451 **EXAMPLES**

24452           None.

24453 **APPLICATION USAGE**

24454           None.

24455 **RATIONALE**

24456           None.

24457 **FUTURE DIRECTIONS**

24458           None.

24459 **SEE ALSO**24460           *mbsinit()*, *mbrtowc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>24461 **CHANGE HISTORY**24462           First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
24463           (E).24464 **Issue 6**24465           The *mbsrtowcs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

24466           The [EINVAL] error condition is marked CX.

24467 **NAME**

24468 mbstowcs — convert a character string to a wide-character string

24469 **SYNOPSIS**

24470 #include &lt;stdlib.h&gt;

24471 size\_t mbstowcs(wchar\_t \*restrict pwcs, const char \*restrict s,  
24472 size\_t n);24473 **DESCRIPTION**24474 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
24475 conflict between the requirements described here and the ISO C standard is unintentional. This  
24476 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.24477 The *mbstowcs()* function shall convert a sequence of characters that begins in the initial shift  
24478 state from the array pointed to by *s* into a sequence of corresponding wide-character codes and  
24479 shall store not more than *n* wide-character codes into the array pointed to by *pwcs*. No  
24480 characters that follow a null byte (which is converted into a wide-character code with value 0)  
24481 shall be examined or converted. Each character shall be converted as if by a call to *mbtowc()*,  
24482 except that the shift state of *mbtowc()* is not affected.24483 No more than *n* elements shall be modified in the array pointed to by *pwcs*. If copying takes  
24484 place between objects that overlap, the behavior is undefined.24485 XSI The behavior of this function shall be affected by the *LC\_CTYPE* category of the current locale. If  
24486 *pwcs* is a null pointer, *mbstowcs()* shall return the length required to convert the entire array  
24487 regardless of the value of *n*, but no values are stored.24488 **RETURN VALUE**24489 CX If an invalid character is encountered, *mbstowcs()* shall return **(size\_t)-1** and may set *errno* to  
24490 indicate the error.24491 XSI Otherwise, *mbstowcs()* shall return the number of the array elements modified (or required if  
24492 *pwcs* is null), not including a terminating 0 code, if any. The array shall not be zero-terminated if  
24493 the value returned is *n*.24494 **ERRORS**24495 The *mbstowcs()* function may fail if:

24496 XSI [EILSEQ] Invalid byte sequence is detected.

24497 **EXAMPLES**

24498 None.

24499 **APPLICATION USAGE**

24500 None.

24501 **RATIONALE**

24502 None.

24503 **FUTURE DIRECTIONS**

24504 None.

24505 **SEE ALSO**24506 *mblen()*, *mbtowc()*, *wctomb()*, *wcstombs()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
24507 <stdlib.h>

24508 **CHANGE HISTORY**

24509 First released in Issue 4. Aligned with the ISO C standard.

24510 **Issue 6**

24511 The *mbstowcs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

24512 Extensions beyond the ISO C standard are marked.

24513 **NAME**

24514 mbtowc — convert a character to a wide-character code

24515 **SYNOPSIS**

24516 #include &lt;stdlib.h&gt;

24517 int mbtowc(wchar\_t \*restrict *pwc*, const char \*restrict *s*, size\_t *n*);24518 **DESCRIPTION**

24519 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 24520 conflict between the requirements described here and the ISO C standard is unintentional. This  
 24521 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24522 If *s* is not a null pointer, *mbtowc()* shall determine the number of bytes that constitute the  
 24523 character pointed to by *s*. It shall then determine the wide-character code for the value of type  
 24524 **wchar\_t** that corresponds to that character. (The value of the wide-character code corresponding  
 24525 to the null byte is 0.) If the character is valid and *pwc* is not a null pointer, *mbtowc()* shall store  
 24526 the wide-character code in the object pointed to by *pwc*.

24527 The behavior of this function is affected by the *LC\_CTYPE* category of the current locale. For a  
 24528 state-dependent encoding, this function is placed into its initial state by a call for which its  
 24529 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null  
 24530 pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a  
 24531 null pointer shall cause this function to return a non-zero value if encodings have state  
 24532 dependency, and 0 otherwise. If the implementation employs special bytes to change the shift  
 24533 state, these bytes shall not produce separate wide-character codes, but shall be grouped with an  
 24534 adjacent character. Changing the *LC\_CTYPE* category causes the shift state of this function to be  
 24535 unspecified. At most *n* bytes of the array pointed to by *s* shall be examined.

24536 The implementation shall behave as if no function defined in this volume of  
 24537 IEEE Std 1003.1-2001 calls *mbtowc()*.

24538 **RETURN VALUE**

24539 If *s* is a null pointer, *mbtowc()* shall return a non-zero or 0 value, if character encodings,  
 24540 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mbtowc()*  
 24541 shall either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the  
 24542 **CX** converted character (if the next *n* or fewer bytes form a valid character), or return -1 and may  
 24543 set *errno* to indicate the error (if they do not form a valid character).

24544 In no case shall the value returned be greater than *n* or the value of the {MB\_CUR\_MAX} macro.

24545 **ERRORS**

24546 The *mbtowc()* function may fail if:

24547 **XSI** [EILSEQ] Invalid character sequence is detected.

24548 **EXAMPLES**

24549 None.

24550 **APPLICATION USAGE**

24551 None.

24552 **RATIONALE**

24553 None.

24554 **FUTURE DIRECTIONS**

24555 None.

24556 **SEE ALSO**

24557            *mblen()*, *mbstowcs()*, *wctomb()*, *wcstombs()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
24558            <**stdlib.h**>

24559 **CHANGE HISTORY**

24560            First released in Issue 4. Aligned with the ISO C standard.

24561 **Issue 6**

24562            The *mbtowc()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

24563            Extensions beyond the ISO C standard are marked.

24564 **NAME**

24565 memccpy — copy bytes in memory

24566 **SYNOPSIS**

```
24567 xSI #include <string.h>
```

```
24568 void *memccpy(void *restrict s1, const void *restrict s2,
24569 int c, size_t n);
24570
```

24571 **DESCRIPTION**

24572 The *memccpy()* function shall copy bytes from memory area *s2* into *s1*, stopping after the first  
24573 occurrence of byte *c* (converted to an **unsigned char**) is copied, or after *n* bytes are copied,  
24574 whichever comes first. If copying takes place between objects that overlap, the behavior is  
24575 undefined.

24576 **RETURN VALUE**

24577 The *memccpy()* function shall return a pointer to the byte after the copy of *c* in *s1*, or a null  
24578 pointer if *c* was not found in the first *n* bytes of *s2*.

24579 **ERRORS**

24580 No errors are defined.

24581 **EXAMPLES**

24582 None.

24583 **APPLICATION USAGE**

24584 The *memccpy()* function does not check for the overflow of the receiving memory area.

24585 **RATIONALE**

24586 None.

24587 **FUTURE DIRECTIONS**

24588 None.

24589 **SEE ALSO**

24590 The Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

24591 **CHANGE HISTORY**

24592 First released in Issue 1. Derived from Issue 1 of the SVID.

24593 **Issue 6**

24594 The **restrict** keyword is added to the *memccpy()* prototype for alignment with the  
24595 ISO/IEC 9899:1999 standard.

24596 **NAME**

24597 memchr — find byte in memory

24598 **SYNOPSIS**

24599 #include &lt;string.h&gt;

24600 void \*memchr(const void \*s, int c, size\_t n);

24601 **DESCRIPTION**

24602 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
24603 conflict between the requirements described here and the ISO C standard is unintentional. This  
24604 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24605 The *memchr()* function shall locate the first occurrence of *c* (converted to an **unsigned char**) in  
24606 the initial *n* bytes (each interpreted as **unsigned char**) of the object pointed to by *s*.

24607 **RETURN VALUE**

24608 The *memchr()* function shall return a pointer to the located byte, or a null pointer if the byte does  
24609 not occur in the object.

24610 **ERRORS**

24611 No errors are defined.

24612 **EXAMPLES**

24613 None.

24614 **APPLICATION USAGE**

24615 None.

24616 **RATIONALE**

24617 None.

24618 **FUTURE DIRECTIONS**

24619 None.

24620 **SEE ALSO**24621 The Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>24622 **CHANGE HISTORY**

24623 First released in Issue 1. Derived from Issue 1 of the SVID.

24624 **NAME**

24625           memcmp — compare bytes in memory

24626 **SYNOPSIS**

24627           #include <string.h>

24628           int memcmp(const void \*s1, const void \*s2, size\_t n);

24629 **DESCRIPTION**

24630 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
24631 conflict between the requirements described here and the ISO C standard is unintentional. This  
24632 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24633       The *memcmp()* function shall compare the first *n* bytes (each interpreted as **unsigned char**) of the  
24634 object pointed to by *s1* to the first *n* bytes of the object pointed to by *s2*.

24635       The sign of a non-zero return value shall be determined by the sign of the difference between the  
24636 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the objects  
24637 being compared.

24638 **RETURN VALUE**

24639       The *memcmp()* function shall return an integer greater than, equal to, or less than 0, if the object  
24640 pointed to by *s1* is greater than, equal to, or less than the object pointed to by *s2*, respectively.

24641 **ERRORS**

24642       No errors are defined.

24643 **EXAMPLES**

24644       None.

24645 **APPLICATION USAGE**

24646       None.

24647 **RATIONALE**

24648       None.

24649 **FUTURE DIRECTIONS**

24650       None.

24651 **SEE ALSO**

24652       The Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>

24653 **CHANGE HISTORY**

24654       First released in Issue 1. Derived from Issue 1 of the SVID.

24655 **NAME**

24656 memcpy — copy bytes in memory

24657 **SYNOPSIS**

24658 #include &lt;string.h&gt;

24659 void \*memcpy(void \*restrict *s1*, const void \*restrict *s2*, size\_t *n*);24660 **DESCRIPTION**

24661 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
24662 conflict between the requirements described here and the ISO C standard is unintentional. This  
24663 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24664 The *memcpy()* function shall copy *n* bytes from the object pointed to by *s2* into the object pointed  
24665 to by *s1*. If copying takes place between objects that overlap, the behavior is undefined.

24666 **RETURN VALUE**24667 The *memcpy()* function shall return *s1*; no return value is reserved to indicate an error.24668 **ERRORS**

24669 No errors are defined.

24670 **EXAMPLES**

24671 None.

24672 **APPLICATION USAGE**24673 The *memcpy()* function does not check for the overflow of the receiving memory area.24674 **RATIONALE**

24675 None.

24676 **FUTURE DIRECTIONS**

24677 None.

24678 **SEE ALSO**24679 The Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>24680 **CHANGE HISTORY**

24681 First released in Issue 1. Derived from Issue 1 of the SVID.

24682 **Issue 6**24683 The *memcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

24684 **NAME**

24685 memmove — copy bytes in memory with overlapping areas

24686 **SYNOPSIS**

24687 #include &lt;string.h&gt;

24688 void \*memmove(void \*s1, const void \*s2, size\_t n);

24689 **DESCRIPTION**

24690 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
24691 conflict between the requirements described here and the ISO C standard is unintentional. This  
24692 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24693 The *memmove()* function shall copy *n* bytes from the object pointed to by *s2* into the object  
24694 pointed to by *s1*. Copying takes place as if the *n* bytes from the object pointed to by *s2* are first  
24695 copied into a temporary array of *n* bytes that does not overlap the objects pointed to by *s1* and  
24696 *s2*, and then the *n* bytes from the temporary array are copied into the object pointed to by *s1*.

24697 **RETURN VALUE**24698 The *memmove()* function shall return *s1*; no return value is reserved to indicate an error.24699 **ERRORS**

24700 No errors are defined.

24701 **EXAMPLES**

24702 None.

24703 **APPLICATION USAGE**

24704 None.

24705 **RATIONALE**

24706 None.

24707 **FUTURE DIRECTIONS**

24708 None.

24709 **SEE ALSO**24710 The Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>24711 **CHANGE HISTORY**

24712 First released in Issue 4. Derived from the ANSI C standard.

24713 **NAME**

24714           memset — set bytes in memory

24715 **SYNOPSIS**

24716           #include &lt;string.h&gt;

24717           void \*memset(void \*s, int c, size\_t n);

24718 **DESCRIPTION**

24719 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
24720       conflict between the requirements described here and the ISO C standard is unintentional. This  
24721       volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

24722       The *memset()* function shall copy *c* (converted to an **unsigned char**) into each of the first *n* bytes  
24723       of the object pointed to by *s*.

24724 **RETURN VALUE**24725       The *memset()* function shall return *s*; no return value is reserved to indicate an error.24726 **ERRORS**

24727       No errors are defined.

24728 **EXAMPLES**

24729       None.

24730 **APPLICATION USAGE**

24731       None.

24732 **RATIONALE**

24733       None.

24734 **FUTURE DIRECTIONS**

24735       None.

24736 **SEE ALSO**24737       The Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>24738 **CHANGE HISTORY**

24739       First released in Issue 1. Derived from Issue 1 of the SVID.

24740 **NAME**

24741        mkdir — make a directory

24742 **SYNOPSIS**

24743        #include &lt;sys/stat.h&gt;

24744        int mkdir(const char \*path, mode\_t mode);

24745 **DESCRIPTION**

24746        The *mkdir()* function shall create a new directory with name *path*. The file permission bits of the new directory shall be initialized from *mode*. These file permission bits of the *mode* argument shall be modified by the process' file creation mask.

24749        When bits in *mode* other than the file permission bits are set, the meaning of these additional bits is implementation-defined.

24751        The directory's user ID shall be set to the process' effective user ID. The directory's group ID shall be set to the group ID of the parent directory or to the effective group ID of the process. Implementations shall provide a way to initialize the directory's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the directory's group ID to the effective group ID of the calling process.

24756        The newly created directory shall be an empty directory.

24757        If *path* names a symbolic link, *mkdir()* shall fail and set *errno* to [EEXIST].

24758        Upon successful completion, *mkdir()* shall mark for update the *st\_atime*, *st\_ctime*, and *st\_mtime* fields of the directory. Also, the *st\_ctime* and *st\_mtime* fields of the directory that contains the new entry shall be marked for update.

24761 **RETURN VALUE**

24762        Upon successful completion, *mkdir()* shall return 0. Otherwise, -1 shall be returned, no directory shall be created, and *errno* shall be set to indicate the error.

24764 **ERRORS**

24765        The *mkdir()* function shall fail if:

24766        [EACCES]        Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be created.

24768        [EEXIST]        The named file exists.

24769        [ELOOP]        A loop exists in symbolic links encountered during resolution of the *path* argument.

24771        [EMLINK]        The link count of the parent directory would exceed {LINK\_MAX}.

24772        [ENAMETOOLONG]

24773        The length of the *path* argument exceeds {PATH\_MAX} or a pathname component is longer than {NAME\_MAX}.

24775        [ENOENT]        A component of the path prefix specified by *path* does not name an existing directory or *path* is an empty string.

24777        [ENOSPC]        The file system does not contain enough space to hold the contents of the new directory or to extend the parent directory of the new directory.

24779        [ENOTDIR]        A component of the path prefix is not a directory.

24780        [EROFS]        The parent directory resides on a read-only file system.

24781 The *mkdir()* function may fail if:

24782 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
24783 resolution of the *path* argument.

24784 [ENAMETOOLONG]

24785 As a result of encountering a symbolic link in resolution of the *path* argument,  
24786 the length of the substituted pathname string exceeded {PATH\_MAX}.

#### 24787 EXAMPLES

##### 24788 **Creating a Directory**

24789 The following example shows how to create a directory named `/home/cnd/mod1`, with  
24790 read/write/search permissions for owner and group, and with read/search permissions for  
24791 others.

```
24792 #include <sys/types.h>
```

```
24793 #include <sys/stat.h>
```

```
24794 int status;
```

```
24795 ...
```

```
24796 status = mkdir("/home/cnd/mod1", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
```

#### 24797 APPLICATION USAGE

24798 None.

#### 24799 RATIONALE

24800 The *mkdir()* function originated in 4.2 BSD and was added to System V in Release 3.0.

24801 4.3 BSD detects [ENAMETOOLONG].

24802 The POSIX.1-1990 standard required that the group ID of a newly created directory be set to the  
24803 group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2  
24804 required that implementations provide a way to have the group ID be set to the group ID of the  
24805 containing directory, but did not prohibit implementations also supporting a way to set the  
24806 group ID to the effective group ID of the creating process. Conforming applications should not  
24807 assume which group ID will be used. If it matters, an application can use *chown()* to set the  
24808 group ID after the directory is created, or determine under what conditions the implementation  
24809 will set the desired group ID.

#### 24810 FUTURE DIRECTIONS

24811 None.

#### 24812 SEE ALSO

24813 *umask()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<sys/stat.h>`, `<sys/types.h>`

#### 24814 CHANGE HISTORY

24815 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

#### 24816 Issue 6

24817 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

24818 The following new requirements on POSIX implementations derive from alignment with the  
24819 Single UNIX Specification:

- 24820 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
24821 required for conforming implementations of previous POSIX specifications, it was not  
24822 required for UNIX applications.

- 24823      • The [ELOOP] mandatory error condition is added.
- 24824      • A second [ENAMETOOLONG] is added as an optional error condition.
- 24825      The following changes were made to align with the IEEE P1003.1a draft standard:
- 24826      • The [ELOOP] optional error condition is added.

24827 **NAME**

24828 mkfifo — make a FIFO special file

24829 **SYNOPSIS**

24830 #include &lt;sys/stat.h&gt;

24831 int mkfifo(const char \*path, mode\_t mode);

24832 **DESCRIPTION**

24833 The *mkfifo()* function shall create a new FIFO special file named by the pathname pointed to by  
 24834 *path*. The file permission bits of the new FIFO shall be initialized from *mode*. The file permission  
 24835 bits of the *mode* argument shall be modified by the process' file creation mask.

24836 When bits in *mode* other than the file permission bits are set, the effect is implementation-  
 24837 defined.

24838 If *path* names a symbolic link, *mkfifo()* shall fail and set *errno* to [EEXIST].

24839 The FIFO's user ID shall be set to the process' effective user ID. The FIFO's group ID shall be set  
 24840 to the group ID of the parent directory or to the effective group ID of the process.  
 24841 Implementations shall provide a way to initialize the FIFO's group ID to the group ID of the  
 24842 parent directory. Implementations may, but need not, provide an implementation-defined way  
 24843 to initialize the FIFO's group ID to the effective group ID of the calling process.

24844 Upon successful completion, *mkfifo()* shall mark for update the *st\_atime*, *st\_ctime*, and *st\_mtime*  
 24845 fields of the file. Also, the *st\_ctime* and *st\_mtime* fields of the directory that contains the new  
 24846 entry shall be marked for update.

24847 **RETURN VALUE**

24848 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned, no FIFO shall  
 24849 be created, and *errno* shall be set to indicate the error.

24850 **ERRORS**24851 The *mkfifo()* function shall fail if:

24852 [EACCES] A component of the path prefix denies search permission, or write permission  
 24853 is denied on the parent directory of the FIFO to be created.

24854 [EEXIST] The named file already exists.

24855 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 24856 argument.

24857 [ENAMETOOLONG]

24858 The length of the *path* argument exceeds {PATH\_MAX} or a pathname  
 24859 component is longer than {NAME\_MAX}.

24860 [ENOENT] A component of the path prefix specified by *path* does not name an existing  
 24861 directory or *path* is an empty string.

24862 [ENOSPC] The directory that would contain the new file cannot be extended or the file  
 24863 system is out of file-allocation resources.

24864 [ENOTDIR] A component of the path prefix is not a directory.

24865 [EROFS] The named file resides on a read-only file system.

24866 The *mkfifo()* function may fail if:

24867 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 24868 resolution of the *path* argument.

24869 [ENAMETOOLONG]

24870 As a result of encountering a symbolic link in resolution of the *path* argument,  
24871 the length of the substituted pathname string exceeded {PATH\_MAX}.

## 24872 EXAMPLES

### 24873 Creating a FIFO File

24874 The following example shows how to create a FIFO file named `/home/cnd/mod_done`, with  
24875 read/write permissions for owner, and with read permissions for group and others.

```
24876 #include <sys/types.h>
24877 #include <sys/stat.h>
24878
24878 int status;
24879 ...
24880 status = mkfifo("/home/cnd/mod_done", S_IWUSR | S_IRUSR |
24881 S_IRGRP | S_IROTH);
```

## 24882 APPLICATION USAGE

24883 None.

## 24884 RATIONALE

24885 The syntax of this function is intended to maintain compatibility with historical  
24886 implementations of *mknod()*. The latter function was included in the 1984 `/usr/group` standard  
24887 but only for use in creating FIFO special files. The *mknod()* function was originally excluded  
24888 from the POSIX.1-1988 standard as implementation-defined and replaced by *mkdir()* and  
24889 *mkfifo()*. The *mknod()* function is now included for alignment with the Single UNIX  
24890 Specification.

24891 The POSIX.1-1990 standard required that the group ID of a newly created FIFO be set to the  
24892 group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2  
24893 required that implementations provide a way to have the group ID be set to the group ID of the  
24894 containing directory, but did not prohibit implementations also supporting a way to set the  
24895 group ID to the effective group ID of the creating process. Conforming applications should not  
24896 assume which group ID will be used. If it matters, an application can use *chown()* to set the  
24897 group ID after the FIFO is created, or determine under what conditions the implementation will  
24898 set the desired group ID.

## 24899 FUTURE DIRECTIONS

24900 None.

## 24901 SEE ALSO

24902 *umask()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<sys/stat.h>`, `<sys/types.h>`

## 24903 CHANGE HISTORY

24904 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

## 24905 Issue 6

24906 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

24907 The following new requirements on POSIX implementations derive from alignment with the  
24908 Single UNIX Specification:

- 24909 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
24910 required for conforming implementations of previous POSIX specifications, it was not  
24911 required for UNIX applications.

24912 • The [ELOOP] mandatory error condition is added.

24913 • A second [ENAMETOOLONG] is added as an optional error condition.

24914 The following changes were made to align with the IEEE P1003.1a draft standard:

24915 • The [ELOOP] optional error condition is added.

24916 **NAME**24917 `mknod` — make a directory, a special file, or a regular file24918 **SYNOPSIS**24919 XSI `#include <sys/stat.h>`24920 `int mknod(const char *path, mode_t mode, dev_t dev);`

24921

24922 **DESCRIPTION**24923 The `mknod()` function shall create a new file named by the pathname to which the argument `path`  
24924 points.24925 The file type for `path` is OR'ed into the `mode` argument, and the application shall select one of the  
24926 following symbolic constants:

24927

24928

24929

24930

24931

24932

24933

| Name    | Description                      |
|---------|----------------------------------|
| S_IFIFO | FIFO-special                     |
| S_IFCHR | Character-special (non-portable) |
| S_IFDIR | Directory (non-portable)         |
| S_IFBLK | Block-special (non-portable)     |
| S_IFREG | Regular (non-portable)           |

24934 The only portable use of `mknod()` is to create a FIFO-special file. If `mode` is not S\_IFIFO or `dev` is  
24935 not 0, the behavior of `mknod()` is unspecified.24936 The permissions for the new file are OR'ed into the `mode` argument, and may be selected from  
24937 any combination of the following symbolic constants:

24938

24939

24940

24941

24942

24943

24944

24945

24946

24947

24948

24949

24950

24951

24952

24953

24954

| Name    | Description                                 |
|---------|---------------------------------------------|
| S_ISUID | Set user ID on execution.                   |
| S_ISGID | Set group ID on execution.                  |
| S_IRWXU | Read, write, or execute (search) by owner.  |
| S_IRUSR | Read by owner.                              |
| S_IWUSR | Write by owner.                             |
| S_IXUSR | Execute (search) by owner.                  |
| S_IRWXG | Read, write, or execute (search) by group.  |
| S_IRGRP | Read by group.                              |
| S_IWGRP | Write by group.                             |
| S_IXGRP | Execute (search) by group.                  |
| S_IRWXO | Read, write, or execute (search) by others. |
| S_IROTH | Read by others.                             |
| S_IWOTH | Write by others.                            |
| S_IXOTH | Execute (search) by others.                 |
| S_ISVTX | On directories, restricted deletion flag.   |

24955 The user ID of the file shall be initialized to the effective user ID of the process. The group ID of  
24956 the file shall be initialized to either the effective group ID of the process or the group ID of the  
24957 parent directory. Implementations shall provide a way to initialize the file's group ID to the  
24958 group ID of the parent directory. Implementations may, but need not, provide an  
24959 implementation-defined way to initialize the file's group ID to the effective group ID of the  
24960 calling process. The owner, group, and other permission bits of `mode` shall be modified by the file  
24961 mode creation mask of the process. The `mknod()` function shall clear each bit whose  
24962 corresponding bit in the file mode creation mask of the process is set.

- 24963 If *path* names a symbolic link, *mknod()* shall fail and set *errno* to [EEXIST].
- 24964 Upon successful completion, *mknod()* shall mark for update the *st\_atime*, *st\_ctime*, and *st\_mtime*  
24965 fields of the file. Also, the *st\_ctime* and *st\_mtime* fields of the directory that contains the new  
24966 entry shall be marked for update.
- 24967 Only a process with appropriate privileges may invoke *mknod()* for file types other than FIFO-  
24968 special.
- 24969 **RETURN VALUE**
- 24970 Upon successful completion, *mknod()* shall return 0. Otherwise, it shall return -1, the new file  
24971 shall not be created, and *errno* shall be set to indicate the error.
- 24972 **ERRORS**
- 24973 The *mknod()* function shall fail if:
- |       |                |                                                                                   |
|-------|----------------|-----------------------------------------------------------------------------------|
| 24974 | [EACCES]       | A component of the path prefix denies search permission, or write permission      |
| 24975 |                | is denied on the parent directory.                                                |
| 24976 | [EEXIST]       | The named file exists.                                                            |
| 24977 | [EINVAL]       | An invalid argument exists.                                                       |
| 24978 | [EIO]          | An I/O error occurred while accessing the file system.                            |
| 24979 | [ELOOP]        | A loop exists in symbolic links encountered during resolution of the <i>path</i>  |
| 24980 |                | argument.                                                                         |
| 24981 | [ENAMETOOLONG] |                                                                                   |
| 24982 |                | The length of a pathname exceeds {PATH_MAX} or a pathname component is            |
| 24983 |                | longer than {NAME_MAX}.                                                           |
| 24984 | [ENOENT]       | A component of the path prefix specified by <i>path</i> does not name an existing |
| 24985 |                | directory or <i>path</i> is an empty string.                                      |
| 24986 | [ENOSPC]       | The directory that would contain the new file cannot be extended or the file      |
| 24987 |                | system is out of file allocation resources.                                       |
| 24988 | [ENOTDIR]      | A component of the path prefix is not a directory.                                |
| 24989 | [EPERM]        | The invoking process does not have appropriate privileges and the file type is    |
| 24990 |                | not FIFO-special.                                                                 |
| 24991 | [EROFS]        | The directory in which the file is to be created is located on a read-only file   |
| 24992 |                | system.                                                                           |
- 24993 The *mknod()* function may fail if:
- |       |                |                                                                        |
|-------|----------------|------------------------------------------------------------------------|
| 24994 | [ELOOP]        | More than {SYMLOOP_MAX} symbolic links were encountered during         |
| 24995 |                | resolution of the <i>path</i> argument.                                |
| 24996 | [ENAMETOOLONG] |                                                                        |
| 24997 |                | Pathname resolution of a symbolic link produced an intermediate result |
| 24998 |                | whose length exceeds {PATH_MAX}.                                       |

24999 **EXAMPLES**25000 **Creating a FIFO Special File**

25001 The following example shows how to create a FIFO special file named `/home/cnd/mod_done`,  
25002 with read/write permissions for owner, and with read permissions for group and others.

```
25003 #include <sys/types.h>
25004 #include <sys/stat.h>
25005 dev_t dev;
25006 int status;
25007 ...
25008 status = mknod("/home/cnd/mod_done", S_IFIFO | S_IWUSR |
25009 S_IRUSR | S_IRGRP | S_IROTH, dev);
```

25010 **APPLICATION USAGE**

25011 The `mkfifo()` function is preferred over this function for making FIFO special files.

25012 **RATIONALE**

25013 The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group  
25014 ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required  
25015 that implementations provide a way to have the group ID be set to the group ID of the  
25016 containing directory, but did not prohibit implementations also supporting a way to set the  
25017 group ID to the effective group ID of the creating process. Conforming applications should not  
25018 assume which group ID will be used. If it matters, an application can use `chown()` to set the  
25019 group ID after the file is created, or determine under what conditions the implementation will  
25020 set the desired group ID.

25021 **FUTURE DIRECTIONS**

25022 None.

25023 **SEE ALSO**

25024 `chmod()`, `creat()`, `exec`, `mkdir()`, `mkfifo()`, `open()`, `stat()`, `umask()`, the Base Definitions volume of  
25025 IEEE Std 1003.1-2001, `<sys/stat.h>`

25026 **CHANGE HISTORY**

25027 First released in Issue 4, Version 2.

25028 **Issue 5**

25029 Moved from X/OPEN UNIX extension to BASE.

25030 **Issue 6**

25031 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

25032 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
25033 [ELOOP] error condition is added.

25034 **NAME**

25035 mkstemp — make a unique filename

25036 **SYNOPSIS**25037 XSI 

```
#include <stdlib.h>
```

25038 

```
int mkstemp(char *template);
```

25039

25040 **DESCRIPTION**

25041 The *mkstemp()* function shall replace the contents of the string pointed to by *template* by a unique  
25042 filename, and return a file descriptor for the file open for reading and writing. The function thus  
25043 prevents any possible race condition between testing whether the file exists and opening it for  
25044 use. The string in *template* should look like a filename with six trailing 'X's; *mkstemp()* replaces  
25045 each 'X' with a character from the portable filename character set. The characters are chosen  
25046 such that the resulting name does not duplicate the name of an existing file at the time of a call  
25047 to *mkstemp()*.

25048 **RETURN VALUE**

25049 Upon successful completion, *mkstemp()* shall return an open file descriptor. Otherwise, -1 shall  
25050 be returned if no suitable file could be created.

25051 **ERRORS**

25052 No errors are defined.

25053 **EXAMPLES**25054 **Generating a Filename**

25055 The following example creates a file with a 10-character name beginning with the characters  
25056 "file" and opens the file for reading and writing. The value returned as the value of *fd* is a file  
25057 descriptor that identifies the file.

```
25058 #include <stdlib.h>
25059 ...
25060 char template[] = "/tmp/fileXXXXXX";
25061 int fd;

25062 fd = mkstemp(template);
```

25063 **APPLICATION USAGE**

25064 It is possible to run out of letters.

25065 The *mkstemp()* function need not check to determine whether the filename part of *template*  
25066 exceeds the maximum allowable filename length.

25067 **RATIONALE**

25068 None.

25069 **FUTURE DIRECTIONS**

25070 None.

25071 **SEE ALSO**

25072 *getpid()*, *open()*, *tmpfile()*, *tmpnam()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
25073 <stdlib.h>

25074 **CHANGE HISTORY**

25075 First released in Issue 4, Version 2.

25076 **Issue 5**

25077 Moved from X/OPEN UNIX extension to BASE.

25078 **NAME**25079 mktemp — make a unique filename (**LEGACY**)25080 **SYNOPSIS**25081 XSI `#include <stdlib.h>`25082 `char *mktemp(char *template);`

25083

25084 **DESCRIPTION**

25085 The *mktemp()* function shall replace the contents of the string pointed to by *template* by a unique  
 25086 filename and return *template*. The application shall initialize *template* to be a filename with six  
 25087 trailing 'X's; *mktemp()* shall replace each 'X' with a single byte character from the portable  
 25088 filename character set.

25089 **RETURN VALUE**

25090 The *mktemp()* function shall return the pointer *template*. If a unique name cannot be created,  
 25091 *template* shall point to a null string.

25092 **ERRORS**

25093 No errors are defined.

25094 **EXAMPLES**25095 **Generating a Filename**

25096 The following example replaces the contents of the "template" string with a 10-character  
 25097 filename beginning with the characters "file" and returns a pointer to the "template" string  
 25098 that contains the new filename.

25099 `#include <stdlib.h>`25100 `...`25101 `char *template = "/tmp/fileXXXXXX";`25102 `char *ptr;`25103 `ptr = mktemp(template);`25104 **APPLICATION USAGE**

25105 Between the time a pathname is created and the file opened, it is possible for some other process  
 25106 to create a file with the same name. The *mkstemp()* function avoids this problem and is preferred  
 25107 over this function.

25108 **RATIONALE**

25109 None.

25110 **FUTURE DIRECTIONS**

25111 This function may be withdrawn in a future version.

25112 **SEE ALSO**25113 *mkstemp()*, *tmpfile()*, *tmpnam()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<stdlib.h>`25114 **CHANGE HISTORY**

25115 First released in Issue 4, Version 2.

25116 **Issue 5**

25117 Moved from X/OPEN UNIX extension to BASE.

25118 **Issue 6**

25119 This function is marked LEGACY.

25120 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

25121 **NAME**

25122 mktime — convert broken-down time into time since the Epoch

25123 **SYNOPSIS**

25124 #include &lt;time.h&gt;

25125 time\_t mktime(struct tm \*timeptr);

25126 **DESCRIPTION**

25127 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 25128 conflict between the requirements described here and the ISO C standard is unintentional. This  
 25129 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

25130 The *mktime()* function shall convert the broken-down time, expressed as local time, in the  
 25131 structure pointed to by *timeptr*, into a time since the Epoch value with the same encoding as that  
 25132 of the values returned by *time()*. The original values of the *tm\_wday* and *tm\_yday* components of  
 25133 the structure are ignored, and the original values of the other components are not restricted to  
 25134 the ranges described in <time.h>.

25135 cx A positive or 0 value for *tm\_isdst* shall cause *mktime()* to presume initially that Daylight Savings  
 25136 Time, respectively, is or is not in effect for the specified time. A negative value for *tm\_isdst* shall  
 25137 cause *mktime()* to attempt to determine whether Daylight Savings Time is in effect for the  
 25138 specified time.

25139 Local timezone information shall be set as though *mktime()* called *tzset()*.

25140 The relationship between the **tm** structure (defined in the <time.h> header) and the time in  
 25141 seconds since the Epoch is that the result shall be as specified in the expression given in the  
 25142 definition of seconds since the Epoch (see the Base Definitions volume of IEEE Std 1003.1-2001,  
 25143 Section 4.14, Seconds Since the Epoch) corrected for timezone and any seasonal time  
 25144 adjustments, where the names in the structure and in the expression correspond.

25145 Upon successful completion, the values of the *tm\_wday* and *tm\_yday* components of the structure  
 25146 shall be set appropriately, and the other components are set to represent the specified time since  
 25147 the Epoch, but with their values forced to the ranges indicated in the <time.h> entry; the final  
 25148 value of *tm\_mday* shall not be set until *tm\_mon* and *tm\_year* are determined.

25149 **RETURN VALUE**

25150 The *mktime()* function shall return the specified time since the Epoch encoded as a value of type  
 25151 **time\_t**. If the time since the Epoch cannot be represented, the function shall return the value  
 25152 (**time\_t**)-1.

25153 **ERRORS**

25154 No errors are defined.

25155 **EXAMPLES**

25156 What day of the week is July 4, 2001?

25157 #include &lt;stdio.h&gt;

25158 #include &lt;time.h&gt;

25159 struct tm time\_str;

25160 char daybuf[20];

25161 int main(void)

25162 {

25163 time\_str.tm\_year = 2001 - 1900;

25164 time\_str.tm\_mon = 7 - 1;

25165 time\_str.tm\_mday = 4;

```
25166 time_str.tm_hour = 0;
25167 time_str.tm_min = 0;
25168 time_str.tm_sec = 1;
25169 time_str.tm_isdst = -1;
25170 if (mktime(&time_str) == -1)
25171 (void)puts("-unknown-");
25172 else {
25173 (void)strftime(daybuf, sizeof(daybuf), "%A", &time_str);
25174 (void)puts(daybuf);
25175 }
25176 return 0;
25177 }
```

**25178 APPLICATION USAGE**

25179 None.

**25180 RATIONALE**

25181 None.

**25182 FUTURE DIRECTIONS**

25183 None.

**25184 SEE ALSO**

25185 *asctime()*, *clock()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *strftime()*, *strptime()*, *time()*, *utime()*,  
25186 the Base Definitions volume of IEEE Std 1003.1-2001, <**time.h**>

**25187 CHANGE HISTORY**

25188 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ANSI C  
25189 standard.

**25190 Issue 6**

25191 Extensions beyond the ISO C standard are marked.

25192 **NAME**25193 mlock, munlock — lock or unlock a range of process address space (**REALTIME**)25194 **SYNOPSIS**

25195 MLR #include &lt;sys/mman.h&gt;

25196 int mlock(const void \*addr, size\_t len);

25197 int munlock(const void \*addr, size\_t len);

25198

25199 **DESCRIPTION**

25200 The *mlock()* function shall cause those whole pages containing any part of the address space of  
 25201 the process starting at address *addr* and continuing for *len* bytes to be memory-resident until  
 25202 unlocked or until the process exits or *execs* another process image. The implementation may  
 25203 require that *addr* be a multiple of {PAGESIZE}.

25204 The *munlock()* function shall unlock those whole pages containing any part of the address space  
 25205 of the process starting at address *addr* and continuing for *len* bytes, regardless of how many  
 25206 times *mlock()* has been called by the process for any of the pages in the specified range. The  
 25207 implementation may require that *addr* be a multiple of {PAGESIZE}.

25208 If any of the pages in the range specified to a call to *munlock()* are also mapped into the address  
 25209 spaces of other processes, any locks established on those pages by another process are  
 25210 unaffected by the call of this process to *munlock()*. If any of the pages in the range specified by a  
 25211 call to *munlock()* are also mapped into other portions of the address space of the calling process  
 25212 outside the range specified, any locks established on those pages via the other mappings are also  
 25213 unaffected by this call.

25214 Upon successful return from *mlock()*, pages in the specified range shall be locked and memory-  
 25215 resident. Upon successful return from *munlock()*, pages in the specified range shall be unlocked  
 25216 with respect to the address space of the process. Memory residency of unlocked pages is  
 25217 unspecified.

25218 The appropriate privilege is required to lock process memory with *mlock()*.

25219 **RETURN VALUE**

25220 Upon successful completion, the *mlock()* and *munlock()* functions shall return a value of zero.  
 25221 Otherwise, no change is made to any locks in the address space of the process, and the function  
 25222 shall return a value of  $-1$  and set *errno* to indicate the error.

25223 **ERRORS**

25224 The *mlock()* and *munlock()* functions shall fail if:

25225 [ENOMEM] Some or all of the address range specified by the *addr* and *len* arguments does  
 25226 not correspond to valid mapped pages in the address space of the process.

25227 The *mlock()* function shall fail if:

25228 [EAGAIN] Some or all of the memory identified by the operation could not be locked  
 25229 when the call was made.

25230 The *mlock()* and *munlock()* functions may fail if:

25231 [EINVAL] The *addr* argument is not a multiple of {PAGESIZE}.

25232 The *mlock()* function may fail if:

25233 [ENOMEM] Locking the pages mapped by the specified range would exceed an  
 25234 implementation-defined limit on the amount of memory that the process may  
 25235 lock.

25236 [EPERM] The calling process does not have the appropriate privilege to perform the  
25237 requested operation.

25238 **EXAMPLES**

25239 None.

25240 **APPLICATION USAGE**

25241 None.

25242 **RATIONALE**

25243 None.

25244 **FUTURE DIRECTIONS**

25245 None.

25246 **SEE ALSO**

25247 *exec*, *exit()*, *fork()*, *mlockall()*, *munmap()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
25248 `<sys/mman.h>`

25249 **CHANGE HISTORY**

25250 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25251 **Issue 6**

25252 The *mlock()* and *munlock()* functions are marked as part of the Range Memory Locking option.

25253 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
25254 implementation does not support the Range Memory Locking option.

## 25255 NAME

25256 mlockall, munlockall — lock/unlock the address space of a process (**REALTIME**)

## 25257 SYNOPSIS

25258 ML 

```
#include <sys/mman.h>
```

25259 

```
int mlockall(int flags);
```

25260 

```
int munlockall(void);
```

25261

## 25262 DESCRIPTION

25263 The *mlockall()* function shall cause all of the pages mapped by the address space of a process to  
 25264 be memory-resident until unlocked or until the process exits or *execs* another process image. The  
 25265 *flags* argument determines whether the pages to be locked are those currently mapped by the  
 25266 address space of the process, those that are mapped in the future, or both. The *flags* argument is  
 25267 constructed from the bitwise-inclusive OR of one or more of the following symbolic constants,  
 25268 defined in *<sys/mman.h>*:

25269 MCL\_CURRENT Lock all of the pages currently mapped into the address space of the process.

25270 MCL\_FUTURE Lock all of the pages that become mapped into the address space of the  
25271 process in the future, when those mappings are established.

25272 If MCL\_FUTURE is specified, and the automatic locking of future mappings eventually causes  
 25273 the amount of locked memory to exceed the amount of available physical memory or any other  
 25274 implementation-defined limit, the behavior is implementation-defined. The manner in which the  
 25275 implementation informs the application of these situations is also implementation-defined.

25276 The *munlockall()* function shall unlock all currently mapped pages of the address space of the  
 25277 process. Any pages that become mapped into the address space of the process after a call to  
 25278 *munlockall()* shall not be locked, unless there is an intervening call to *mlockall()* specifying  
 25279 MCL\_FUTURE or a subsequent call to *mlockall()* specifying MCL\_CURRENT. If pages mapped  
 25280 into the address space of the process are also mapped into the address spaces of other processes  
 25281 and are locked by those processes, the locks established by the other processes shall be  
 25282 unaffected by a call by this process to *munlockall()*.

25283 Upon successful return from the *mlockall()* function that specifies MCL\_CURRENT, all currently  
 25284 mapped pages of the process' address space shall be memory-resident and locked. Upon return  
 25285 from the *munlockall()* function, all currently mapped pages of the process' address space shall be  
 25286 unlocked with respect to the process' address space. The memory residency of unlocked pages is  
 25287 unspecified.

25288 The appropriate privilege is required to lock process memory with *mlockall()*.

## 25289 RETURN VALUE

25290 Upon successful completion, the *mlockall()* function shall return a value of zero. Otherwise, no  
 25291 additional memory shall be locked, and the function shall return a value of  $-1$  and set *errno* to  
 25292 indicate the error. The effect of failure of *mlockall()* on previously existing locks in the address  
 25293 space is unspecified.

25294 If it is supported by the implementation, the *munlockall()* function shall always return a value of  
 25295 zero. Otherwise, the function shall return a value of  $-1$  and set *errno* to indicate the error.

## 25296 ERRORS

25297 The *mlockall()* function shall fail if:

25298 [EAGAIN] Some or all of the memory identified by the operation could not be locked  
 25299 when the call was made.

- 25300 [EINVAL] The *flags* argument is zero, or includes unimplemented flags.
- 25301 The *mlockall()* function may fail if:
- 25302 [ENOMEM] Locking all of the pages currently mapped into the address space of the  
25303 process would exceed an implementation-defined limit on the amount of  
25304 memory that the process may lock.
- 25305 [EPERM] The calling process does not have the appropriate privilege to perform the  
25306 requested operation.
- 25307 **EXAMPLES**
- 25308 None.
- 25309 **APPLICATION USAGE**
- 25310 None.
- 25311 **RATIONALE**
- 25312 None.
- 25313 **FUTURE DIRECTIONS**
- 25314 None.
- 25315 **SEE ALSO**
- 25316 *exec*, *exit()*, *fork()*, *mlock()*, *munmap()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
25317 <**sys/mman.h**>
- 25318 **CHANGE HISTORY**
- 25319 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
- 25320 **Issue 6**
- 25321 The *mlockall()* and *munlockall()* functions are marked as part of the Process Memory Locking  
25322 option.
- 25323 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
25324 implementation does not support the Process Memory Locking option.

## 25325 NAME

25326 mmap — map pages of memory

## 25327 SYNOPSIS

25328 MC3 #include &lt;sys/mman.h&gt;

```
25329 void *mmap(void *addr, size_t len, int prot, int flags,
25330 int fildes, off_t off);
```

25331

## 25332 DESCRIPTION

25333 The *mmap()* function shall establish a mapping between a process' address space and a file,  
 25334 TYM shared memory object, or typed memory object. The format of the call is as follows:

```
25335 pa=mmap(addr, len, prot, flags, fildes, off);
```

25336 The *mmap()* function shall establish a mapping between the address space of the process at an  
 25337 address *pa* for *len* bytes to the memory object represented by the file descriptor *fildes* at offset *off*  
 25338 for *len* bytes. The value of *pa* is an implementation-defined function of the parameter *addr* and  
 25339 the values of *flags*, further described below. A successful *mmap()* call shall return *pa* as its result.  
 25340 The address range starting at *pa* and continuing for *len* bytes shall be legitimate for the possible  
 25341 (not necessarily current) address space of the process. The range of bytes starting at *off* and  
 25342 continuing for *len* bytes shall be legitimate for the possible (not necessarily current) offsets in the  
 25343 TYM file, shared memory object, or typed memory object represented by *fildes*.

25344 TYM If *fildes* represents a typed memory object opened with either the  
 25345 POSIX\_TYPED\_MEM\_ALLOCATE flag or the POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG  
 25346 flag, the memory object to be mapped shall be that portion of the typed memory object allocated  
 25347 by the implementation as specified below. In this case, if *off* is non-zero, the behavior of *mmap()*  
 25348 is undefined. If *fildes* refers to a valid typed memory object that is not accessible from the calling  
 25349 process, *mmap()* shall fail.

25350 The mapping established by *mmap()* shall replace any previous mappings for those whole pages  
 25351 containing any part of the address space of the process starting at *pa* and continuing for *len*  
 25352 bytes.

25353 If the size of the mapped file changes after the call to *mmap()* as a result of some other operation  
 25354 on the mapped file, the effect of references to portions of the mapped region that correspond to  
 25355 added or removed portions of the file is unspecified.

25356 TYM The *mmap()* function shall be supported for regular files, shared memory objects, and typed  
 25357 memory objects. Support for any other type of file is unspecified.

25358 The parameter *prot* determines whether read, write, execute, or some combination of accesses  
 25359 are permitted to the data being mapped. The *prot* shall be either PROT\_NONE or the bitwise-  
 25360 inclusive OR of one or more of the other flags in the following table, defined in the  
 25361 <sys/mman.h> header.

25362

25363

25364

25365

25366

25367

| Symbolic Constant | Description              |
|-------------------|--------------------------|
| PROT_READ         | Data can be read.        |
| PROT_WRITE        | Data can be written.     |
| PROT_EXEC         | Data can be executed.    |
| PROT_NONE         | Data cannot be accessed. |

25368 If an implementation cannot support the combination of access types specified by *prot*, the call  
 25369 to *mmap()* shall fail.

25370 MPR An implementation may permit accesses other than those specified by *prot*; however, if the  
 25371 Memory Protection option is supported, the implementation shall not permit a write to succeed  
 25372 where PROT\_WRITE has not been set or shall not permit any access where PROT\_NONE alone  
 25373 has been set. The implementation shall support at least the following values of *prot*:  
 25374 PROT\_NONE, PROT\_READ, PROT\_WRITE, and the bitwise-inclusive OR of PROT\_READ and  
 25375 PROT\_WRITE. If the Memory Protection option is not supported, the result of any access that  
 25376 conflicts with the specified protection is undefined. The file descriptor *fildev* shall have been  
 25377 opened with read permission, regardless of the protection options specified. If PROT\_WRITE is  
 25378 specified, the application shall ensure that it has opened the file descriptor *fildev* with write  
 25379 permission unless MAP\_PRIVATE is specified in the *flags* parameter as described below.

25380 The parameter *flags* provides other information about the handling of the mapped data. The  
 25381 value of *flags* is the bitwise-inclusive OR of these options, defined in `<sys/mman.h>`:

25382

25383

25384

25385

25386

| Symbolic Constant | Description                    |
|-------------------|--------------------------------|
| MAP_SHARED        | Changes are shared.            |
| MAP_PRIVATE       | Changes are private.           |
| MAP_FIXED         | Interpret <i>addr</i> exactly. |

25387 Implementations that do not support the Memory Mapped Files option are not required to  
 25388 support MAP\_PRIVATE.

25389 XSI It is implementation-defined whether MAP\_FIXED shall be supported. MAP\_FIXED shall be  
 25390 supported on XSI-conformant systems.

25391 MAP\_SHARED and MAP\_PRIVATE describe the disposition of write references to the memory  
 25392 object. If MAP\_SHARED is specified, write references shall change the underlying object. If  
 25393 MAP\_PRIVATE is specified, modifications to the mapped data by the calling process shall be  
 25394 visible only to the calling process and shall not change the underlying object. It is unspecified  
 25395 whether modifications to the underlying object done after the MAP\_PRIVATE mapping is  
 25396 established are visible through the MAP\_PRIVATE mapping. Either MAP\_SHARED or  
 25397 MAP\_PRIVATE can be specified, but not both. The mapping type is retained across *fork*().

25398 TYM When *fildev* represents a typed memory object opened with either the  
 25399 POSIX\_TYPED\_MEM\_ALLOCATE flag or the POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG  
 25400 flag, *mmap*() shall, if there are enough resources available, map *len* bytes allocated from the  
 25401 corresponding typed memory object which were not previously allocated to any process in any  
 25402 processor that may access that typed memory object. If there are not enough resources available,  
 25403 the function shall fail. If *fildev* represents a typed memory object opened with the  
 25404 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG flag, these allocated bytes shall be contiguous  
 25405 within the typed memory object. If *fildev* represents a typed memory object opened with the  
 25406 POSIX\_TYPED\_MEM\_ALLOCATE flag, these allocated bytes may be composed of non-  
 25407 contiguous fragments within the typed memory object. If *fildev* represents a typed memory  
 25408 object opened with neither the POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG flag nor the  
 25409 POSIX\_TYPED\_MEM\_ALLOCATE flag, *len* bytes starting at offset *off* within the typed memory  
 25410 object are mapped, exactly as when mapping a file or shared memory object. In this case, if two  
 25411 processes map an area of typed memory using the same *off* and *len* values and using file  
 25412 descriptors that refer to the same memory pool (either from the same port or from a different  
 25413 port), both processes shall map the same region of storage.

25414 When MAP\_FIXED is set in the *flags* argument, the implementation is informed that the value of  
 25415 *pa* shall be *addr*, exactly. If MAP\_FIXED is set, *mmap*() may return MAP\_FAILED and set *errno* to  
 25416 [EINVAL]. If a MAP\_FIXED request is successful, the mapping established by *mmap*() replaces  
 25417 any previous mappings for the process' pages in the range [*pa*,*pa+len*).

25418 When MAP\_FIXED is not set, the implementation uses *addr* in an implementation-defined  
 25419 manner to arrive at *pa*. The *pa* so chosen shall be an area of the address space that the  
 25420 implementation deems suitable for a mapping of *len* bytes to the file. All implementations  
 25421 interpret an *addr* value of 0 as granting the implementation complete freedom in selecting *pa*,  
 25422 subject to constraints described below. A non-zero value of *addr* is taken to be a suggestion of a  
 25423 process address near which the mapping should be placed. When the implementation selects a  
 25424 value for *pa*, it never places a mapping at address 0, nor does it replace any extant mapping.

25425 The *off* argument is constrained to be aligned and sized according to the value returned by  
 25426 *sysconf()* when passed *\_SC\_PAGESIZE* or *\_SC\_PAGE\_SIZE*. When MAP\_FIXED is specified, the  
 25427 application shall ensure that the argument *addr* also meets these constraints. The  
 25428 implementation performs mapping operations over whole pages. Thus, while the argument *len*  
 25429 need not meet a size or alignment constraint, the implementation shall include, in any mapping  
 25430 operation, any partial page specified by the range [*pa*,*pa+len*).

25431 The system shall always zero-fill any partial page at the end of an object. Further, the system  
 25432 shall never write out any modified portions of the last page of an object which are beyond its  
 25433 MPR end. References within the address range starting at *pa* and continuing for *len* bytes to whole  
 25434 pages following the end of an object shall result in delivery of a SIGBUS signal.

25435 An implementation may generate SIGBUS signals when a reference would cause an error in the  
 25436 mapped object, such as out-of-space condition.

25437 The *mmap()* function shall add an extra reference to the file associated with the file descriptor  
 25438 *fdes* which is not removed by a subsequent *close()* on that file descriptor. This reference shall be  
 25439 removed when there are no more mappings to the file.

25440 The *st\_atime* field of the mapped file may be marked for update at any time between the *mmap()*  
 25441 call and the corresponding *munmap()* call. The initial read or write reference to a mapped region  
 25442 shall cause the file's *st\_atime* field to be marked for update if it has not already been marked for  
 25443 update.

25444 The *st\_ctime* and *st\_mtime* fields of a file that is mapped with MAP\_SHARED and PROT\_WRITE  
 25445 shall be marked for update at some point in the interval between a write reference to the  
 25446 mapped region and the next call to *msync()* with MS\_ASYNC or MS\_SYNC for that portion of  
 25447 the file by any process. If there is no such call and if the underlying file is modified as a result of  
 25448 a write reference, then these fields shall be marked for update at some time after the write  
 25449 reference.

25450 There may be implementation-defined limits on the number of memory regions that can be  
 25451 mapped (per process or per system).

25452 XSI If such a limit is imposed, whether the number of memory regions that can be mapped by a  
 25453 process is decreased by the use of *shmat()* is implementation-defined.

25454 If *mmap()* fails for reasons other than [EBADF], [EINVAL], or [ENOTSUP], some of the  
 25455 mappings in the address range starting at *addr* and continuing for *len* bytes may have been  
 25456 unmapped.

#### 25457 RETURN VALUE

25458 Upon successful completion, the *mmap()* function shall return the address at which the mapping  
 25459 was placed (*pa*); otherwise, it shall return a value of MAP\_FAILED and set *errno* to indicate the  
 25460 error. The symbol MAP\_FAILED is defined in the <sys/mman.h> header. No successful return  
 25461 from *mmap()* shall return the value MAP\_FAILED.

25462 **ERRORS**

- 25463 The *mmap()* function shall fail if:
- 25464 [EACCES] The *fildev* argument is not open for read, regardless of the protection specified, or *fildev* is not open for write and PROT\_WRITE was specified for a MAP\_SHARED type mapping.
- 25465
- 25466
- 25467 ML [EAGAIN] The mapping could not be locked in memory, if required by *mlockall()*, due to a lack of resources.
- 25468
- 25469 [EBADF] The *fildev* argument is not a valid open file descriptor.
- 25470 [EINVAL] The *addr* argument (if MAP\_FIXED was specified) or *off* is not a multiple of the page size as returned by *sysconf()*, or is considered invalid by the implementation.
- 25471
- 25472
- 25473 [EINVAL] The value of *flags* is invalid (neither MAP\_PRIVATE nor MAP\_SHARED is set).
- 25474
- 25475 [EMFILE] The number of mapped regions would exceed an implementation-defined limit (per process or per system).
- 25476
- 25477 [ENODEV] The *fildev* argument refers to a file whose type is not supported by *mmap()*.
- 25478 [ENOMEM] MAP\_FIXED was specified, and the range [*addr,addr+len*) exceeds that allowed for the address space of a process; or, if MAP\_FIXED was not specified and there is insufficient room in the address space to effect the mapping.
- 25479
- 25480
- 25481 ML [ENOMEM] The mapping could not be locked in memory, if required by *mlockall()*, because it would require more space than the system is able to supply.
- 25482
- 25483 TYM [ENOMEM] Not enough unallocated memory resources remain in the typed memory object designated by *fildev* to allocate *len* bytes.
- 25484
- 25485 [ENOTSUP] MAP\_FIXED or MAP\_PRIVATE was specified in the *flags* argument and the implementation does not support this functionality.
- 25486
- 25487 The implementation does not support the combination of accesses requested in the *prot* argument.
- 25488
- 25489 [ENXIO] Addresses in the range [*off,off+len*) are invalid for the object specified by *fildev*.
- 25490 [ENXIO] MAP\_FIXED was specified in *flags* and the combination of *addr*, *len*, and *off* is invalid for the object specified by *fildev*.
- 25491
- 25492 TYM [ENXIO] The *fildev* argument refers to a typed memory object that is not accessible from the calling process.
- 25493
- 25494 [EOVERFLOW] The file is a regular file and the value of *off* plus *len* exceeds the offset maximum established in the open file description associated with *fildev*.
- 25495

25496 **EXAMPLES**

25497 None.

25498 **APPLICATION USAGE**

- 25499 Use of *mmap()* may reduce the amount of memory available to other memory allocation functions.
- 25500
- 25501 Use of MAP\_FIXED may result in unspecified behavior in further use of *malloc()* and *shmat()*.
- 25502 The use of MAP\_FIXED is discouraged, as it may prevent an implementation from making the most effective use of resources.
- 25503

25504 The application must ensure correct synchronization when using *mmap()* in conjunction with  
25505 any other file access method, such as *read()* and *write()*, standard input/output, and *shmat()*.

25506 The *mmap()* function allows access to resources via address space manipulations, instead of  
25507 *read()/write()*. Once a file is mapped, all a process has to do to access it is use the data at the  
25508 address to which the file was mapped. So, using pseudo-code to illustrate the way in which an  
25509 existing program might be changed to use *mmap()*, the following:

```
25510 fildes = open(...)
25511 lseek(fildes, some_offset)
25512 read(fildes, buf, len)
25513 /* Use data in buf. */
```

25514 becomes:

```
25515 fildes = open(...)
25516 address = mmap(0, len, PROT_READ, MAP_PRIVATE, fildes, some_offset)
25517 /* Use data at address. */
```

#### 25518 RATIONALE

25519 After considering several other alternatives, it was decided to adopt the *mmap()* definition found  
25520 in SVR4 for mapping memory objects into process address spaces. The SVR4 definition is  
25521 minimal, in that it describes only what has been built, and what appears to be necessary for a  
25522 general and portable mapping facility.

25523 Note that while *mmap()* was first designed for mapping files, it is actually a general-purpose  
25524 mapping facility. It can be used to map any appropriate object, such as memory, files, devices,  
25525 and so on, into the address space of a process.

25526 When a mapping is established, it is possible that the implementation may need to map more  
25527 than is requested into the address space of the process because of hardware requirements. An  
25528 application, however, cannot count on this behavior. Implementations that do not use a paged  
25529 architecture may simply allocate a common memory region and return the address of it; such  
25530 implementations probably do not allocate any more than is necessary. References past the end of  
25531 the requested area are unspecified.

25532 If an application requests a mapping that would overlay existing mappings in the process, it  
25533 might be desirable that an implementation detect this and inform the application. However, the  
25534 default, portable (not *MAP\_FIXED*) operation does not overlay existing mappings. On the other  
25535 hand, if the program specifies a fixed address mapping (which requires some implementation  
25536 knowledge to determine a suitable address, if the function is supported at all), then the program  
25537 is presumed to be successfully managing its own address space and should be trusted when it  
25538 asks to map over existing data structures. Furthermore, it is also desirable to make as few system  
25539 calls as possible, and it might be considered onerous to require an *munmap()* before an *mmap()*  
25540 to the same address range. This volume of IEEE Std 1003.1-2001 specifies that the new mappings  
25541 replace any existing mappings, following existing practice in this regard.

25542 It is not expected, when the Memory Protection option is supported, that all hardware  
25543 implementations are able to support all combinations of permissions at all addresses. When this  
25544 option is supported, implementations are required to disallow write access to mappings without  
25545 write permission and to disallow access to mappings without any access permission. Other than  
25546 these restrictions, implementations may allow access types other than those requested by the  
25547 application. For example, if the application requests only *PROT\_WRITE*, the implementation  
25548 may also allow read access. A call to *mmap()* fails if the implementation cannot support allowing  
25549 all the access requested by the application. For example, some implementations cannot support  
25550 a request for both write access and execute access simultaneously. All implementations  
25551 supporting the Memory Protection option must support requests for no access, read access,

25552 write access, and both read and write access. Strictly conforming code must only rely on the  
25553 required checks. These restrictions allow for portability across a wide range of hardware.

25554 The MAP\_FIXED address treatment is likely to fail for non-page-aligned values and for certain  
25555 architecture-dependent address ranges. Conforming implementations cannot count on being  
25556 able to choose address values for MAP\_FIXED without utilizing non-portable, implementation-  
25557 defined knowledge. Nonetheless, MAP\_FIXED is provided as a standard interface conforming to  
25558 existing practice for utilizing such knowledge when it is available.

25559 Similarly, in order to allow implementations that do not support virtual addresses, support for  
25560 directly specifying any mapping addresses via MAP\_FIXED is not required and thus a  
25561 conforming application may not count on it.

25562 The MAP\_PRIVATE function can be implemented efficiently when memory protection hardware  
25563 is available. When such hardware is not available, implementations can implement such  
25564 “mappings” by simply making a real copy of the relevant data into process private memory,  
25565 though this tends to behave similarly to *read()*.

25566 The function has been defined to allow for many different models of using shared memory.  
25567 However, all uses are not equally portable across all machine architectures. In particular, the  
25568 *mmap()* function allows the system as well as the application to specify the address at which to  
25569 map a specific region of a memory object. The most portable way to use the function is always to  
25570 let the system choose the address, specifying NULL as the value for the argument *addr* and not  
25571 to specify MAP\_FIXED.

25572 If it is intended that a particular region of a memory object be mapped at the same address in a  
25573 group of processes (on machines where this is even possible), then MAP\_FIXED can be used to  
25574 pass in the desired mapping address. The system can still be used to choose the desired address  
25575 if the first such mapping is made without specifying MAP\_FIXED, and then the resulting  
25576 mapping address can be passed to subsequent processes for them to pass in via MAP\_FIXED.  
25577 The availability of a specific address range cannot be guaranteed, in general.

25578 The *mmap()* function can be used to map a region of memory that is larger than the current size  
25579 of the object. Memory access within the mapping but beyond the current end of the underlying  
25580 objects may result in SIGBUS signals being sent to the process. The reason for this is that the size  
25581 of the object can be manipulated by other processes and can change at any moment. The  
25582 implementation should tell the application that a memory reference is outside the object where  
25583 this can be detected; otherwise, written data may be lost and read data may not reflect actual  
25584 data in the object.

25585 Note that references beyond the end of the object do not extend the object as the new end cannot  
25586 be determined precisely by most virtual memory hardware. Instead, the size can be directly  
25587 manipulated by *ftruncate()*.

25588 Process memory locking does apply to shared memory regions, and the MEMLOCK\_FUTURE  
25589 argument to *mlockall()* can be relied upon to cause new shared memory regions to be  
25590 automatically locked.

25591 Existing implementations of *mmap()* return the value `-1` when unsuccessful. Since the casting of  
25592 this value to type `void *` cannot be guaranteed by the ISO C standard to be distinct from a  
25593 successful value, this volume of IEEE Std 1003.1-2001 defines the symbol MAP\_FAILED, which a  
25594 conforming implementation does not return as the result of a successful call.

#### 25595 FUTURE DIRECTIONS

25596 None.

25597 **SEE ALSO**

25598 *exec*, *fcntl()*, *fork()*, *lockf()*, *msync()*, *munmap()*, *mprotect()*, *posix\_typed\_mem\_open()*, *shmat()*,  
 25599 *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/mman.h>

25600 **CHANGE HISTORY**

25601 First released in Issue 4, Version 2.

25602 **Issue 5**

25603 Moved from X/OPEN UNIX extension to BASE.

25604 Aligned with *mmap()* in the POSIX Realtime Extension as follows:

- 25605 • The DESCRIPTION is extensively reworded.
- 25606 • The [EAGAIN] and [ENOTSUP] mandatory error conditions are added.
- 25607 • New cases of [ENOMEM] and [ENXIO] are added as mandatory error conditions.
- 25608 • The value returned on failure is the value of the constant MAP\_FAILED; this was previously  
 25609 defined as -1.

25610 Large File Summit extensions are added.

25611 **Issue 6**

25612 The *mmap()* function is marked as part of the Memory Mapped Files option.

25613 The Open Group Corrigendum U028/6 is applied, changing (void \*)-1 to MAP\_FAILED.

25614 The following new requirements on POSIX implementations derive from alignment with the  
 25615 Single UNIX Specification:

- 25616 • The DESCRIPTION is updated to describe the use of MAP\_FIXED.
- 25617 • The DESCRIPTION is updated to describe the addition of an extra reference to the file  
 25618 associated with the file descriptor passed to *mmap()*.
- 25619 • The DESCRIPTION is updated to state that there may be implementation-defined limits on  
 25620 the number of memory regions that can be mapped.
- 25621 • The DESCRIPTION is updated to describe constraints on the alignment and size of the *off*  
 25622 argument.
- 25623 • The [EINVAL] and [EMFILE] error conditions are added.
- 25624 • The [EOVERFLOW] error condition is added. This change is to support large files.

25625 The following changes are made for alignment with the ISO POSIX-1:1996 standard:

- 25626 • The DESCRIPTION is updated to describe the cases when MAP\_PRIVATE and MAP\_FIXED  
 25627 need not be supported.

25628 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 25629 • Semantics for typed memory objects are added to the DESCRIPTION.
- 25630 • New [ENOMEM] and [ENXIO] errors are added to the ERRORS section.
- 25631 • The *posix\_typed\_mem\_open()* function is added to the SEE ALSO section.

25632 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

25633 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/34 is applied, changing the margin code |  
 25634 in the SYNOPSIS from MF | SHM to MC3 (notation for MF | SHM | TYM). |

25635 **NAME**

25636 modf, modff, modfl — decompose a floating-point number

25637 **SYNOPSIS**

25638 #include <math.h>

25639 double modf(double *x*, double \**iptr*);

25640 float modff(float *value*, float \**iptr*);

25641 long double modfl(long double *value*, long double \**iptr*);

25642 **DESCRIPTION**

25643 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 25644 conflict between the requirements described here and the ISO C standard is unintentional. This  
 25645 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

25646 These functions shall break the argument *x* into integral and fractional parts, each of which has  
 25647 the same sign as the argument. It stores the integral part as a **double** (for the *modf()* function), a  
 25648 **float** (for the *modff()* function), or a **long double** (for the *modfl()* function), in the object pointed  
 25649 to by *iptr*.

25650 **RETURN VALUE**

25651 Upon successful completion, these functions shall return the signed fractional part of *x*.

25652 **MX** If *x* is NaN, a NaN shall be returned, and \**iptr* shall be set to a NaN.

25653 If *x* is ±Inf, ±0 shall be returned, and \**iptr* shall be set to ±Inf.

25654 **ERRORS**

25655 No errors are defined.

25656 **EXAMPLES**

25657 None.

25658 **APPLICATION USAGE**

25659 The *modf()* function computes the function result and \**iptr* such that:

25660 a = modf(x, iptr) ;

25661 x == a+\*iptr ;

25662 allowing for the usual floating-point inaccuracies.

25663 **RATIONALE**

25664 None.

25665 **FUTURE DIRECTIONS**

25666 None.

25667 **SEE ALSO**

25668 *frexp()*, *isnan()*, *ldexp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <math.h>

25669 **CHANGE HISTORY**

25670 First released in Issue 1. Derived from Issue 1 of the SVID.

25671 **Issue 5**

25672 The DESCRIPTION is updated to indicate how an application should check for an error. This  
 25673 text was previously published in the APPLICATION USAGE section.

25674 **Issue 6**

25675 The *modff()* and *modfl()* functions are added for alignment with the ISO/IEC 9899:1999  
 25676 standard.

25677 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
25678 revised to align with the ISO/IEC 9899:1999 standard.

25679 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
25680 marked.

25681 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/35 is applied, correcting the code example  
25682 in the APPLICATION USAGE section. |

25683 **NAME**25684 `mprotect` — set protection of memory mapping25685 **SYNOPSIS**25686 MPR `#include <sys/mman.h>`25687 `int mprotect(void *addr, size_t len, int prot);`

25688

25689 **DESCRIPTION**

25690 The `mprotect()` function shall change the access protections to be that specified by `prot` for those  
 25691 whole pages containing any part of the address space of the process starting at address `addr` and  
 25692 continuing for `len` bytes. The parameter `prot` determines whether read, write, execute, or some  
 25693 combination of accesses are permitted to the data being mapped. The `prot` argument should be  
 25694 either `PROT_NONE` or the bitwise-inclusive OR of one or more of `PROT_READ`, `PROT_WRITE`,  
 25695 and `PROT_EXEC`.

25696 If an implementation cannot support the combination of access types specified by `prot`, the call  
 25697 to `mprotect()` shall fail.

25698 An implementation may permit accesses other than those specified by `prot`; however, no  
 25699 implementation shall permit a write to succeed where `PROT_WRITE` has not been set or shall  
 25700 permit any access where `PROT_NONE` alone has been set. Implementations shall support at  
 25701 least the following values of `prot`: `PROT_NONE`, `PROT_READ`, `PROT_WRITE`, and the bitwise-  
 25702 inclusive OR of `PROT_READ` and `PROT_WRITE`. If `PROT_WRITE` is specified, the application  
 25703 shall ensure that it has opened the mapped objects in the specified address range with write  
 25704 permission, unless `MAP_PRIVATE` was specified in the original mapping, regardless of whether  
 25705 the file descriptors used to map the objects have since been closed.

25706 The implementation shall require that `addr` be a multiple of the page size as returned by  
 25707 `sysconf()`.

25708 The behavior of this function is unspecified if the mapping was not established by a call to  
 25709 `mmap()`.

25710 When `mprotect()` fails for reasons other than `[EINVAL]`, the protections on some of the pages in  
 25711 the range `[addr,addr+len)` may have been changed.

25712 **RETURN VALUE**

25713 Upon successful completion, `mprotect()` shall return 0; otherwise, it shall return `-1` and set `errno`  
 25714 to indicate the error.

25715 **ERRORS**

25716 The `mprotect()` function shall fail if:

25717 `[EACCES]` The `prot` argument specifies a protection that violates the access permission  
 25718 the process has to the underlying memory object.

25719 `[EAGAIN]` The `prot` argument specifies `PROT_WRITE` over a `MAP_PRIVATE` mapping  
 25720 and there are insufficient memory resources to reserve for locking the private  
 25721 page.

25722 `[EINVAL]` The `addr` argument is not a multiple of the page size as returned by `sysconf()`.

25723 `[ENOMEM]` Addresses in the range `[addr,addr+len)` are invalid for the address space of a  
 25724 process, or specify one or more pages which are not mapped.

25725 `[ENOMEM]` The `prot` argument specifies `PROT_WRITE` on a `MAP_PRIVATE` mapping, and  
 25726 it would require more space than the system is able to supply for locking the  
 25727 private pages, if required.

25728 [ENOTSUP] The implementation does not support the combination of accesses requested  
 25729 in the *prot* argument.

#### 25730 EXAMPLES

25731 None.

#### 25732 APPLICATION USAGE

25733 The [EINVAL] error above is marked EX because it is defined as an optional error in the POSIX  
 25734 Realtime Extension.

#### 25735 RATIONALE

25736 None.

#### 25737 FUTURE DIRECTIONS

25738 None.

#### 25739 SEE ALSO

25740 *mmap()*, *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/mman.h>

#### 25741 CHANGE HISTORY

25742 First released in Issue 4, Version 2.

#### 25743 Issue 5

25744 Moved from X/OPEN UNIX extension to BASE.

25745 Aligned with *mprotect()* in the POSIX Realtime Extension as follows:

- 25746 • The DESCRIPTION is largely reworded.
- 25747 • [ENOTSUP] and a second form of [ENOMEM] are added as mandatory error conditions.
- 25748 • [EAGAIN] is moved from the optional to the mandatory error conditions.

#### 25749 Issue 6

25750 The *mprotect()* function is marked as part of the Memory Protection option.

25751 The following new requirements on POSIX implementations derive from alignment with the  
 25752 Single UNIX Specification:

- 25753 • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of  
 25754 the page size as returned by *sysconf()*.
- 25755 • The [EINVAL] error condition is added.

25756 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

25757 **NAME**25758 `mq_close` — close a message queue (**REALTIME**)25759 **SYNOPSIS**25760 MSG `#include <mqueue.h>`25761 `int mq_close(mqd_t mqdes);`

25762

25763 **DESCRIPTION**

25764 The `mq_close()` function shall remove the association between the message queue descriptor,  
25765 `mqdes`, and its message queue. The results of using this message queue descriptor after  
25766 successful return from this `mq_close()`, and until the return of this message queue descriptor  
25767 from a subsequent `mq_open()`, are undefined.

25768 If the process has successfully attached a notification request to the message queue via this  
25769 `mqdes`, this attachment shall be removed, and the message queue is available for another process  
25770 to attach for notification.

25771 **RETURN VALUE**

25772 Upon successful completion, the `mq_close()` function shall return a value of zero; otherwise, the  
25773 function shall return a value of `-1` and set `errno` to indicate the error.

25774 **ERRORS**25775 The `mq_close()` function shall fail if:

25776 [EBADF] The `mqdes` argument is not a valid message queue descriptor.

25777 **EXAMPLES**

25778 None.

25779 **APPLICATION USAGE**

25780 None.

25781 **RATIONALE**

25782 None.

25783 **FUTURE DIRECTIONS**

25784 None.

25785 **SEE ALSO**

25786 `mq_open()`, `mq_unlink()`, `msgctl()`, `msgget()`, `msgrcv()`, `msgsnd()`, the Base Definitions volume of  
25787 IEEE Std 1003.1-2001, `<mqueue.h>`

25788 **CHANGE HISTORY**

25789 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25790 **Issue 6**25791 The `mq_close()` function is marked as part of the Message Passing option.

25792 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
25793 implementation does not support the Message Passing option.

25794 **NAME**25795 mq\_getattr — get message queue attributes (**REALTIME**)25796 **SYNOPSIS**

25797 MSG #include &lt;mqueue.h&gt;

25798 int mq\_getattr(mqd\_t mqdes, struct mq\_attr \*mqstat);

25799

25800 **DESCRIPTION**25801 The *mq\_getattr()* function shall obtain status information and attributes of the message queue  
25802 and the open message queue description associated with the message queue descriptor.25803 The *mqdes* argument specifies a message queue descriptor.25804 The results shall be returned in the **mq\_attr** structure referenced by the *mqstat* argument.25805 Upon return, the following members shall have the values associated with the open message  
25806 queue description as set when the message queue was opened and as modified by subsequent  
25807 *mq\_setattr()* calls: *mq\_flags*.25808 The following attributes of the message queue shall be returned as set at message queue  
25809 creation: *mq\_maxmsg*, *mq\_msgsize*.25810 Upon return, the following members within the **mq\_attr** structure referenced by the *mqstat*  
25811 argument shall be set to the current state of the message queue:25812 *mq\_curmsgs* The number of messages currently on the queue.25813 **RETURN VALUE**25814 Upon successful completion, the *mq\_getattr()* function shall return zero. Otherwise, the function  
25815 shall return  $-1$  and set *errno* to indicate the error.25816 **ERRORS**25817 The *mq\_getattr()* function shall fail if:25818 [EBADF] The *mqdes* argument is not a valid message queue descriptor.25819 **EXAMPLES**

25820 None.

25821 **APPLICATION USAGE**

25822 None.

25823 **RATIONALE**

25824 None.

25825 **FUTURE DIRECTIONS**

25826 None.

25827 **SEE ALSO**25828 *mq\_open()*, *mq\_send()*, *mq\_setattr()*, *mq\_timedsend()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the  
25829 Base Definitions volume of IEEE Std 1003.1-2001, <mqueue.h>25830 **CHANGE HISTORY**

25831 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25832 **Issue 6**25833 The *mq\_getattr()* function is marked as part of the Message Passing option.25834 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
25835 implementation does not support the Message Passing option.

25836  
25837

The *mq\_timedsend()* function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

25838 **NAME**25839 mq\_notify — notify process that a message is available (**REALTIME**)25840 **SYNOPSIS**

25841 MSG #include &lt;mqqueue.h&gt;

25842 int mq\_notify(mqd\_t mqdes, const struct sigevent \*notification);

25843

25844 **DESCRIPTION**

25845 If the argument *notification* is not NULL, this function shall register the calling process to be  
 25846 notified of message arrival at an empty message queue associated with the specified message  
 25847 queue descriptor, *mqdes*. The notification specified by the *notification* argument shall be sent to  
 25848 the process when the message queue transitions from empty to non-empty. At any time, only  
 25849 one process may be registered for notification by a message queue. If the calling process or any  
 25850 other process has already registered for notification of message arrival at the specified message  
 25851 queue, subsequent attempts to register for that message queue shall fail.

25852 If *notification* is NULL and the process is currently registered for notification by the specified  
 25853 message queue, the existing registration shall be removed.

25854 When the notification is sent to the registered process, its registration shall be removed. The  
 25855 message queue shall then be available for registration.

25856 If a process has registered for notification of message arrival at a message queue and some  
 25857 thread is blocked in *mq\_receive()* waiting to receive a message when a message arrives at the  
 25858 queue, the arriving message shall satisfy the appropriate *mq\_receive()*. The resulting behavior is  
 25859 as if the message queue remains empty, and no notification shall be sent.

25860 **RETURN VALUE**

25861 Upon successful completion, the *mq\_notify()* function shall return a value of zero; otherwise, the  
 25862 function shall return a value of -1 and set *errno* to indicate the error.

25863 **ERRORS**25864 The *mq\_notify()* function shall fail if:25865 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

25866 [EBUSY] A process is already registered for notification by the message queue.

25867 **EXAMPLES**

25868 None.

25869 **APPLICATION USAGE**

25870 None.

25871 **RATIONALE**

25872 None.

25873 **FUTURE DIRECTIONS**

25874 None.

25875 **SEE ALSO**

25876 *mq\_open()*, *mq\_send()*, *mq\_timedsend()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base  
 25877 Definitions volume of IEEE Std 1003.1-2001, <mqqueue.h>

25878 **CHANGE HISTORY**

25879 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25880 **Issue 6**

25881 The *mq\_notify()* function is marked as part of the Message Passing option.

25882 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
25883 implementation does not support the Message Passing option.

25884 The *mq\_timedsend()* function is added to the SEE ALSO section for alignment with  
25885 IEEE Std 1003.1d-1999.

## 25886 NAME

25887 mq\_open — open a message queue (**REALTIME**)

## 25888 SYNOPSIS

25889 MSG #include &lt;mqueue.h&gt;

25890 mqd\_t mq\_open(const char \*name, int oflag, ...);

25891

## 25892 DESCRIPTION

25893 The *mq\_open()* function shall establish the connection between a process and a message queue  
 25894 with a message queue descriptor. It shall create an open message queue description that refers to  
 25895 the message queue, and a message queue descriptor that refers to that open message queue  
 25896 description. The message queue descriptor is used by other functions to refer to that message  
 25897 queue. The *name* argument points to a string naming a message queue. It is unspecified whether  
 25898 the name appears in the file system and is visible to other functions that take pathnames as  
 25899 arguments. The *name* argument shall conform to the construction rules for a pathname. If *name*  
 25900 begins with the slash character, then processes calling *mq\_open()* with the same value of *name*  
 25901 shall refer to the same message queue object, as long as that name has not been removed. If *name*  
 25902 does not begin with the slash character, the effect is implementation-defined. The interpretation  
 25903 of slash characters other than the leading slash character in *name* is implementation-defined. If  
 25904 the *name* argument is not the name of an existing message queue and creation is not requested,  
 25905 *mq\_open()* shall fail and return an error.

25906 A message queue descriptor may be implemented using a file descriptor, in which case  
 25907 applications can open up to at least {OPEN\_MAX} file and message queues.

25908 The *oflag* argument requests the desired receive and/or send access to the message queue. The  
 25909 requested access permission to receive messages or send messages shall be granted if the calling  
 25910 process would be granted read or write access, respectively, to an equivalently protected file.

25911 The value of *oflag* is the bitwise-inclusive OR of values from the following list. Applications  
 25912 shall specify exactly one of the first three values (access modes) below in the value of *oflag*:

25913 **O\_RDONLY** Open the message queue for receiving messages. The process can use the  
 25914 returned message queue descriptor with *mq\_receive()*, but not *mq\_send()*. A  
 25915 message queue may be open multiple times in the same or different processes  
 25916 for receiving messages.

25917 **O\_WRONLY** Open the queue for sending messages. The process can use the returned  
 25918 message queue descriptor with *mq\_send()* but not *mq\_receive()*. A message  
 25919 queue may be open multiple times in the same or different processes for  
 25920 sending messages.

25921 **O\_RDWR** Open the queue for both receiving and sending messages. The process can use  
 25922 any of the functions allowed for **O\_RDONLY** and **O\_WRONLY**. A message  
 25923 queue may be open multiple times in the same or different processes for  
 25924 sending messages.

25925 Any combination of the remaining flags may be specified in the value of *oflag*:

25926 **O\_CREAT** Create a message queue. It requires two additional arguments: *mode*, which  
 25927 shall be of type **mode\_t**, and *attr*, which shall be a pointer to an **mq\_attr**  
 25928 structure. If the pathname *name* has already been used to create a message  
 25929 queue that still exists, then this flag shall have no effect, except as noted under  
 25930 **O\_EXCL**. Otherwise, a message queue shall be created without any messages  
 25931 in it. The user ID of the message queue shall be set to the effective user ID of  
 25932 the process, and the group ID of the message queue shall be set to the effective

|       |                     |                                                                                               |
|-------|---------------------|-----------------------------------------------------------------------------------------------|
| 25933 |                     | group ID of the process. The file permission bits shall be set to the value of                |
| 25934 |                     | <i>mode</i> . When bits in <i>mode</i> other than file permission bits are set, the effect is |
| 25935 |                     | implementation-defined. If <i>attr</i> is NULL, the message queue shall be created            |
| 25936 |                     | with implementation-defined default message queue attributes. If <i>attr</i> is non-          |
| 25937 |                     | NULL and the calling process has the appropriate privilege on <i>name</i> , the               |
| 25938 |                     | message queue <i>mq_maxmsg</i> and <i>mq_msgsize</i> attributes shall be set to the values    |
| 25939 |                     | of the corresponding members in the <b>mq_attr</b> structure referred to by <i>attr</i> . If  |
| 25940 |                     | <i>attr</i> is non-NULL, but the calling process does not have the appropriate                |
| 25941 |                     | privilege on <i>name</i> , the <i>mq_open()</i> function shall fail and return an error       |
| 25942 |                     | without creating the message queue.                                                           |
| 25943 | O_EXCL              | If O_EXCL and O_CREAT are set, <i>mq_open()</i> shall fail if the message queue               |
| 25944 |                     | <i>name</i> exists. The check for the existence of the message queue and the creation         |
| 25945 |                     | of the message queue if it does not exist shall be atomic with respect to other               |
| 25946 |                     | threads executing <i>mq_open()</i> naming the same <i>name</i> with O_EXCL and                |
| 25947 |                     | O_CREAT set. If O_EXCL is set and O_CREAT is not set, the result is                           |
| 25948 |                     | undefined.                                                                                    |
| 25949 | O_NONBLOCK          | Determines whether an <i>mq_send()</i> or <i>mq_receive()</i> waits for resources or          |
| 25950 |                     | messages that are not currently available, or fails with <i>errno</i> set to [EAGAIN];        |
| 25951 |                     | see <i>mq_send()</i> and <i>mq_receive()</i> for details.                                     |
| 25952 |                     | The <i>mq_open()</i> function does not add or remove messages from the queue.                 |
| 25953 | <b>RETURN VALUE</b> |                                                                                               |
| 25954 |                     | Upon successful completion, the function shall return a message queue descriptor; otherwise,  |
| 25955 |                     | the function shall return ( <b>mqd_t</b> )−1 and set <i>errno</i> to indicate the error.      |
| 25956 | <b>ERRORS</b>       |                                                                                               |
| 25957 |                     | The <i>mq_open()</i> function shall fail if:                                                  |
| 25958 | [EACCES]            | The message queue exists and the permissions specified by <i>oflag</i> are denied, or         |
| 25959 |                     | the message queue does not exist and permission to create the message queue                   |
| 25960 |                     | is denied.                                                                                    |
| 25961 | [EEXIST]            | O_CREAT and O_EXCL are set and the named message queue already exists.                        |
| 25962 | [EINTR]             | The <i>mq_open()</i> function was interrupted by a signal.                                    |
| 25963 | [EINVAL]            | The <i>mq_open()</i> function is not supported for the given name.                            |
| 25964 | [EINVAL]            | O_CREAT was specified in <i>oflag</i> , the value of <i>attr</i> is not NULL, and either      |
| 25965 |                     | <i>mq_maxmsg</i> or <i>mq_msgsize</i> was less than or equal to zero.                         |
| 25966 | [EMFILE]            | Too many message queue descriptors or file descriptors are currently in use by                |
| 25967 |                     | this process.                                                                                 |
| 25968 | [ENAMETOOLONG]      |                                                                                               |
| 25969 |                     | The length of the <i>name</i> argument exceeds {PATH_MAX} or a pathname                       |
| 25970 |                     | component is longer than {NAME_MAX}.                                                          |
| 25971 | [ENFILE]            | Too many message queues are currently open in the system.                                     |
| 25972 | [ENOENT]            | O_CREAT is not set and the named message queue does not exist.                                |
| 25973 | [ENOSPC]            | There is insufficient space for the creation of the new message queue.                        |

25974 **EXAMPLES**

25975 None.

25976 **APPLICATION USAGE**

25977 None.

25978 **RATIONALE**

25979 None.

25980 **FUTURE DIRECTIONS**

25981 None.

25982 **SEE ALSO**

25983 *mq\_close()*, *mq\_getattr()*, *mq\_receive()*, *mq\_send()*, *mq\_setattr()*, *mq\_timedreceive()*, *mq\_timedsend()*,  
25984 *mq\_unlink()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base Definitions volume of  
25985 IEEE Std 1003.1-2001, <mqqueue.h>

25986 **CHANGE HISTORY**

25987 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25988 **Issue 6**25989 The *mq\_open()* function is marked as part of the Message Passing option.

25990 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
25991 implementation does not support the Message Passing option.

25992 The *mq\_timedreceive()* and *mq\_timedsend()* functions are added to the SEE ALSO section for  
25993 alignment with IEEE Std 1003.1d-1999.

25994 The DESCRIPTION of O\_EXCL is updated in response to IEEE PASC Interpretation 1003.1c #48.

## 25995 NAME

25996 mq\_receive, mq\_timedreceive — receive a message from a message queue (**REALTIME**)

## 25997 SYNOPSIS

25998 MSG #include <mqqueue.h>

```
25999 ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len,
26000 unsigned *msg_prio);
```

26001

26002 MSG TMO #include <mqqueue.h>

26003 #include <time.h>

```
26004 ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
26005 size_t msg_len, unsigned *restrict msg_prio,
26006 const struct timespec *restrict abs_timeout);
```

26007

## 26008 DESCRIPTION

26009 The *mq\_receive()* function shall receive the oldest of the highest priority message(s) from the  
 26010 message queue specified by *mqdes*. If the size of the buffer in bytes, specified by the *msg\_len*  
 26011 argument, is less than the *mq\_msgsize* attribute of the message queue, the function shall fail and  
 26012 return an error. Otherwise, the selected message shall be removed from the queue and copied to  
 26013 the buffer pointed to by the *msg\_ptr* argument.

26014 If the value of *msg\_len* is greater than {SSIZE\_MAX}, the result is implementation-defined.

26015 If the argument *msg\_prio* is not NULL, the priority of the selected message shall be stored in the  
 26016 location referenced by *msg\_prio*.

26017 If the specified message queue is empty and O\_NONBLOCK is not set in the message queue  
 26018 description associated with *mqdes*, *mq\_receive()* shall block until a message is enqueued on the  
 26019 message queue or until *mq\_receive()* is interrupted by a signal. If more than one thread is waiting  
 26020 to receive a message when a message arrives at an empty queue and the Priority Scheduling  
 26021 option is supported, then the thread of highest priority that has been waiting the longest shall be  
 26022 selected to receive the message. Otherwise, it is unspecified which waiting thread receives the  
 26023 message. If the specified message queue is empty and O\_NONBLOCK is set in the message  
 26024 queue description associated with *mqdes*, no message shall be removed from the queue, and  
 26025 *mq\_receive()* shall return an error.

26026 TMO The *mq\_timedreceive()* function shall receive the oldest of the highest priority messages from the  
 26027 message queue specified by *mqdes* as described for the *mq\_receive()* function. However, if  
 26028 O\_NONBLOCK was not specified when the message queue was opened via the *mq\_open()*  
 26029 function, and no message exists on the queue to satisfy the receive, the wait for such a message  
 26030 shall be terminated when the specified timeout expires. If O\_NONBLOCK is set, this function is  
 26031 equivalent to *mq\_receive()*.

26032 The timeout expires when the absolute time specified by *abs\_timeout* passes, as measured by the  
 26033 clock on which timeouts are based (that is, when the value of that clock equals or exceeds  
 26034 *abs\_timeout*), or if the absolute time specified by *abs\_timeout* has already been passed at the time  
 26035 of the call.

26036 TMO TMR If the Timers option is supported, the timeout shall be based on the CLOCK\_REALTIME clock; if  
 26037 the Timers option is not supported, the timeout shall be based on the system clock as returned  
 26038 by the *time()* function.

26039 TMO The resolution of the timeout shall be the resolution of the clock on which it is based. The  
 26040 *timespec* argument is defined in the <time.h> header.

26041 Under no circumstance shall the operation fail with a timeout if a message can be removed from  
 26042 the message queue immediately. The validity of the *abs\_timeout* parameter need not be checked  
 26043 if a message can be removed from the message queue immediately.

#### 26044 RETURN VALUE

26045 TMO Upon successful completion, the *mq\_receive()* and *mq\_timedreceive()* functions shall return the  
 26046 length of the selected message in bytes and the message shall be removed from the queue.  
 26047 Otherwise, no message shall be removed from the queue, the functions shall return a value of -1,  
 26048 and set *errno* to indicate the error.

#### 26049 ERRORS

26050 TMO The *mq\_receive()* and *mq\_timedreceive()* functions shall fail if:

26051 [EAGAIN] O\_NONBLOCK was set in the message description associated with *mqdes*,  
 26052 and the specified message queue is empty.

26053 [EBADF] The *mqdes* argument is not a valid message queue descriptor open for reading.

26054 [EMSGSIZE] The specified message buffer size, *msg\_len*, is less than the message size  
 26055 attribute of the message queue.

26056 TMO [EINTR] The *mq\_receive()* or *mq\_timedreceive()* operation was interrupted by a signal.

26057 TMO [EINVAL] The process or thread would have blocked, and the *abs\_timeout* parameter  
 26058 specified a nanoseconds field value less than zero or greater than or equal to  
 26059 1 000 million.

26060 TMO [ETIMEDOUT] The O\_NONBLOCK flag was not set when the message queue was opened,  
 26061 but no message arrived on the queue before the specified timeout expired.

26062 TMO The *mq\_receive()* and *mq\_timedreceive()* functions may fail if:

26063 [EBADMSG] The implementation has detected a data corruption problem with the  
 26064 message.

#### 26065 EXAMPLES

26066 None.

#### 26067 APPLICATION USAGE

26068 None.

#### 26069 RATIONALE

26070 None.

#### 26071 FUTURE DIRECTIONS

26072 None.

#### 26073 SEE ALSO

26074 *mq\_open()*, *mq\_send()*, *mq\_timedsend()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, *time()*, the Base  
 26075 Definitions volume of IEEE Std 1003.1-2001, <**mqqueue.h**>, <**time.h**>

#### 26076 CHANGE HISTORY

26077 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

#### 26078 Issue 6

26079 The *mq\_receive()* function is marked as part of the Message Passing option.

26080 The Open Group Corrigendum U021/4 is applied. The DESCRIPTION is changed to refer to  
 26081 *msg\_len* rather than *maxsize*.

26082 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 26083 implementation does not support the Message Passing option.

26084 The following new requirements on POSIX implementations derive from alignment with the  
26085 Single UNIX Specification:

- 26086 • In this function it is possible for the return value to exceed the range of the type **ssize\_t** (since  
26087 **size\_t** has a larger range of positive values than **ssize\_t**). A sentence restricting the size of  
26088 the **size\_t** object is added to the description to resolve this conflict.

26089 The *mq\_timedreceive()* function is added for alignment with IEEE Std 1003.1d-1999.

26090 The **restrict** keyword is added to the *mq\_timedreceive()* prototype for alignment with the  
26091 ISO/IEC 9899:1999 standard.

26092 IEEE PASC Interpretation 1003.1 #109 is applied, correcting the return type for *mq\_timedreceive()*  
26093 from **int** to **ssize\_t**.

## 26094 NAME

26095 mq\_send, mq\_timedsend — send a message to a message queue (**REALTIME**)

## 26096 SYNOPSIS

26097 MSG #include &lt;mqueue.h&gt;

26098 int mq\_send(mqd\_t mqdes, const char \*msg\_ptr, size\_t msg\_len,  
26099 unsigned msg\_prio);

26100

26101 MSG TMO #include &lt;mqueue.h&gt;

26102 #include &lt;time.h&gt;

26103 int mq\_timedsend(mqd\_t mqdes, const char \*msg\_ptr, size\_t msg\_len,  
26104 unsigned msg\_prio, const struct timespec \*abs\_timeout);

26105

## 26106 DESCRIPTION

26107 The *mq\_send()* function shall add the message pointed to by the argument *msg\_ptr* to the  
26108 message queue specified by *mqdes*. The *msg\_len* argument specifies the length of the message, in  
26109 bytes, pointed to by *msg\_ptr*. The value of *msg\_len* shall be less than or equal to the *mq\_msgsize*  
26110 attribute of the message queue, or *mq\_send()* shall fail.

26111 If the specified message queue is not full, *mq\_send()* shall behave as if the message is inserted  
26112 into the message queue at the position indicated by the *msg\_prio* argument. A message with a  
26113 larger numeric value of *msg\_prio* shall be inserted before messages with lower values of  
26114 *msg\_prio*. A message shall be inserted after other messages in the queue, if any, with equal  
26115 *msg\_prio*. The value of *msg\_prio* shall be less than {MQ\_PRIO\_MAX}.

26116 If the specified message queue is full and O\_NONBLOCK is not set in the message queue  
26117 description associated with *mqdes*, *mq\_send()* shall block until space becomes available to  
26118 enqueue the message, or until *mq\_send()* is interrupted by a signal. If more than one thread is  
26119 waiting to send when space becomes available in the message queue and the Priority Scheduling  
26120 option is supported, then the thread of the highest priority that has been waiting the longest  
26121 shall be unblocked to send its message. Otherwise, it is unspecified which waiting thread is  
26122 unblocked. If the specified message queue is full and O\_NONBLOCK is set in the message  
26123 queue description associated with *mqdes*, the message shall not be queued and *mq\_send()* shall  
26124 return an error.

26125 TMO The *mq\_timedsend()* function shall add a message to the message queue specified by *mqdes* in the  
26126 manner defined for the *mq\_send()* function. However, if the specified message queue is full and  
26127 O\_NONBLOCK is not set in the message queue description associated with *mqdes*, the wait for  
26128 sufficient room in the queue shall be terminated when the specified timeout expires. If  
26129 O\_NONBLOCK is set in the message queue description, this function shall be equivalent to  
26130 *mq\_send()*.

26131 The timeout shall expire when the absolute time specified by *abs\_timeout* passes, as measured by  
26132 the clock on which timeouts are based (that is, when the value of that clock equals or exceeds  
26133 *abs\_timeout*), or if the absolute time specified by *abs\_timeout* has already been passed at the time  
26134 of the call.

26135 TMO TMR If the Timers option is supported, the timeout shall be based on the CLOCK\_REALTIME clock; if  
26136 the Timers option is not supported, the timeout shall be based on the system clock as returned  
26137 by the *time()* function.

26138 TMO The resolution of the timeout shall be the resolution of the clock on which it is based. The  
26139 *timespec* argument is defined in the <time.h> header.

26140 Under no circumstance shall the operation fail with a timeout if there is sufficient room in the  
 26141 queue to add the message immediately. The validity of the *abs\_timeout* parameter need not be  
 26142 checked when there is sufficient room in the queue.

#### 26143 RETURN VALUE

26144 TMO Upon successful completion, the *mq\_send()* and *mq\_timedsend()* functions shall return a value of  
 26145 zero. Otherwise, no message shall be enqueued, the functions shall return  $-1$ , and *errno* shall be  
 26146 set to indicate the error.

#### 26147 ERRORS

26148 TMO The *mq\_send()* and *mq\_timedsend()* functions shall fail if:

26149 [EAGAIN] The O\_NONBLOCK flag is set in the message queue description associated  
 26150 with *mqdes*, and the specified message queue is full.

26151 [EBADF] The *mqdes* argument is not a valid message queue descriptor open for writing.

26152 TMO [EINTR] A signal interrupted the call to *mq\_send()* or *mq\_timedsend()*.

26153 [EINVAL] The value of *msg\_prio* was outside the valid range.

26154 TMO [EINVAL] The process or thread would have blocked, and the *abs\_timeout* parameter  
 26155 specified a nanoseconds field value less than zero or greater than or equal to  
 26156 1 000 million.

26157 [EMSGSIZE] The specified message length, *msg\_len*, exceeds the message size attribute of  
 26158 the message queue.

26159 TMO [ETIMEDOUT] The O\_NONBLOCK flag was not set when the message queue was opened,  
 26160 but the timeout expired before the message could be added to the queue.

#### 26161 EXAMPLES

26162 None.

#### 26163 APPLICATION USAGE

26164 The value of the symbol {MQ\_PRIO\_MAX} limits the number of priority levels supported by the  
 26165 application. Message priorities range from 0 to {MQ\_PRIO\_MAX}-1.

#### 26166 RATIONALE

26167 None.

#### 26168 FUTURE DIRECTIONS

26169 None.

#### 26170 SEE ALSO

26171 *mq\_open()*, *mq\_receive()*, *mq\_setattr()*, *mq\_timedreceive()*, *time()*, the Base Definitions volume of  
 26172 IEEE Std 1003.1-2001, <mqqueue.h>, <time.h>

#### 26173 CHANGE HISTORY

26174 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

#### 26175 Issue 6

26176 The *mq\_send()* function is marked as part of the Message Passing option.

26177 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 26178 implementation does not support the Message Passing option.

26179 The *mq\_timedsend()* function is added for alignment with IEEE Std 1003.1d-1999.

## 26180 NAME

26181 mq\_setattr — set message queue attributes (**REALTIME**)

## 26182 SYNOPSIS

26183 MSG #include <mqqueue.h>

```
26184 int mq_setattr(mqd_t mqdes, const struct mq_attr *restrict mqstat,
26185 struct mq_attr *restrict omqstat);
26186
```

## 26187 DESCRIPTION

26188 The *mq\_setattr()* function shall set attributes associated with the open message queue  
26189 description referenced by the message queue descriptor specified by *mqdes*.

26190 The message queue attributes corresponding to the following members defined in the **mq\_attr**  
26191 structure shall be set to the specified values upon successful completion of *mq\_setattr()*:

26192 *mq\_flags* The value of this member is the bitwise-logical OR of zero or more of  
26193 O\_NONBLOCK and any implementation-defined flags.

26194 The values of the *mq\_maxmsg*, *mq\_msgsize*, and *mq\_curmsgs* members of the **mq\_attr** structure  
26195 shall be ignored by *mq\_setattr()*.

26196 If *omqstat* is non-NULL, the *mq\_setattr()* function shall store, in the location referenced by  
26197 *omqstat*, the previous message queue attributes and the current queue status. These values shall  
26198 be the same as would be returned by a call to *mq\_getattr()* at that point.

## 26199 RETURN VALUE

26200 Upon successful completion, the function shall return a value of zero and the attributes of the  
26201 message queue shall have been changed as specified.

26202 Otherwise, the message queue attributes shall be unchanged, and the function shall return a  
26203 value of -1 and set *errno* to indicate the error.

## 26204 ERRORS

26205 The *mq\_setattr()* function shall fail if:

26206 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

## 26207 EXAMPLES

26208 None.

## 26209 APPLICATION USAGE

26210 None.

## 26211 RATIONALE

26212 None.

## 26213 FUTURE DIRECTIONS

26214 None.

## 26215 SEE ALSO

26216 *mq\_open()*, *mq\_send()*, *mq\_timedsend()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base  
26217 Definitions volume of IEEE Std 1003.1-2001, <**mqqueue.h**>

## 26218 CHANGE HISTORY

26219 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

26220 **Issue 6**

- 26221 The *mq\_setattr()* function is marked as part of the Message Passing option.
- 26222 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
26223 implementation does not support the Message Passing option.
- 26224 The *mq\_timedsend()* function is added to the SEE ALSO section for alignment with  
26225 IEEE Std 1003.1d-1999.
- 26226 The **restrict** keyword is added to the *mq\_setattr()* prototype for alignment with the  
26227 ISO/IEC 9899:1999 standard.

26228 **NAME**

26229 mq\_timedreceive — receive a message from a message queue (**ADVANCED REALTIME**)

26230 **SYNOPSIS**

26231 MSG TMO #include <mqueue.h>

26232 #include <time.h>

26233 ssize\_t mq\_timedreceive(mqd\_t mqdes, char \*restrict msg\_ptr,

26234 size\_t msg\_len, unsigned \*restrict msg\_prio,

26235 const struct timespec \*restrict abs\_timeout);

26236

26237 **DESCRIPTION**

26238 Refer to *mq\_receive()*.

26239 **NAME**26240 mq\_timedsend — send a message to a message queue (**ADVANCED REALTIME**)26241 **SYNOPSIS**

26242 MSG TMO #include &lt;mqueue.h&gt;

26243 #include &lt;time.h&gt;

26244 int mq\_timedsend(mqd\_t mqdes, const char \*msg\_ptr, size\_t msg\_len,  
26245 unsigned msg\_prio, const struct timespec \*abs\_timeout);

26246

26247 **DESCRIPTION**26248 Refer to *mq\_send()*.

26249 **NAME**26250 mq\_unlink — remove a message queue (**REALTIME**)26251 **SYNOPSIS**

26252 MSG #include &lt;mqueue.h&gt;

26253 int mq\_unlink(const char \*name);

26254

26255 **DESCRIPTION**

26256 The *mq\_unlink()* function shall remove the message queue named by the pathname *name*. After  
 26257 a successful call to *mq\_unlink()* with *name*, a call to *mq\_open()* with *name* shall fail if the flag  
 26258 O\_CREAT is not set in *flags*. If one or more processes have the message queue open when  
 26259 *mq\_unlink()* is called, destruction of the message queue shall be postponed until all references to  
 26260 the message queue have been closed.

26261 Calls to *mq\_open()* to recreate the message queue may fail until the message queue is actually  
 26262 removed. However, the *mq\_unlink()* call need not block until all references have been closed; it  
 26263 may return immediately.

26264 **RETURN VALUE**

26265 Upon successful completion, the function shall return a value of zero. Otherwise, the named  
 26266 message queue shall be unchanged by this function call, and the function shall return a value of  
 26267 -1 and set *errno* to indicate the error.

26268 **ERRORS**26269 The *mq\_unlink()* function shall fail if:

26270 [EACCES] Permission is denied to unlink the named message queue.

26271 [ENAMETOOLONG]

26272 The length of the *name* argument exceeds {PATH\_MAX} or a pathname  
 26273 component is longer than {NAME\_MAX}.

26274 [ENOENT] The named message queue does not exist.

26275 **EXAMPLES**

26276 None.

26277 **APPLICATION USAGE**

26278 None.

26279 **RATIONALE**

26280 None.

26281 **FUTURE DIRECTIONS**

26282 None.

26283 **SEE ALSO**

26284 *mq\_close()*, *mq\_open()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base Definitions volume of  
 26285 IEEE Std 1003.1-2001, <mqueue.h>

26286 **CHANGE HISTORY**

26287 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

26288 **Issue 6**26289 The *mq\_unlink()* function is marked as part of the Message Passing option.

26290 The Open Group Corrigendum U021/5 is applied, clarifying that upon unsuccessful completion,  
 26291 the named message queue is unchanged by this function.

26292  
26293

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

26294 **NAME**

26295           mrnd48 — generate uniformly distributed pseudo-random signed long integers

26296 **SYNOPSIS**

26297 xSI       #include <stdlib.h>

26298           long mrnd48(void);

26299

26300 **DESCRIPTION**

26301           Refer to *drand48()*.

26302 **NAME**

26303 msgctl — XSI message control operations

26304 **SYNOPSIS**

26305 XSI #include &lt;sys/msg.h&gt;

26306 int msgctl(int *msqid*, int *cmd*, struct *msqid\_ds* \**buf*);

26307

26308 **DESCRIPTION**

26309 The *msgctl()* function operates on XSI message queues (see the Base Definitions volume of  
 26310 IEEE Std 1003.1-2001, Section 3.224, Message Queue). It is unspecified whether this function  
 26311 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on  
 26312 page 41).

26313 The *msgctl()* function shall provide message control operations as specified by *cmd*. The  
 26314 following values for *cmd*, and the message control operations they specify, are:

26315 **IPC\_STAT** Place the current value of each member of the **msqid\_ds** data structure  
 26316 associated with *msqid* into the structure pointed to by *buf*. The contents of this  
 26317 structure are defined in <sys/msg.h>.

26318 **IPC\_SET** Set the value of the following members of the **msqid\_ds** data structure  
 26319 associated with *msqid* to the corresponding value found in the structure  
 26320 pointed to by *buf*:

26321 msg\_perm.uid  
 26322 msg\_perm.gid  
 26323 msg\_perm.mode  
 26324 msg\_qbytes

26325 **IPC\_SET** can only be executed by a process with appropriate privileges or that  
 26326 has an effective user ID equal to the value of **msg\_perm.cuid** or  
 26327 **msg\_perm.uid** in the **msqid\_ds** data structure associated with *msqid*. Only a  
 26328 process with appropriate privileges can raise the value of **msg\_qbytes**.

26329 **IPC\_RMID** Remove the message queue identifier specified by *msqid* from the system and  
 26330 destroy the message queue and **msqid\_ds** data structure associated with it.  
 26331 **IPC\_RMD** can only be executed by a process with appropriate privileges or  
 26332 one that has an effective user ID equal to the value of **msg\_perm.cuid** or  
 26333 **msg\_perm.uid** in the **msqid\_ds** data structure associated with *msqid*.

26334 **RETURN VALUE**

26335 Upon successful completion, *msgctl()* shall return 0; otherwise, it shall return -1 and set *errno* to  
 26336 indicate the error.

26337 **ERRORS**

26338 The *msgctl()* function shall fail if:

26339 **[EACCES]** The argument *cmd* is **IPC\_STAT** and the calling process does not have read  
 26340 permission; see Section 2.7 (on page 39).

26341 **[EINVAL]** The value of *msqid* is not a valid message queue identifier; or the value of *cmd*  
 26342 is not a valid command.

26343 **[EPERM]** The argument *cmd* is **IPC\_RMID** or **IPC\_SET** and the effective user ID of the  
 26344 calling process is not equal to that of a process with appropriate privileges  
 26345 and it is not equal to the value of **msg\_perm.cuid** or **msg\_perm.uid** in the data  
 26346 structure associated with *msqid*.

26347 [EPERM] The argument *cmd* is IPC\_SET, an attempt is being made to increase to the  
26348 value of **msg\_qbytes**, and the effective user ID of the calling process does not  
26349 have appropriate privileges.

26350 **EXAMPLES**

26351 None.

26352 **APPLICATION USAGE**

26353 The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
26354 (IPC). Application developers who need to use IPC should design their applications so that  
26355 modules using the IPC routines described in Section 2.7 (on page 39) can be easily modified to  
26356 use the alternative interfaces.

26357 **RATIONALE**

26358 None.

26359 **FUTURE DIRECTIONS**

26360 None.

26361 **SEE ALSO**

26362 Section 2.7 (on page 39), Section 2.8 (on page 41), *mq\_close()*, *mq\_getattr()*, *mq\_notify()*,  
26363 *mq\_open()*, *mq\_receive()*, *mq\_send()*, *mq\_setattr()*, *mq\_unlink()*, *msgget()*, *msgrcv()*, *msgsnd()*, the  
26364 Base Definitions volume of IEEE Std 1003.1-2001, <sys/msg.h>

26365 **CHANGE HISTORY**

26366 First released in Issue 2. Derived from Issue 2 of the SVID.

26367 **Issue 5**

26368 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
26369 DIRECTIONS to a new APPLICATION USAGE section.

26370 **NAME**

26371 msgget — get the XSI message queue identifier

26372 **SYNOPSIS**26373 XSI 

```
#include <sys/msg.h>
```

26374 

```
int msgget(key_t key, int msgflg);
```

26375

26376 **DESCRIPTION**

26377 The *msgget()* function operates on XSI message queues (see the Base Definitions volume of  
 26378 IEEE Std 1003.1-2001, Section 3.224, Message Queue). It is unspecified whether this function  
 26379 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on  
 26380 page 41).

26381 The *msgget()* function shall return the message queue identifier associated with the argument  
 26382 *key*.

26383 A message queue identifier, associated message queue, and data structure (see <sys/msg.h>),  
 26384 shall be created for the argument *key* if one of the following is true:

- 26385 • The argument *key* is equal to `IPC_PRIVATE`.
- 26386 • The argument *key* does not already have a message queue identifier associated with it, and  
 26387 (*msgflg* & `IPC_CREAT`) is non-zero.

26388 Upon creation, the data structure associated with the new message queue identifier shall be  
 26389 initialized as follows:

- 26390 • `msg_perm.cuid`, `msg_perm.uid`, `msg_perm.cgid`, and `msg_perm.gid` shall be set equal to the  
 26391 effective user ID and effective group ID, respectively, of the calling process.
- 26392 • The low-order 9 bits of `msg_perm.mode` shall be set equal to the low-order 9 bits of *msgflg*.
- 26393 • `msg_qnum`, `msg_lspid`, `msg_lrpid`, `msg_stime`, and `msg_rtime` shall be set equal to 0.
- 26394 • `msg_ctime` shall be set equal to the current time.
- 26395 • `msg_qbytes` shall be set equal to the system limit.

26396 **RETURN VALUE**

26397 Upon successful completion, *msgget()* shall return a non-negative integer, namely a message  
 26398 queue identifier. Otherwise, it shall return `-1` and set *errno* to indicate the error.

26399 **ERRORS**

26400 The *msgget()* function shall fail if:

- |       |          |                                                                                                                                                                                                    |
|-------|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 26401 | [EACCES] | A message queue identifier exists for the argument <i>key</i> , but operation permission as specified by the low-order 9 bits of <i>msgflg</i> would not be granted; see Section 2.7 (on page 39). |
| 26402 |          |                                                                                                                                                                                                    |
| 26403 |          |                                                                                                                                                                                                    |
| 26404 | [EEXIST] | A message queue identifier exists for the argument <i>key</i> but $((msgflg \& IPC\_CREAT) \&\& (msgflg \& IPC\_EXCL))$ is non-zero.                                                               |
| 26405 |          |                                                                                                                                                                                                    |
| 26406 | [ENOENT] | A message queue identifier does not exist for the argument <i>key</i> and $(msgflg \& IPC\_CREAT)$ is 0.                                                                                           |
| 26407 |          |                                                                                                                                                                                                    |
| 26408 | [ENOSPC] | A message queue identifier is to be created but the system-imposed limit on the maximum number of allowed message queue identifiers system-wide would be exceeded.                                 |
| 26409 |          |                                                                                                                                                                                                    |
| 26410 |          |                                                                                                                                                                                                    |

26411 **EXAMPLES**

26412 None.

26413 **APPLICATION USAGE**

26414 The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
26415 (IPC). Application developers who need to use IPC should design their applications so that  
26416 modules using the IPC routines described in Section 2.7 (on page 39) can be easily modified to  
26417 use the alternative interfaces.

26418 **RATIONALE**

26419 None.

26420 **FUTURE DIRECTIONS**

26421 None.

26422 **SEE ALSO**

26423 Section 2.7 (on page 39), Section 2.8 (on page 41), *mq\_close()*, *mq\_getattr()*, *mq\_notify()*,  
26424 *mq\_open()*, *mq\_receive()*, *mq\_send()*, *mq\_setattr()*, *mq\_unlink()*, *msgctl()*, *msgrcv()*, *msgsnd()*, the  
26425 Base Definitions volume of IEEE Std 1003.1-2001, <sys/msg.h>

26426 **CHANGE HISTORY**

26427 First released in Issue 2. Derived from Issue 2 of the SVID.

26428 **Issue 5**

26429 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
26430 DIRECTIONS to a new APPLICATION USAGE section.

## 26431 NAME

26432 msgrcv — XSI message receive operation

## 26433 SYNOPSIS

26434 XSI 

```
#include <sys/msg.h>
```

26435 

```
ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,
```

  
26436 

```
int msgflg);
```

26437

## 26438 DESCRIPTION

26439 The *msgrcv()* function operates on XSI message queues (see the Base Definitions volume of  
26440 IEEE Std 1003.1-2001, Section 3.224, Message Queue). It is unspecified whether this function  
26441 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on  
26442 page 41).26443 The *msgrcv()* function shall read a message from the queue associated with the message queue  
26444 identifier specified by *msqid* and place it in the user-defined buffer pointed to by *msgp*.26445 The application shall ensure that the argument *msgp* points to a user-defined buffer that contains  
26446 first a field of type **long** specifying the type of the message, and then a data portion that holds  
26447 the data bytes of the message. The structure below is an example of what this user-defined  
26448 buffer might look like:26449 

```
struct mymsg {
```

  
26450 

```
 long mtype; /* Message type. */
```

  
26451 

```
 char mtext[1]; /* Message text. */
```

  
26452 

```
}
```

26453 The structure member *mtype* is the received message's type as specified by the sending process.26454 The structure member *mtext* is the text of the message.26455 The argument *msgsz* specifies the size in bytes of *mtext*. The received message shall be truncated  
26456 to *msgsz* bytes if it is larger than *msgsz* and (*msgflg* & MSG\_NOERROR) is non-zero. The  
26457 truncated part of the message shall be lost and no indication of the truncation shall be given to  
26458 the calling process.26459 If the value of *msgsz* is greater than {SSIZE\_MAX}, the result is implementation-defined.26460 The argument *msgtyp* specifies the type of message requested as follows:

- 26461
- If *msgtyp* is 0, the first message on the queue shall be received.
  - If *msgtyp* is greater than 0, the first message of type *msgtyp* shall be received.
  - If *msgtyp* is less than 0, the first message of the lowest type that is less than or equal to the  
26464 absolute value of *msgtyp* shall be received.

26465 The argument *msgflg* specifies the action to be taken if a message of the desired type is not on the  
26466 queue. These are as follows:

- 26467
- If (*msgflg* & IPC\_NOWAIT) is non-zero, the calling thread shall return immediately with a  
26468 return value of -1 and *errno* set to [ENOMSG].
  - If (*msgflg* & IPC\_NOWAIT) is 0, the calling thread shall suspend execution until one of the  
26470 following occurs:
    - A message of the desired type is placed on the queue.
    - The message queue identifier *msqid* is removed from the system; when this occurs, *errno*  
26473 shall be set equal to [EIDRM] and -1 shall be returned.

26474 — The calling thread receives a signal that is to be caught; in this case a message is not  
 26475 received and the calling thread resumes execution in the manner prescribed in *sigaction()*.

26476 Upon successful completion, the following actions are taken with respect to the data structure  
 26477 associated with *msqid*:

- 26478 • **msg\_qnum** shall be decremented by 1.
- 26479 • **msg\_lrpid** shall be set equal to the process ID of the calling process.
- 26480 • **msg\_rtime** shall be set equal to the current time.

#### 26481 RETURN VALUE

26482 Upon successful completion, *msgrcv()* shall return a value equal to the number of bytes actually  
 26483 placed into the buffer *mtext*. Otherwise, no message shall be received, *msgrcv()* shall return  
 26484 (**ssize\_t**)−1, and *errno* shall be set to indicate the error.

#### 26485 ERRORS

26486 The *msgrcv()* function shall fail if:

- |       |          |                                                                                                         |
|-------|----------|---------------------------------------------------------------------------------------------------------|
| 26487 | [E2BIG]  | The value of <i>mtext</i> is greater than <i>msgsz</i> and ( <i>msgflg</i> & MSG_NOERROR) is 0.         |
| 26488 | [EACCES] | Operation permission is denied to the calling process; see Section 2.7 (on page 39).                    |
| 26489 |          |                                                                                                         |
| 26490 | [EIDRM]  | The message queue identifier <i>msqid</i> is removed from the system.                                   |
| 26491 | [EINTR]  | The <i>msgrcv()</i> function was interrupted by a signal.                                               |
| 26492 | [EINVAL] | <i>msqid</i> is not a valid message queue identifier.                                                   |
| 26493 | [ENOMSG] | The queue does not contain a message of the desired type and ( <i>msgflg</i> & IPC_NOWAIT) is non-zero. |
| 26494 |          |                                                                                                         |

#### 26495 EXAMPLES

##### 26496 Receiving a Message

26497 The following example receives the first message on the queue (based on the value of the *msgtyp*  
 26498 argument, 0). The queue is identified by the *msqid* argument (assuming that the value has  
 26499 previously been set). This call specifies that an error should be reported if no message is  
 26500 available, but not if the message is too large. The message size is calculated directly using the  
 26501 *sizeof* operator.

```

26502 #include <sys/msg.h>
26503 ...
26504 int result;
26505 int msqid;
26506 struct message {
26507 long type;
26508 char text[20];
26509 } msg;
26510 long msgtyp = 0;
26511 ...
26512 result = msgrcv(msqid, (void *) &msg, sizeof(msg.text),
26513 msgtyp, MSG_NOERROR | IPC_NOWAIT);

```

**26514 APPLICATION USAGE**

26515 The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
26516 (IPC). Application developers who need to use IPC should design their applications so that  
26517 modules using the IPC routines described in Section 2.7 (on page 39) can be easily modified to  
26518 use the alternative interfaces.

**26519 RATIONALE**

26520 None.

**26521 FUTURE DIRECTIONS**

26522 None.

**26523 SEE ALSO**

26524 Section 2.7 (on page 39), Section 2.8 (on page 41), *mq\_close()*, *mq\_getattr()*, *mq\_notify()*,  
26525 *mq\_open()*, *mq\_receive()*, *mq\_send()*, *mq\_setattr()*, *mq\_unlink()*, *msgctl()*, *msgget()*, *msgsnd()*,  
26526 *sigaction()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/msg.h>

**26527 CHANGE HISTORY**

26528 First released in Issue 2. Derived from Issue 2 of the SVID.

**26529 Issue 5**

26530 The type of the return value is changed from **int** to **ssize\_t**, and a warning is added to the  
26531 DESCRIPTION about values of *msgsz* larger than `{SSIZE_MAX}`.

26532 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
26533 DIRECTIONS to the APPLICATION USAGE section.

**26534 Issue 6**

26535 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

## 26536 NAME

26537 msgsnd — XSI message send operation

## 26538 SYNOPSIS

26539 XSI 

```
#include <sys/msg.h>
```

26540 

```
int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

26541

## 26542 DESCRIPTION

26543 The *msgsnd()* function operates on XSI message queues (see the Base Definitions volume of  
 26544 IEEE Std 1003.1-2001, Section 3.224, Message Queue). It is unspecified whether this function  
 26545 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on  
 26546 page 41).

26547 The *msgsnd()* function shall send a message to the queue associated with the message queue  
 26548 identifier specified by *msqid*.

26549 The application shall ensure that the argument *msgp* points to a user-defined buffer that contains  
 26550 first a field of type **long** specifying the type of the message, and then a data portion that holds  
 26551 the data bytes of the message. The structure below is an example of what this user-defined  
 26552 buffer might look like:

```
26553 struct mymsg {
26554 long mtype; /* Message type. */
26555 char mtext[1]; /* Message text. */
26556 }
```

26557 The structure member *mtype* is a non-zero positive type **long** that can be used by the receiving  
 26558 process for message selection.

26559 The structure member *mtext* is any text of length *msgsz* bytes. The argument *msgsz* can range  
 26560 from 0 to a system-imposed maximum.

26561 The argument *msgflg* specifies the action to be taken if one or more of the following is true:

- 26562 • The number of bytes already on the queue is equal to **msg\_qbytes**; see `<sys/msg.h>`.
- 26563 • The total number of messages on all queues system-wide is equal to the system-imposed  
 26564 limit.

26565 These actions are as follows:

- 26566 • If (*msgflg* & IPC\_NOWAIT) is non-zero, the message shall not be sent and the calling thread  
 26567 shall return immediately.
- 26568 • If (*msgflg* & IPC\_NOWAIT) is 0, the calling thread shall suspend execution until one of the  
 26569 following occurs:
  - 26570 — The condition responsible for the suspension no longer exists, in which case the message  
 26571 is sent.
  - 26572 — The message queue identifier *msqid* is removed from the system; when this occurs, *errno*  
 26573 shall be set equal to [EIDRM] and `-1` shall be returned.
  - 26574 — The calling thread receives a signal that is to be caught; in this case the message is not  
 26575 sent and the calling thread resumes execution in the manner prescribed in *sigaction()*.

26576 Upon successful completion, the following actions are taken with respect to the data structure  
 26577 associated with *msqid*; see `<sys/msg.h>`:

- 26578           • **msg\_qnum** shall be incremented by 1.
- 26579           • **msg\_lspid** shall be set equal to the process ID of the calling process.
- 26580           • **msg\_stime** shall be set equal to the current time.

**26581 RETURN VALUE**

26582           Upon successful completion, *msgsnd()* shall return 0; otherwise, no message shall be sent,  
26583           *msgsnd()* shall return -1, and *errno* shall be set to indicate the error.

**26584 ERRORS**

26585           The *msgsnd()* function shall fail if:

- |                         |          |                                                                                                                                                                                                       |
|-------------------------|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 26586<br>26587          | [EACCES] | Operation permission is denied to the calling process; see Section 2.7 (on page 39).                                                                                                                  |
| 26588<br>26589          | [EAGAIN] | The message cannot be sent for one of the reasons cited above and ( <i>msgflg</i> & <i>IPC_NOWAIT</i> ) is non-zero.                                                                                  |
| 26590                   | [EIDRM]  | The message queue identifier <i>msqid</i> is removed from the system.                                                                                                                                 |
| 26591                   | [EINTR]  | The <i>msgsnd()</i> function was interrupted by a signal.                                                                                                                                             |
| 26592<br>26593<br>26594 | [EINVAL] | The value of <i>msqid</i> is not a valid message queue identifier, or the value of <i>mtype</i> is less than 1; or the value of <i>msgsz</i> is less than 0 or greater than the system-imposed limit. |

**26595 EXAMPLES****26596 Sending a Message**

26597           The following example sends a message to the queue identified by the *msqid* argument  
26598           (assuming that value has previously been set). This call specifies that an error should be  
26599           reported if no message is available. The message size is calculated directly using the *sizeof*  
26600           operator.

```
26601 #include <sys/msg.h>
26602 ...
26603 int result;
26604 int msqid;
26605 struct message {
26606 long type;
26607 char text[20];
26608 } msg;

26609 msg.type = 1;
26610 strcpy(msg.text, "This is message 1");
26611 ...
26612 result = msgsnd(msqid, (void *) &msg, sizeof(msg.text), IPC_NOWAIT);
```

**26613 APPLICATION USAGE**

26614           The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
26615           (IPC). Application developers who need to use IPC should design their applications so that  
26616           modules using the IPC routines described in Section 2.7 (on page 39) can be easily modified to  
26617           use the alternative interfaces.

26618 **RATIONALE**

26619 None.

26620 **FUTURE DIRECTIONS**

26621 None.

26622 **SEE ALSO**

26623 Section 2.7 (on page 39), Section 2.8 (on page 41), *mq\_close()*, *mq\_getattr()*, *mq\_notify()*,  
26624 *mq\_open()*, *mq\_receive()*, *mq\_send()*, *mq\_setattr()*, *mq\_unlink()*, *msgctl()*, *msgget()*, *msgrcv()*,  
26625 *sigaction()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/msg.h>

26626 **CHANGE HISTORY**

26627 First released in Issue 2. Derived from Issue 2 of the SVID.

26628 **Issue 5**

26629 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
26630 DIRECTIONS to a new APPLICATION USAGE section.

26631 **Issue 6**

26632 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

26633 **NAME**26634 `msync` — synchronize memory with physical storage26635 **SYNOPSIS**26636 MF SIO `#include <sys/mman.h>`26637 `int msync(void *addr, size_t len, int flags);`

26638

26639 **DESCRIPTION**

26640 The `msync()` function shall write all modified data to permanent storage locations, if any, in  
 26641 those whole pages containing any part of the address space of the process starting at address  
 26642 `addr` and continuing for `len` bytes. If no such storage exists, `msync()` need not have any effect. If  
 26643 requested, the `msync()` function shall then invalidate cached copies of data.

26644 The implementation shall require that `addr` be a multiple of the page size as returned by  
 26645 `sysconf()`.

26646 For mappings to files, the `msync()` function shall ensure that all write operations are completed  
 26647 as defined for synchronized I/O data integrity completion. It is unspecified whether the  
 26648 implementation also writes out other file attributes. When the `msync()` function is called on  
 26649 `MAP_PRIVATE` mappings, any modified data shall not be written to the underlying object and  
 26650 shall not cause such data to be made visible to other processes. It is unspecified whether data in  
 26651 `MAP_PRIVATE` mappings has any permanent storage locations. The effect of `msync()` on a  
 26652 shared memory object or a typed memory object is unspecified. The behavior of this function is  
 26653 unspecified if the mapping was not established by a call to `mmap()`.

26654 The `flags` argument is constructed from the bitwise-inclusive OR of one or more of the following  
 26655 flags defined in the `<sys/mman.h>` header:

26656

26657

26658

26659

26660

| Symbolic Constant          | Description                  |
|----------------------------|------------------------------|
| <code>MS_ASYNC</code>      | Perform asynchronous writes. |
| <code>MS_SYNC</code>       | Perform synchronous writes.  |
| <code>MS_INVALIDATE</code> | Invalidate cached data.      |

26661 When `MS_ASYNC` is specified, `msync()` shall return immediately once all the write operations  
 26662 are initiated or queued for servicing; when `MS_SYNC` is specified, `msync()` shall not return until  
 26663 all write operations are completed as defined for synchronized I/O data integrity completion.  
 26664 Either `MS_ASYNC` or `MS_SYNC` is specified, but not both.

26665 When `MS_INVALIDATE` is specified, `msync()` shall invalidate all cached copies of mapped data  
 26666 that are inconsistent with the permanent storage locations such that subsequent references shall  
 26667 obtain data that was consistent with the permanent storage locations sometime between the call  
 26668 to `msync()` and the first subsequent memory reference to the data.

26669 If `msync()` causes any write to a file, the file's `st_ctime` and `st_mtime` fields shall be marked for  
 26670 update.

26671 **RETURN VALUE**

26672 Upon successful completion, `msync()` shall return 0; otherwise, it shall return `-1` and set `errno` to  
 26673 indicate the error.

26674 **ERRORS**

26675 The `msync()` function shall fail if:

26676 `[EBUSY]` Some or all of the addresses in the range starting at `addr` and continuing for `len`  
 26677 bytes are locked, and `MS_INVALIDATE` is specified.

- 26678 [EINVAL] The value of *flags* is invalid.
- 26679 [EINVAL] The value of *addr* is not a multiple of the page size {PAGESIZE}.
- 26680 [ENOMEM] The addresses in the range starting at *addr* and continuing for *len* bytes are outside the range allowed for the address space of a process or specify one or more pages that are not mapped.
- 26681
- 26682

26683 **EXAMPLES**

26684 None.

26685 **APPLICATION USAGE**

26686 The *msync()* function is only supported if the Memory Mapped Files option and the Synchronized Input and Output option are supported, and thus need not be available on all implementations.

26687

26688

26689 The *msync()* function should be used by programs that require a memory object to be in a known state; for example, in building transaction facilities.

26690

26691 Normal system activity can cause pages to be written to disk. Therefore, there are no guarantees that *msync()* is the only control over when pages are or are not written to disk.

26692

26693 **RATIONALE**

26694 The *msync()* function writes out data in a mapped region to the permanent storage for the underlying object. The call to *msync()* ensures data integrity of the file.

26695

26696 After the data is written out, any cached data may be invalidated if the MS\_INVALIDATE flag was specified. This is useful on systems that do not support read/write consistency.

26697

26698 **FUTURE DIRECTIONS**

26699 None.

26700 **SEE ALSO**26701 *mmap()*, *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/mman.h>26702 **CHANGE HISTORY**

26703 First released in Issue 4, Version 2.

26704 **Issue 5**

26705 Moved from X/OPEN UNIX extension to BASE.

26706 Aligned with *msync()* in the POSIX Realtime Extension as follows:

- 26707
- The DESCRIPTION is extensively reworded.
  - [EBUSY] and a new form of [EINVAL] are added as mandatory error conditions.
- 26708

26709 **Issue 6**

26710 The *msync()* function is marked as part of the Memory Mapped Files and Synchronized Input and Output options.

26711

26712 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 26713
- The [EBUSY] mandatory error condition is added.

26714 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

26715

- 26716
- The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of the page size.
  - The second [EINVAL] error condition is made mandatory.
- 26717
- 26718

26719  
26720

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding reference to typed memory objects.

26721 **NAME**

26722       munlock — unlock a range of process address space

26723 **SYNOPSIS**

26724 MLR    #include <sys/mman.h>

26725       int munlock(const void \*addr, size\_t len);

26726

26727 **DESCRIPTION**

26728       Refer to *mlock()*.

26729 **NAME**

26730           munlockall — unlock the address space of a process

26731 **SYNOPSIS**

26732 ML       #include &lt;sys/mman.h&gt;

26733           int munlockall(void);

26734

26735 **DESCRIPTION**26736           Refer to *mlockall()*.

26737 **NAME**

26738           munmap — unmap pages of memory

26739 **SYNOPSIS**

26740 MC3       #include &lt;sys/mman.h&gt;

26741           int munmap(void \*addr, size\_t len);

26742

26743 **DESCRIPTION**

26744       The *munmap()* function shall remove any mappings for those entire pages containing any part of  
 26745       the address space of the process starting at *addr* and continuing for *len* bytes. Further references  
 26746       to these pages shall result in the generation of a SIGSEGV signal to the process. If there are no  
 26747       mappings in the specified address range, then *munmap()* has no effect.

26748       The implementation shall require that *addr* be a multiple of the page size {PAGESIZE}.

26749       If a mapping to be removed was private, any modifications made in this address range shall be  
 26750       discarded.

26751 ML|MLR     Any memory locks (see *mlock()* and *mlockall()*) associated with this address range shall be  
 26752       removed, as if by an appropriate call to *munlock()*.

26753 TYM       If a mapping removed from a typed memory object causes the corresponding address range of  
 26754       the memory pool to be inaccessible by any process in the system except through allocatable  
 26755       mappings (that is, mappings of typed memory objects opened with the  
 26756       POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE flag), then that range of the memory pool shall  
 26757       become deallocated and may become available to satisfy future typed memory allocation  
 26758       requests.

26759       A mapping removed from a typed memory object opened with the  
 26760       POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE flag shall not affect in any way the availability of  
 26761       that typed memory for allocation.

26762       The behavior of this function is unspecified if the mapping was not established by a call to  
 26763       *mmap()*.

26764 **RETURN VALUE**

26765       Upon successful completion, *munmap()* shall return 0; otherwise, it shall return -1 and set *errno*  
 26766       to indicate the error.

26767 **ERRORS**

26768       The *munmap()* function shall fail if:

26769       [EINVAL]       Addresses in the range [*addr*,*addr+len*) are outside the valid range for the  
 26770       address space of a process.

26771       [EINVAL]       The *len* argument is 0.

26772       [EINVAL]       The *addr* argument is not a multiple of the page size as returned by *sysconf()*.

26773 **EXAMPLES**

26774 None.

26775 **APPLICATION USAGE**

26776 The *munmap()* function is only supported if the Memory Mapped Files option or the Shared  
 26777 Memory Objects option is supported.

26778 **RATIONALE**26779 The *munmap()* function corresponds to SVR4, just as the *mmap()* function does.

26780 It is possible that an application has applied process memory locking to a region that contains  
 26781 shared memory. If this has occurred, the *munmap()* call ignores those locks and, if necessary,  
 26782 causes those locks to be removed.

26783 **FUTURE DIRECTIONS**

26784 None.

26785 **SEE ALSO**

26786 *mlock()*, *mlockall()*, *mmap()*, *posix\_typed\_mem\_open()*, *sysconf()*, the Base Definitions volume of  
 26787 IEEE Std 1003.1-2001, <signal.h>, <sys/mman.h>

26788 **CHANGE HISTORY**

26789 First released in Issue 4, Version 2.

26790 **Issue 5**

26791 Moved from X/OPEN UNIX extension to BASE.

26792 Aligned with *munmap()* in the POSIX Realtime Extension as follows:

- 26793 • The DESCRIPTION is extensively reworded.
- 26794 • The SIGBUS error is no longer permitted to be generated.

26795 **Issue 6**

26796 The *munmap()* function is marked as part of the Memory Mapped Files and Shared Memory  
 26797 Objects option.

26798 The following new requirements on POSIX implementations derive from alignment with the  
 26799 Single UNIX Specification:

- 26800 • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of  
 26801 the page size.
- 26802 • The [EINVAL] error conditions are added.

26803 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 26804 • Semantics for typed memory objects are added to the DESCRIPTION.
- 26805 • The *posix\_typed\_mem\_open()* function is added to the SEE ALSO section.

26806 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/36 is applied, changing the margin code |  
 26807 in the SYNOPSIS from MF | SHM to MC3 (notation for MF | SHM | TYM). |

26808 **NAME**

26809 nan, nanf, nanl — return quiet NaN

26810 **SYNOPSIS**

26811 #include <math.h>

26812 double nan(const char \*tagp);

26813 float nanf(const char \*tagp);

26814 long double nanl(const char \*tagp);

26815 **DESCRIPTION**

26816 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
26817 conflict between the requirements described here and the ISO C standard is unintentional. This  
26818 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

26819 The function call *nan("n-char-sequence")* shall be equivalent to:

26820 `strtod("NAN(n-char-sequence)", (char **) NULL);`

26821 The function call *nan("")* shall be equivalent to:

26822 `strtod("NAN()", (char **) NULL)`

26823 If *tagp* does not point to an *n-char* sequence or an empty string, the function call shall be  
26824 equivalent to:

26825 `strtod("NAN", (char **) NULL)`

26826 Function calls to *nanf()* and *nanl()* are equivalent to the corresponding function calls to *strtof()*  
26827 and *strtold()*.

26828 **RETURN VALUE**

26829 These functions shall return a quiet NaN, if available, with content indicated through *tagp*.

26830 If the implementation does not support quiet NaNs, these functions shall return zero.

26831 **ERRORS**

26832 No errors are defined.

26833 **EXAMPLES**

26834 None.

26835 **APPLICATION USAGE**

26836 None.

26837 **RATIONALE**

26838 None.

26839 **FUTURE DIRECTIONS**

26840 None.

26841 **SEE ALSO**

26842 *strtod()*, *strtold()*, the Base Definitions volume of IEEE Std 1003.1-2001, <math.h>

26843 **CHANGE HISTORY**

26844 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

26845 **NAME**26846 nanosleep — high resolution sleep (**REALTIME**)26847 **SYNOPSIS**26848 TMR `#include <time.h>`26849 `int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);`

26850

26851 **DESCRIPTION**

26852 The *nanosleep()* function shall cause the current thread to be suspended from execution until  
 26853 either the time interval specified by the *rqtp* argument has elapsed or a signal is delivered to the  
 26854 calling thread, and its action is to invoke a signal-catching function or to terminate the process.  
 26855 The suspension time may be longer than requested because the argument value is rounded up to  
 26856 an integer multiple of the sleep resolution or because of the scheduling of other activity by the  
 26857 system. But, except for the case of being interrupted by a signal, the suspension time shall not be  
 26858 less than the time specified by *rqtp*, as measured by the system clock `CLOCK_REALTIME`.

26859 The use of the *nanosleep()* function has no effect on the action or blockage of any signal.26860 **RETURN VALUE**26861 If the *nanosleep()* function returns because the requested time has elapsed, its return value shall  
26862 be zero.

26863 If the *nanosleep()* function returns because it has been interrupted by a signal, it shall return a  
 26864 value of `-1` and set *errno* to indicate the interruption. If the *rmtp* argument is non-NULL, the  
 26865 **timespec** structure referenced by it is updated to contain the amount of time remaining in the  
 26866 interval (the requested time minus the time actually slept). If the *rmtp* argument is NULL, the  
 26867 remaining time is not returned.

26868 If *nanosleep()* fails, it shall return a value of `-1` and set *errno* to indicate the error.26869 **ERRORS**26870 The *nanosleep()* function shall fail if:26871 [EINTR] The *nanosleep()* function was interrupted by a signal.26872 [EINVAL] The *rqtp* argument specified a nanosecond value less than zero or greater than  
26873 or equal to 1 000 million.26874 **EXAMPLES**

26875 None.

26876 **APPLICATION USAGE**

26877 None.

26878 **RATIONALE**

26879 It is common to suspend execution of a process for an interval in order to poll the status of a  
 26880 non-interrupting function. A large number of actual needs can be met with a simple extension to  
 26881 *sleep()* that provides finer resolution.

26882 In the POSIX.1-1990 standard and SVR4, it is possible to implement such a routine, but the  
 26883 frequency of wakeup is limited by the resolution of the *alarm()* and *sleep()* functions. In 4.3 BSD,  
 26884 it is possible to write such a routine using no static storage and reserving no system facilities.  
 26885 Although it is possible to write a function with similar functionality to *sleep()* using the  
 26886 remainder of the *timer\_\**() functions, such a function requires the use of signals and the  
 26887 reservation of some signal number. This volume of IEEE Std 1003.1-2001 requires that  
 26888 *nanosleep()* be non-intrusive of the signals function.

26889 The *nanosleep()* function shall return a value of 0 on success and –1 on failure or if interrupted.  
26890 This latter case is different from *sleep()*. This was done because the remaining time is returned  
26891 via an argument structure pointer, *rmtp*, instead of as the return value.

26892 **FUTURE DIRECTIONS**

26893 None.

26894 **SEE ALSO**

26895 *clock\_nanosleep()*, *sleep()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**time.h**>

26896 **CHANGE HISTORY**

26897 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

26898 **Issue 6**

26899 The *nanosleep()* function is marked as part of the Timers option.

26900 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
26901 implementation does not support the Timers option.

26902 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/37 is applied, updating the SEE ALSO  
26903 section to include the *clock\_nanosleep()* function.

26904 **NAME**

26905 nearbyint, nearbyintf, nearbyintl — floating-point rounding functions

26906 **SYNOPSIS**

26907 #include &lt;math.h&gt;

26908 double nearbyint(double x);

26909 float nearbyintf(float x);

26910 long double nearbyintl(long double x);

26911 **DESCRIPTION**

26912 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 26913 conflict between the requirements described here and the ISO C standard is unintentional. This  
 26914 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

26915 These functions shall round their argument to an integer value in floating-point format, using  
 26916 the current rounding direction and without raising the inexact floating-point exception.

26917 An application wishing to check for error situations should set *errno* to zero and call  
 26918 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 26919 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 26920 zero, an error has occurred.

26921 **RETURN VALUE**

26922 Upon successful completion, these functions shall return the rounded integer value.

26923 **MX** If *x* is NaN, a NaN shall be returned.26924 If *x* is  $\pm 0$ ,  $\pm 0$  shall be returned.26925 If *x* is  $\pm\text{Inf}$ , *x* shall be returned.

26926 **XSI** If the correct value would cause overflow, a range error shall occur and *nearbyint*(), *nearbyintf*(),  
 26927 and *nearbyintl*() shall return the value of the macro  $\pm\text{HUGE\_VAL}$ ,  $\pm\text{HUGE\_VALF}$ , and  
 26928  $\pm\text{HUGE\_VALL}$  (with the same sign as *x*), respectively.

26929 **ERRORS**

26930 These functions shall fail if:

26931 **XSI** **Range Error** The result would cause an overflow.

26932 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 26933 then *errno* shall be set to [ERANGE]. If the integer expression  
 26934 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow  
 26935 floating-point exception shall be raised.

26936 **EXAMPLES**

26937 None.

26938 **APPLICATION USAGE**

26939 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 26940 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

26941 **RATIONALE**

26942 None.

26943 **FUTURE DIRECTIONS**

26944 None.

26945 **SEE ALSO**

26946            *feclearexcept()*, *fetetestexcept()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18,  
26947            Treatment of Error Conditions for Mathematical Functions, <**math.h**>

26948 **CHANGE HISTORY**

26949            First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

26950 **NAME**

26951 nextafter, nextafterf, nextafterl, nexttoward, nexttowardf, nexttowardl — next representable  
 26952 floating-point number

26953 **SYNOPSIS**

```
26954 #include <math.h>

26955 double nextafter(double x, double y);
26956 float nextafterf(float x, float y);
26957 long double nextafterl(long double x, long double y);
26958 double nexttoward(double x, long double y);
26959 float nexttowardf(float x, long double y);
26960 long double nexttowardl(long double x, long double y);
```

26961 **DESCRIPTION**

26962 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 26963 conflict between the requirements described here and the ISO C standard is unintentional. This  
 26964 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

26965 The *nextafter()*, *nextafterf()*, and *nextafterl()* functions shall compute the next representable  
 26966 floating-point value following *x* in the direction of *y*. Thus, if *y* is less than *x*, *nextafter()* shall  
 26967 return the largest representable floating-point number less than *x*. The *nextafter()*, *nextafterf()*,  
 26968 and *nextafterl()* functions shall return *y* if *x* equals *y*.

26969 The *nexttoward()*, *nexttowardf()*, and *nexttowardl()* functions shall be equivalent to the  
 26970 corresponding *nextafter()* functions, except that the second parameter shall have type **long**  
 26971 **double** and the functions shall return *y* converted to the type of the function if *x* equals *y*.

26972 An application wishing to check for error situations should set *errno* to zero and call  
 26973 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 26974 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 26975 zero, an error has occurred.

26976 **RETURN VALUE**

26977 Upon successful completion, these functions shall return the next representable floating-point  
 26978 value following *x* in the direction of *y*.

26979 If *x*==*y*, *y* (of the type *x*) shall be returned.

26980 If *x* is finite and the correct function value would overflow, a range error shall occur and  
 26981 ±HUGE\_VAL, ±HUGE\_VALF, and ±HUGE\_VALL (with the same sign as *x*) shall be returned as  
 26982 appropriate for the return type of the function.

26983 **MX** If *x* or *y* is NaN, a NaN shall be returned.

26984 If *x*!=*y* and the correct function value is subnormal, zero, or underflows, a range error shall  
 26985 occur, and either the correct function value (if representable) or 0.0 shall be returned.

26986 **ERRORS**

26987 These functions shall fail if:

26988 **Range Error** The correct value overflows.

26989 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 26990 then *errno* shall be set to [ERANGE]. If the integer expression  
 26991 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow  
 26992 floating-point exception shall be raised.

26993 **MX** **Range Error** The correct value is subnormal or underflows.

26994 If the integer expression (`math_errhandling & MATH_ERRNO`) is non-zero,  
26995 then *errno* shall be set to [ERANGE]. If the integer expression  
26996 (`math_errhandling & MATH_ERREXCEPT`) is non-zero, then the underflow  
26997 floating-point exception shall be raised.

**26998 EXAMPLES**

26999 None.

**27000 APPLICATION USAGE**

27001 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`  
27002 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

**27003 RATIONALE**

27004 None.

**27005 FUTURE DIRECTIONS**

27006 None.

**27007 SEE ALSO**

27008 *feclearexcept()*, *fetestexcept()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18,  
27009 Treatment of Error Conditions for Mathematical Functions, <**math.h**>

**27010 CHANGE HISTORY**

27011 First released in Issue 4, Version 2.

**27012 Issue 5**

27013 Moved from X/OPEN UNIX extension to BASE.

**27014 Issue 6**

27015 The *nextafter()* function is no longer marked as an extension.

27016 The *nextafterf()*, *nextafterl()*, *nexttoward()*, *nexttowardf()*, and *nexttowardl()* functions are added  
27017 for alignment with the ISO/IEC 9899:1999 standard.

27018 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
27019 revised to align with the ISO/IEC 9899:1999 standard.

27020 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
27021 marked.

27022 **NAME**

27023 nftw — walk a file tree

27024 **SYNOPSIS**27025 XSI 

```
#include <ftw.h>
```

```
27026 int nftw(const char *path, int (*fn)(const char *,
27027 const struct stat *, int, struct FTW *), int depth, int flags);
27028
```

27029 **DESCRIPTION**

27030 The *nftw()* function shall recursively descend the directory hierarchy rooted in *path*. The *nftw()*  
 27031 function has a similar effect to *ftw()* except that it takes an additional argument *flags*, which is a  
 27032 bitwise-inclusive OR of zero or more of the following flags:

27033 **FTW\_CHDIR** If set, *nftw()* shall change the current working directory to each directory as it  
 27034 reports files in that directory. If clear, *nftw()* shall not change the current  
 27035 working directory.

27036 **FTW\_DEPTH** If set, *nftw()* shall report all files in a directory before reporting the directory  
 27037 itself. If clear, *nftw()* shall report any directory before reporting the files in that  
 27038 directory.

27039 **FTW\_MOUNT** If set, *nftw()* shall only report files in the same file system as *path*. If clear,  
 27040 *nftw()* shall report all files encountered during the walk.

27041 **FTW\_PHYS** If set, *nftw()* shall perform a physical walk and shall not follow symbolic links.

27042 If **FTW\_PHYS** is clear and **FTW\_DEPTH** is set, *nftw()* shall follow links instead of reporting  
 27043 them, but shall not report any directory that would be a descendant of itself. If **FTW\_PHYS** is  
 27044 clear and **FTW\_DEPTH** is clear, *nftw()* shall follow links instead of reporting them, but shall not  
 27045 report the contents of any directory that would be a descendant of itself.

27046 At each file it encounters, *nftw()* shall call the user-supplied function *fn* with four arguments:

- 27047 • The first argument is the pathname of the object.
- 27048 • The second argument is a pointer to the **stat** buffer containing information on the object.
- 27049 • The third argument is an integer giving additional information. Its value is one of the  
 27050 following:

27051 **FTW\_F** The object is a file.

27052 **FTW\_D** The object is a directory.

27053 **FTW\_DP** The object is a directory and subdirectories have been visited. (This condition  
 27054 shall only occur if the **FTW\_DEPTH** flag is included in *flags*.)

27055 **FTW\_SL** The object is a symbolic link. (This condition shall only occur if the **FTW\_PHYS**  
 27056 flag is included in *flags*.)

27057 **FTW\_SLN** The object is a symbolic link that does not name an existing file. (This  
 27058 condition shall only occur if the **FTW\_PHYS** flag is not included in *flags*.)

27059 **FTW\_DNR** The object is a directory that cannot be read. The *fn* function shall not be called  
 27060 for any of its descendants.

27061 **FTW\_NS** The *stat()* function failed on the object because of lack of appropriate  
 27062 permission. The **stat** buffer passed to *fn* is undefined. Failure of *stat()* for any  
 27063 other reason is considered an error and *nftw()* shall return  $-1$ .

27064 • The fourth argument is a pointer to an **FTW** structure. The value of **base** is the offset of the  
 27065 object's filename in the pathname passed as the first argument to *fn*. The value of **level**  
 27066 indicates depth relative to the root of the walk, where the root level is 0.

27067 The results are unspecified if the application-supplied *fn* function does not preserve the current  
 27068 working directory.

27069 The argument *depth* sets the maximum number of file descriptors that shall be used by *nftw()*  
 27070 while traversing the file tree. At most one file descriptor shall be used for each directory level.

27071 The *nftw()* function need not be reentrant. A function that is not required to be reentrant is not  
 27072 required to be thread-safe.

#### 27073 RETURN VALUE

27074 The *nftw()* function shall continue until the first of the following conditions occurs:

- 27075 • An invocation of *fn* shall return a non-zero value, in which case *nftw()* shall return that value.
- 27076 • The *nftw()* function detects an error other than [EACCES] (see FTW\_DNR and FTW\_NS  
 27077 above), in which case *nftw()* shall return  $-1$  and set *errno* to indicate the error.
- 27078 • The tree is exhausted, in which case *nftw()* shall return 0.

#### 27079 ERRORS

27080 The *nftw()* function shall fail if:

27081 [EACCES] Search permission is denied for any component of *path* or read permission is  
 27082 denied for *path*, or *fn* returns  $-1$  and does not reset *errno*.

27083 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 27084 argument.

27085 [ENAMETOOLONG]

27086 The length of the *path* argument exceeds {PATH\_MAX} or a pathname  
 27087 component is longer than {NAME\_MAX}.

27088 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

27089 [ENOTDIR] A component of *path* is not a directory.

27090 [EOVERFLOW] A field in the **stat** structure cannot be represented correctly in the current  
 27091 programming environment for one or more files found in the file hierarchy.

27092 The *nftw()* function may fail if:

27093 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 27094 resolution of the *path* argument.

27095 [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

27096 [ENAMETOOLONG]

27097 Pathname resolution of a symbolic link produced an intermediate result  
 27098 whose length exceeds {PATH\_MAX}.

27099 [ENFILE] Too many files are currently open in the system.

27100 In addition, *errno* may be set if the function pointed to by *fn* causes *errno* to be set.

27101 **EXAMPLES**

27102 The following example walks the `/tmp` directory and its subdirectories, calling the `nftw()`  
 27103 function for every directory entry, to a maximum of 5 levels deep.

```
27104 #include <ftw.h>
27105 ...
27106 int nftwfunc(const char *, const struct stat *, int, struct FTW *);
27107
27107 int nftwfunc(const char *filename, const struct stat *statptr,
27108 int fileflags, struct FTW *pftw)
27109 {
27110 return 0;
27111 }
27112 ...
27113 char *startpath = "/tmp";
27114 int depth = 5;
27115 int flags = FTW_CHDIR | FTW_DEPTH | FTW_MOUNT;
27116 int ret;
27117
27117 ret = nftw(startpath, nftwfunc, depth, flags);
```

27118 **APPLICATION USAGE**

27119 None.

27120 **RATIONALE**

27121 None.

27122 **FUTURE DIRECTIONS**

27123 None.

27124 **SEE ALSO**

27125 `lstat()`, `opendir()`, `readdir()`, `stat()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<ftw.h>`

27126 **CHANGE HISTORY**

27127 First released in Issue 4, Version 2.

27128 **Issue 5**

27129 Moved from X/OPEN UNIX extension to BASE.

27130 In the DESCRIPTION, the definition of the *depth* argument is clarified.

27131 **Issue 6**

27132 The Open Group Base Resolution bwg97-003 is applied.

27133 The ERRORS section is updated as follows:

- 27134 • The wording of the mandatory [ELOOP] error condition is updated.
- 27135 • A second optional [ELOOP] error condition is added.
- 27136 • The [EOVERFLOW] mandatory error condition is added.

27137 Text is added to the DESCRIPTION to say that the `nftw()` function need not be reentrant and  
 27138 that the results are unspecified if the application-supplied *fn* function does not preserve the  
 27139 current working directory.

27140 **NAME**

27141 nice — change the nice value of a process

27142 **SYNOPSIS**27143 XSI `#include <unistd.h>`27144 `int nice(int incr);`

27145

27146 **DESCRIPTION**

27147 The *nice()* function shall add the value of *incr* to the nice value of the calling process. A process' nice value is a non-negative number for which a more positive value shall result in less favorable scheduling.

27150 A maximum nice value of  $2*\{\text{NZERO}\}-1$  and a minimum nice value of 0 shall be imposed by the system. Requests for values above or below these limits shall result in the nice value being set to the corresponding limit. Only a process with appropriate privileges can lower the nice value.

27153 PS|TPS Calling the *nice()* function has no effect on the priority of processes or threads with policy SCHED\_FIFO or SCHED\_RR. The effect on processes or threads with other scheduling policies is implementation-defined.

27156 The nice value set with *nice()* shall be applied to the process. If the process is multi-threaded, the nice value shall affect all system scope threads in the process.

27158 As  $-1$  is a permissible return value in a successful situation, an application wishing to check for error situations should set *errno* to 0, then call *nice()*, and if it returns  $-1$ , check to see whether *errno* is non-zero.

27161 **RETURN VALUE**

27162 Upon successful completion, *nice()* shall return the new nice value  $-\{\text{NZERO}\}$ . Otherwise,  $-1$  shall be returned, the process' nice value shall not be changed, and *errno* shall be set to indicate the error.

27165 **ERRORS**27166 The *nice()* function shall fail if:

27167 [EPERM] The *incr* argument is negative and the calling process does not have appropriate privileges.

27169 **EXAMPLES**27170 **Changing the Nice Value**

27171 The following example adds the value of the *incr* argument,  $-20$ , to the nice value of the calling process.

27173 `#include <unistd.h>`27174 `...`27175 `int incr = -20;`27176 `int ret;`27177 `ret = nice(incr);`27178 **APPLICATION USAGE**

27179 None.

27180 **RATIONALE**

27181       None.

27182 **FUTURE DIRECTIONS**

27183       None.

27184 **SEE ALSO**27185       *getpriority()*, *setpriority()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**limits.h**>,  
27186       <**unistd.h**>27187 **CHANGE HISTORY**

27188       First released in Issue 1. Derived from Issue 1 of the SVID.

27189 **Issue 5**27190       A statement is added to the description indicating the effects of this function on the different  
27191       scheduling policies and multi-threaded processes.

27192 **NAME**

27193 nl\_langinfo — language information

27194 **SYNOPSIS**

27195 xSI #include &lt;langinfo.h&gt;

27196 char \*nl\_langinfo(nl\_item item);

27197

27198 **DESCRIPTION**

27199 The *nl\_langinfo()* function shall return a pointer to a string containing information relevant to  
 27200 the particular language or cultural area defined in the program's locale (see <langinfo.h>). The  
 27201 manifest constant names and values of *item* are defined in <langinfo.h>. For example:

27202 nl\_langinfo(ABDAY\_1)

27203 would return a pointer to the string "Dom" if the identified language was Portuguese, and  
 27204 "Sun" if the identified language was English.

27205 Calls to *setlocale()* with a category corresponding to the category of *item* (see <langinfo.h>), or to  
 27206 the category *LC\_ALL*, may overwrite the array pointed to by the return value.

27207 The *nl\_langinfo()* function need not be reentrant. A function that is not required to be reentrant is  
 27208 not required to be thread-safe.

27209 **RETURN VALUE**

27210 In a locale where *langinfo* data is not defined, *nl\_langinfo()* shall return a pointer to the  
 27211 corresponding string in the POSIX locale. In all locales, *nl\_langinfo()* shall return a pointer to an  
 27212 empty string if *item* contains an invalid setting.

27213 This pointer may point to static data that may be overwritten on the next call.

27214 **ERRORS**

27215 No errors are defined.

27216 **EXAMPLES**27217 **Getting Date and Time Formatting Information**

27218 The following example returns a pointer to a string containing date and time formatting  
 27219 information, as defined in the *LC\_TIME* category of the current locale.

27220 #include &lt;time.h&gt;

27221 #include &lt;langinfo.h&gt;

27222 ...

27223 strftime(datestring, sizeof(datestring), nl\_langinfo(D\_T\_FMT), tm);

27224 ...

27225 **APPLICATION USAGE**

27226 The array pointed to by the return value should not be modified by the program, but may be  
 27227 modified by further calls to *nl\_langinfo()*.

27228 **RATIONALE**

27229 None.

27230 **FUTURE DIRECTIONS**

27231 None.

27232 **SEE ALSO**

27233        *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <**langinfo.h**>,  
27234        <**nl\_types.h**>

27235 **CHANGE HISTORY**

27236        First released in Issue 2.

27237 **Issue 5**

27238        The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

27239        A note indicating that this function need not be reentrant is added to the DESCRIPTION.

27240 **NAME**

27241 nrand48 — generate uniformly distributed pseudo-random non-negative long integers

27242 **SYNOPSIS**

27243 xSI #include <stdlib.h>

27244 long nrand48(unsigned short xsubi[3]);

27245

27246 **DESCRIPTION**

27247 Refer to *drand48()*.

27248 **NAME**

27249           ntohl, ntohs — convert values between host and network byte order

27250 **SYNOPSIS**

27251           #include <arpa/inet.h>

27252           uint32\_t ntohl(uint32\_t *netlong*);

27253           uint16\_t ntohs(uint16\_t *netshort*);

27254 **DESCRIPTION**

27255           Refer to *htonl()*.

## 27256 NAME

27257 open — open a file

## 27258 SYNOPSIS

27259 OH #include &lt;sys/stat.h&gt;

27260 #include &lt;fcntl.h&gt;

27261 int open(const char \*path, int oflag, ... );

## 27262 DESCRIPTION

27263 The *open()* function shall establish the connection between a file and a file descriptor. It shall  
 27264 create an open file description that refers to a file and a file descriptor that refers to that open file  
 27265 description. The file descriptor is used by other I/O functions to refer to that file. The *path*  
 27266 argument points to a pathname naming the file.

27267 The *open()* function shall return a file descriptor for the named file that is the lowest file  
 27268 descriptor not currently open for that process. The open file description is new, and therefore the  
 27269 file descriptor shall not share it with any other process in the system. The FD\_CLOEXEC file  
 27270 descriptor flag associated with the new file descriptor shall be cleared.

27271 The file offset used to mark the current position within the file shall be set to the beginning of the  
 27272 file.

27273 The file status flags and file access modes of the open file description shall be set according to  
 27274 the value of *oflag*.

27275 Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list,  
 27276 defined in <fcntl.h>. Applications shall specify exactly one of the first three values (file access  
 27277 modes) below in the value of *oflag*:

27278 O\_RDONLY Open for reading only.

27279 O\_WRONLY Open for writing only.

27280 O\_RDWR Open for reading and writing. The result is undefined if this flag is applied to  
27281 a FIFO.

27282 Any combination of the following may be used:

27283 O\_APPEND If set, the file offset shall be set to the end of the file prior to each write.

27284 O\_CREAT If the file exists, this flag has no effect except as noted under O\_EXCL below.  
 27285 Otherwise, the file shall be created; the user ID of the file shall be set to the  
 27286 effective user ID of the process; the group ID of the file shall be set to the  
 27287 group ID of the file's parent directory or to the effective group ID of the  
 27288 process; and the access permission bits (see <sys/stat.h>) of the file mode shall  
 27289 be set to the value of the third argument taken as type **mode\_t** modified as  
 27290 follows: a bitwise AND is performed on the file-mode bits and the  
 27291 corresponding bits in the complement of the process' file mode creation mask.  
 27292 Thus, all bits in the file mode whose corresponding bit in the file mode  
 27293 creation mask is set are cleared. When bits other than the file permission bits  
 27294 are set, the effect is unspecified. The third argument does not affect whether  
 27295 the file is open for reading, writing, or for both. Implementations shall provide  
 27296 a way to initialize the file's group ID to the group ID of the parent directory.  
 27297 Implementations may, but need not, provide an implementation-defined way  
 27298 to initialize the file's group ID to the effective group ID of the calling process.

27299 SIO O\_DSYNC Write I/O operations on the file descriptor shall complete as defined by  
 27300 synchronized I/O data integrity completion.

|           |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 27301     | O_EXCL     | If O_CREAT and O_EXCL are set, <i>open()</i> shall fail if the file exists. The check for the existence of the file and the creation of the file if it does not exist shall be atomic with respect to other threads executing <i>open()</i> naming the same filename in the same directory with O_EXCL and O_CREAT set. If O_EXCL and O_CREAT are set, and <i>path</i> names a symbolic link, <i>open()</i> shall fail and set <i>errno</i> to [EEXIST], regardless of the contents of the symbolic link. If O_EXCL is set and O_CREAT is not set, the result is undefined. |
| 27302     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27303     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27304     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27305     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27306     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27307     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27308     | O_NOCTTY   | If set and <i>path</i> identifies a terminal device, <i>open()</i> shall not cause the terminal device to become the controlling terminal for the process.                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 27309     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27310     | O_NONBLOCK | When opening a FIFO with O_RDONLY or O_WRONLY set:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 27311     |            | • If O_NONBLOCK is set, an <i>open()</i> for reading-only shall return without delay. An <i>open()</i> for writing-only shall return an error if no process currently has the file open for reading.                                                                                                                                                                                                                                                                                                                                                                        |
| 27312     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27313     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27314     |            | • If O_NONBLOCK is clear, an <i>open()</i> for reading-only shall block the calling thread until a thread opens the file for writing. An <i>open()</i> for writing-only shall block the calling thread until a thread opens the file for reading.                                                                                                                                                                                                                                                                                                                           |
| 27315     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27316     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27317     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27318     |            | When opening a block special or character special file that supports non-blocking opens:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 27319     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27320     |            | • If O_NONBLOCK is set, the <i>open()</i> function shall return without blocking for the device to be ready or available. Subsequent behavior of the device is device-specific.                                                                                                                                                                                                                                                                                                                                                                                             |
| 27321     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27322     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27323     |            | • If O_NONBLOCK is clear, the <i>open()</i> function shall block the calling thread until the device is ready or available before returning.                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 27324     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27325     |            | Otherwise, the behavior of O_NONBLOCK is unspecified.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 27326 SIO | O_RSYNC    | Read I/O operations on the file descriptor shall complete at the same level of integrity as specified by the O_DSYNC and O_SYNC flags. If both O_DSYNC and O_RSYNC are set in <i>oflag</i> , all I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion. If both O_SYNC and O_RSYNC are set in flags, all I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.                                                                                             |
| 27327     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27328     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27329     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27330     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27331     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27332     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27333 SIO | O_SYNC     | Write I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27334     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27335     | O_TRUNC    | If the file exists and is a regular file, and the file is successfully opened O_RDWR or O_WRONLY, its length shall be truncated to 0, and the mode and owner shall be unchanged. It shall have no effect on FIFO special files or terminal device files. Its effect on other file types is implementation-defined. The result of using O_TRUNC with O_RDONLY is undefined.                                                                                                                                                                                                  |
| 27336     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27337     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27338     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27339     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27340     |            | If O_CREAT is set and the file did not previously exist, upon successful completion, <i>open()</i> shall mark for update the <i>st_atime</i> , <i>st_ctime</i> , and <i>st_mtime</i> fields of the file and the <i>st_ctime</i> and <i>st_mtime</i> fields of the parent directory.                                                                                                                                                                                                                                                                                         |
| 27341     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27342     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27343     |            | If O_TRUNC is set and the file did previously exist, upon successful completion, <i>open()</i> shall mark for update the <i>st_ctime</i> and <i>st_mtime</i> fields of the file.                                                                                                                                                                                                                                                                                                                                                                                            |
| 27344     |            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

|                                                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 27345 SIO<br>27346                                     | If both the O_SYNC and O_DSYNC flags are set, the effect is as if only the O_SYNC flag was set.                                                                                                                                                                                                                                                                                                                                                              |
| 27347 XSR<br>27348<br>27349<br>27350<br>27351<br>27352 | If <i>path</i> refers to a STREAMS file, <i>oflag</i> may be constructed from O_NONBLOCK OR'ed with either O_RDONLY, O_WRONLY, or O_RDWR. Other flag values are not applicable to STREAMS devices and shall have no effect on them. The value O_NONBLOCK affects the operation of STREAMS drivers and certain functions applied to file descriptors associated with STREAMS files. For STREAMS drivers, the implementation of O_NONBLOCK is device-specific. |
| 27353 XSI<br>27354<br>27355                            | If <i>path</i> names the master side of a pseudo-terminal device, then it is unspecified whether <i>open()</i> locks the slave side so that it cannot be opened. Conforming applications shall call <i>unlockpt()</i> before opening the slave side.                                                                                                                                                                                                         |
| 27356<br>27357                                         | The largest value that can be represented correctly in an object of type <b>off_t</b> shall be established as the offset maximum in the open file description.                                                                                                                                                                                                                                                                                               |
| 27358                                                  | <b>RETURN VALUE</b>                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 27359<br>27360<br>27361                                | Upon successful completion, the function shall open the file and return a non-negative integer representing the lowest numbered unused file descriptor. Otherwise, <b>-1</b> shall be returned and <i>errno</i> set to indicate the error. No files shall be created or modified if the function returns <b>-1</b> .                                                                                                                                         |
| 27362                                                  | <b>ERRORS</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 27363                                                  | The <i>open()</i> function shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 27364<br>27365<br>27366<br>27367                       | [EACCES] Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by <i>oflag</i> are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created, or O_TRUNC is specified and write permission is denied.                                                                                                                                   |
| 27368                                                  | [EEXIST] O_CREAT and O_EXCL are set, and the named file exists.                                                                                                                                                                                                                                                                                                                                                                                              |
| 27369                                                  | [EINTR] A signal was caught during <i>open()</i> .                                                                                                                                                                                                                                                                                                                                                                                                           |
| 27370 SIO                                              | [EINVAL] The implementation does not support synchronized I/O for this file.                                                                                                                                                                                                                                                                                                                                                                                 |
| 27371 XSR<br>27372                                     | [EIO] The <i>path</i> argument names a STREAMS file and a hangup or error occurred during the <i>open()</i> .                                                                                                                                                                                                                                                                                                                                                |
| 27373                                                  | [EISDIR] The named file is a directory and <i>oflag</i> includes O_WRONLY or O_RDWR.                                                                                                                                                                                                                                                                                                                                                                         |
| 27374<br>27375                                         | [ELOOP] A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.                                                                                                                                                                                                                                                                                                                                                           |
| 27376                                                  | [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.                                                                                                                                                                                                                                                                                                                                                                              |
| 27377<br>27378<br>27379                                | [ENAMETOOLONG] The length of the <i>path</i> argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.                                                                                                                                                                                                                                                                                                                                  |
| 27380                                                  | [ENFILE] The maximum allowable number of files is currently open in the system.                                                                                                                                                                                                                                                                                                                                                                              |
| 27381<br>27382<br>27383                                | [ENOENT] O_CREAT is not set and the named file does not exist; or O_CREAT is set and either the path prefix does not exist or the <i>path</i> argument points to an empty string.                                                                                                                                                                                                                                                                            |
| 27384 XSR<br>27385                                     | [ENOSR] The <i>path</i> argument names a STREAMS-based file and the system is unable to allocate a STREAM.                                                                                                                                                                                                                                                                                                                                                   |
| 27386<br>27387                                         | [ENOSPC] The directory or file system that would contain the new file cannot be expanded, the file does not exist, and O_CREAT is specified.                                                                                                                                                                                                                                                                                                                 |

|           |                |                                                                                                                                                                      |
|-----------|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 27388     | [ENOTDIR]      | A component of the path prefix is not a directory.                                                                                                                   |
| 27389     | [ENXIO]        | O_NONBLOCK is set, the named file is a FIFO, O_WRONLY is set, and no process has the file open for reading.                                                          |
| 27390     |                |                                                                                                                                                                      |
| 27391     | [ENXIO]        | The named file is a character special or block special file, and the device associated with this special file does not exist.                                        |
| 27392     |                |                                                                                                                                                                      |
| 27393     | [EOVERFLOW]    | The named file is a regular file and the size of the file cannot be represented correctly in an object of type <code>off_t</code> .                                  |
| 27394     |                |                                                                                                                                                                      |
| 27395     | [EROFS]        | The named file resides on a read-only file system and either O_WRONLY, O_RDWR, O_CREAT (if the file does not exist), or O_TRUNC is set in the <i>oflag</i> argument. |
| 27396     |                |                                                                                                                                                                      |
| 27397     |                |                                                                                                                                                                      |
| 27398     |                | The <i>open()</i> function may fail if:                                                                                                                              |
| 27399 XSI | [EAGAIN]       | The <i>path</i> argument names the slave side of a pseudo-terminal device that is locked.                                                                            |
| 27400     |                |                                                                                                                                                                      |
| 27401     | [EINVAL]       | The value of the <i>oflag</i> argument is not valid.                                                                                                                 |
| 27402     | [ELOOP]        | More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.                                                               |
| 27403     |                |                                                                                                                                                                      |
| 27404     | [ENAMETOOLONG] | As a result of encountering a symbolic link in resolution of the <i>path</i> argument, the length of the substituted pathname string exceeded {PATH_MAX}.            |
| 27405     |                |                                                                                                                                                                      |
| 27406     |                |                                                                                                                                                                      |
| 27407 XSR | [ENOMEM]       | The <i>path</i> argument names a STREAMS file and the system is unable to allocate resources.                                                                        |
| 27408     |                |                                                                                                                                                                      |
| 27409     | [ETXTBSY]      | The file is a pure procedure (shared text) file that is being executed and <i>oflag</i> is O_WRONLY or O_RDWR.                                                       |
| 27410     |                |                                                                                                                                                                      |

## 27411 EXAMPLES

### 27412 Opening a File for Writing by the Owner

27413 The following example opens the file `/tmp/file`, either by creating it (if it does not already exist),  
 27414 or by truncating its length to 0 (if it does exist). In the former case, if the call creates a new file,  
 27415 the access permission bits in the file mode of the file are set to permit reading and writing by the  
 27416 owner, and to permit reading only by group members and others.

27417 If the call to *open()* is successful, the file is opened for writing.

```

27418 #include <fcntl.h>
27419 ...
27420 int fd;
27421 mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
27422 char *filename = "/tmp/file";
27423 ...
27424 fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, mode);
27425 ...

```

27426 **Opening a File Using an Existence Check**

27427 The following example uses the *open()* function to try to create the **LOCKFILE** file and open it  
 27428 for writing. Since the *open()* function specifies the **O\_EXCL** flag, the call fails if the file already  
 27429 exists. In that case, the program assumes that someone else is updating the password file and  
 27430 exits.

```
27431 #include <fcntl.h>
27432 #include <stdio.h>
27433 #include <stdlib.h>

27434 #define LOCKFILE "/etc/ptmp"
27435 ...
27436 int pfd; /* Integer for file descriptor returned by open() call. */
27437 ...
27438 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
27439 S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
27440 {
27441 fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
27442 exit(1);
27443 }
27444 ...
```

27445 **Opening a File for Writing**

27446 The following example opens a file for writing, creating the file if it does not already exist. If the  
 27447 file does exist, the system truncates the file to zero bytes.

```
27448 #include <fcntl.h>
27449 #include <stdio.h>
27450 #include <stdlib.h>

27451 #define LOCKFILE "/etc/ptmp"
27452 ...
27453 int pfd;
27454 char filename[PATH_MAX+1];
27455 ...
27456 if ((pfd = open(filename, O_WRONLY | O_CREAT | O_TRUNC,
27457 S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
27458 {
27459 perror("Cannot open output file\n"); exit(1);
27460 }
27461 ...
```

27462 **APPLICATION USAGE**

27463 None.

27464 **RATIONALE**

27465 Except as specified in this volume of IEEE Std 1003.1-2001, the flags allowed in *oflag* are not  
 27466 mutually-exclusive and any number of them may be used simultaneously.

27467 Some implementations permit opening FIFOs with **O\_RDWR**. Since FIFOs could be  
 27468 implemented in other ways, and since two file descriptors can be used to the same effect, this  
 27469 possibility is left as undefined.

27470 See *getgroups()* about the group of a newly created file.

27471 The use of *open()* to create a regular file is preferable to the use of *creat()*, because the latter is  
27472 redundant and included only for historical reasons.

27473 The use of the O\_TRUNC flag on FIFOs and directories (pipes cannot be *open()*-ed) must be  
27474 permissible without unexpected side effects (for example, *creat()* on a FIFO must not remove  
27475 data). Since terminal special files might have type-ahead data stored in the buffer, O\_TRUNC  
27476 should not affect their content, particularly if a program that normally opens a regular file  
27477 should open the current controlling terminal instead. Other file types, particularly  
27478 implementation-defined ones, are left implementation-defined.

27479 IEEE Std 1003.1-2001 permits [EACCES] to be returned for conditions other than those explicitly  
27480 listed.

27481 The O\_NOCTTY flag was added to allow applications to avoid unintentionally acquiring a  
27482 controlling terminal as a side effect of opening a terminal file. This volume of  
27483 IEEE Std 1003.1-2001 does not specify how a controlling terminal is acquired, but it allows an  
27484 implementation to provide this on *open()* if the O\_NOCTTY flag is not set and other conditions  
27485 specified in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal  
27486 Interface are met. The O\_NOCTTY flag is an effective no-op if the file being opened is not a  
27487 terminal device.

27488 In historical implementations the value of O\_RDONLY is zero. Because of that, it is not possible  
27489 to detect the presence of O\_RDONLY and another option. Future implementations should  
27490 encode O\_RDONLY and O\_WRONLY as bit flags so that:

```
27491 O_RDONLY | O_WRONLY == O_RDWR
```

27492 In general, the *open()* function follows the symbolic link if *path* names a symbolic link. However,  
27493 the *open()* function, when called with O\_CREAT and O\_EXCL, is required to fail with [EEXIST]  
27494 if *path* names an existing symbolic link, even if the symbolic link refers to a nonexistent file. This  
27495 behavior is required so that privileged applications can create a new file in a known location  
27496 without the possibility that a symbolic link might cause the file to be created in a different  
27497 location.

27498 For example, a privileged application that must create a file with a predictable name in a user-  
27499 writable directory, such as the user's home directory, could be compromised if the user creates a  
27500 symbolic link with that name that refers to a nonexistent file in a system directory. If the user can  
27501 influence the contents of a file, the user could compromise the system by creating a new system  
27502 configuration or spool file that would then be interpreted by the system. The test for a symbolic  
27503 link which refers to a nonexistent file must be atomic with the creation of a new file.

27504 The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group  
27505 ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required  
27506 that implementations provide a way to have the group ID be set to the group ID of the  
27507 containing directory, but did not prohibit implementations also supporting a way to set the  
27508 group ID to the effective group ID of the creating process. Conforming applications should not  
27509 assume which group ID will be used. If it matters, an application can use *chown()* to set the  
27510 group ID after the file is created, or determine under what conditions the implementation will  
27511 set the desired group ID.

#### 27512 FUTURE DIRECTIONS

27513 None.

#### 27514 SEE ALSO

27515 *chmod()*, *close()*, *creat()*, *dup()*, *fcntl()*, *lseek()*, *read()*, *umask()*, *unlockpt()*, *write()*, the Base  
27516 Definitions volume of IEEE Std 1003.1-2001, <fcntl.h>, <sys/stat.h>, <sys/types.h>

27517 **CHANGE HISTORY**

27518 First released in Issue 1. Derived from Issue 1 of the SVID.

27519 **Issue 5**

27520 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
27521 Threads Extension.

27522 Large File Summit extensions are added.

27523 **Issue 6**

27524 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

27525 The following new requirements on POSIX implementations derive from alignment with the  
27526 Single UNIX Specification:

27527 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
27528 required for conforming implementations of previous POSIX specifications, it was not  
27529 required for UNIX applications.

27530 • In the DESCRIPTION, `O_CREAT` is amended to state that the group ID of the file is set to the  
27531 group ID of the file's parent directory or to the effective group ID of the process. This is a  
27532 FIPS requirement.

27533 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file  
27534 description. This change is to support large files.

27535 • In the ERRORS section, the `[E_OVERFLOW]` condition is added. This change is to support  
27536 large files.

27537 • The `[ENXIO]` mandatory error condition is added.

27538 • The `[EINVAL]`, `[ENAMETOOLONG]`, and `[ETXTBSY]` optional error conditions are added.

27539 The DESCRIPTION and ERRORS sections are updated so that items related to the optional XSI  
27540 STREAMS Option Group are marked.

27541 The following changes were made to align with the IEEE P1003.1a draft standard:

27542 • An explanation is added of the effect of the `O_CREAT` and `O_EXCL` flags when the path  
27543 refers to a symbolic link.

27544 • The `[ELOOP]` optional error condition is added.

27545 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

27546 The DESCRIPTION of `O_EXCL` is updated in response to IEEE PASC Interpretation 1003.1c #48.

27547 **NAME**

27548            opendir — open a directory

27549 **SYNOPSIS**

27550            #include &lt;dirent.h&gt;

27551            DIR \*opendir(const char \*dirname);

27552 **DESCRIPTION**

27553            The *opendir()* function shall open a directory stream corresponding to the directory named by  
 27554            the *dirname* argument. The directory stream is positioned at the first entry. If the type **DIR** is  
 27555            implemented using a file descriptor, applications shall only be able to open up to a total of  
 27556            {OPEN\_MAX} files and directories.

27557 **RETURN VALUE**

27558            Upon successful completion, *opendir()* shall return a pointer to an object of type **DIR**.  
 27559            Otherwise, a null pointer shall be returned and *errno* set to indicate the error.

27560 **ERRORS**27561            The *opendir()* function shall fail if:

27562            [EACCES]            Search permission is denied for the component of the path prefix of *dirname* or  
 27563            read permission is denied for *dirname*.

27564            [ELOOP]            A loop exists in symbolic links encountered during resolution of the *dirname*  
 27565            argument.

27566            [ENAMETOOLONG]       The length of the *dirname* argument exceeds {PATH\_MAX} or a pathname  
 27567            component is longer than {NAME\_MAX}.  
 27568

27569            [ENOENT]            A component of *dirname* does not name an existing directory or *dirname* is an  
 27570            empty string.

27571            [ENOTDIR]          A component of *dirname* is not a directory.

27572            The *opendir()* function may fail if:

27573            [ELOOP]            More than {SYMLOOP\_MAX} symbolic links were encountered during  
 27574            resolution of the *dirname* argument.

27575            [EMFILE]            {OPEN\_MAX} file descriptors are currently open in the calling process.

27576            [ENAMETOOLONG]       As a result of encountering a symbolic link in resolution of the *dirname*  
 27577            argument, the length of the substituted pathname string exceeded  
 27578            {PATH\_MAX}.  
 27579

27580            [ENFILE]            Too many files are currently open in the system.

27581 **EXAMPLES**27582 **Open a Directory Stream**

27583 The following program fragment demonstrates how the *opendir()* function is used.

```

27584 #include <sys/types.h>
27585 #include <dirent.h>
27586 #include <libgen.h>
27587 ...
27588 DIR *dir;
27589 struct dirent *dp;
27590 ...
27591 if ((dir = opendir (".")) == NULL) {
27592 perror ("Cannot open .");
27593 exit (1);
27594 }
27595 while ((dp = readdir (dir)) != NULL) {
27596 ...

```

27597 **APPLICATION USAGE**

27598 The *opendir()* function should be used in conjunction with *readdir()*, *closedir()*, and *rewinddir()* to  
 27599 examine the contents of the directory (see the EXAMPLES section in *readdir()*). This method is  
 27600 recommended for portability.

27601 **RATIONALE**

27602 Based on historical implementations, the rules about file descriptors apply to directory streams  
 27603 as well. However, this volume of IEEE Std 1003.1-2001 does not mandate that the directory  
 27604 stream be implemented using file descriptors. The description of *closedir()* clarifies that if a file  
 27605 descriptor is used for the directory stream, it is mandatory that *closedir()* deallocate the file  
 27606 descriptor. When a file descriptor is used to implement the directory stream, it behaves as if the  
 27607 FD\_CLOEXEC had been set for the file descriptor.

27608 The directory entries for dot and dot-dot are optional. This volume of IEEE Std 1003.1-2001 does  
 27609 not provide a way to test *a priori* for their existence because an application that is portable must  
 27610 be written to look for (and usually ignore) those entries. Writing code that presumes that they  
 27611 are the first two entries does not always work, as many implementations permit them to be  
 27612 other than the first two entries, with a “normal” entry preceding them. There is negligible value  
 27613 in providing a way to determine what the implementation does because the code to deal with  
 27614 dot and dot-dot must be written in any case and because such a flag would add to the list of  
 27615 those flags (which has proven in itself to be objectionable) and might be abused.

27616 Since the structure and buffer allocation, if any, for directory operations are defined by the  
 27617 implementation, this volume of IEEE Std 1003.1-2001 imposes no portability requirements for  
 27618 erroneous program constructs, erroneous data, or the use of unspecified values such as the use  
 27619 or referencing of a *dirp* value or a **dirent** structure value after a directory stream has been closed  
 27620 or after a *fork()* or one of the *exec* function calls.

27621 **FUTURE DIRECTIONS**

27622 None.

27623 **SEE ALSO**

27624 *closedir()*, *lstat()*, *readdir()*, *rewinddir()*, *symlink()*, the Base Definitions volume of  
 27625 IEEE Std 1003.1-2001, <dirent.h>, <limits.h>, <sys/types.h>

27626 **CHANGE HISTORY**

27627 First released in Issue 2.

27628 **Issue 6**27629 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.27630 The following new requirements on POSIX implementations derive from alignment with the  
27631 Single UNIX Specification:27632 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
27633 required for conforming implementations of previous POSIX specifications, it was not  
27634 required for UNIX applications.

27635 • The [ELOOP] mandatory error condition is added.

27636 • A second [ENAMETOOLONG] is added as an optional error condition.

27637 The following changes were made to align with the IEEE P1003.1a draft standard:

27638 • The [ELOOP] optional error condition is added.

27639 **NAME**

27640            **openlog** — open a connection to the logging facility

27641 **SYNOPSIS**

27642 xSI        #include <syslog.h>

27643            void openlog(const char \*ident, int logopt, int facility);

27644

27645 **DESCRIPTION**

27646            Refer to *closelog()*.

27647 **NAME**

27648           optarg, opterr, optind, optopt — options parsing variables

27649 **SYNOPSIS**

27650           #include <unistd.h>

27651           extern char \*optarg;

27652           extern int opterr, optind, optopt;

27653 **DESCRIPTION**

27654           Refer to *getopt()*.

27655 **NAME**

27656 pathconf — get configurable pathname variables

27657 **SYNOPSIS**

27658 #include <unistd.h>

27659 long pathconf(const char \*path, int name);

27660 **DESCRIPTION**

27661 Refer to *fpathconf()*.

27662 **NAME**

27663 pause — suspend the thread until a signal is received

27664 **SYNOPSIS**

27665 #include <unistd.h>

27666 int pause(void);

27667 **DESCRIPTION**

27668 The *pause()* function shall suspend the calling thread until delivery of a signal whose action is  
27669 either to execute a signal-catching function or to terminate the process.

27670 If the action is to terminate the process, *pause()* shall not return.

27671 If the action is to execute a signal-catching function, *pause()* shall return after the signal-catching  
27672 function returns.

27673 **RETURN VALUE**

27674 Since *pause()* suspends thread execution indefinitely unless interrupted by a signal, there is no  
27675 successful completion return value. A value of  $-1$  shall be returned and *errno* set to indicate the  
27676 error.

27677 **ERRORS**

27678 The *pause()* function shall fail if:

27679 [EINTR] A signal is caught by the calling process and control is returned from the  
27680 signal-catching function.

27681 **EXAMPLES**

27682 None.

27683 **APPLICATION USAGE**

27684 Many common uses of *pause()* have timing windows. The scenario involves checking a  
27685 condition related to a signal and, if the signal has not occurred, calling *pause()*. When the signal  
27686 occurs between the check and the call to *pause()*, the process often blocks indefinitely. The  
27687 *sigprocmask()* and *sigsuspend()* functions can be used to avoid this type of problem.

27688 **RATIONALE**

27689 None.

27690 **FUTURE DIRECTIONS**

27691 None.

27692 **SEE ALSO**

27693 *sigsuspend()*, the Base Definitions volume of IEEE Std 1003.1-2001, <unistd.h>

27694 **CHANGE HISTORY**

27695 First released in Issue 1. Derived from Issue 1 of the SVID.

27696 **Issue 5**

27697 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

27698 **Issue 6**

27699 The APPLICATION USAGE section is added.

27700 **NAME**

27701 pclose — close a pipe stream to or from a process

27702 **SYNOPSIS**

27703 cx #include &lt;stdio.h&gt;

27704 int pclose(FILE \*stream);

27705

27706 **DESCRIPTION**

27707 The *pclose()* function shall close a stream that was opened by *popen()*, wait for the command to  
 27708 terminate, and return the termination status of the process that was running the command  
 27709 language interpreter. However, if a call caused the termination status to be unavailable to  
 27710 *pclose()*, then *pclose()* shall return  $-1$  with *errno* set to [ECHILD] to report this situation. This can  
 27711 happen if the application calls one of the following functions:

- 27712 • *wait()*
- 27713 • *waitpid()* with a *pid* argument less than or equal to 0 or equal to the process ID of the  
 27714 command line interpreter
- 27715 • Any other function not defined in this volume of IEEE Std 1003.1-2001 that could do one of  
 27716 the above

27717 In any case, *pclose()* shall not return before the child process created by *popen()* has terminated.

27718 If the command language interpreter cannot be executed, the child termination status returned  
 27719 by *pclose()* shall be as if the command language interpreter terminated using *exit(127)* or  
 27720 *\_exit(127)*.

27721 The *pclose()* function shall not affect the termination status of any child of the calling process  
 27722 other than the one created by *popen()* for the associated stream.

27723 If the argument *stream* to *pclose()* is not a pointer to a stream created by *popen()*, the result of  
 27724 *pclose()* is undefined.

27725 **RETURN VALUE**

27726 Upon successful return, *pclose()* shall return the termination status of the command language  
 27727 interpreter. Otherwise, *pclose()* shall return  $-1$  and set *errno* to indicate the error.

27728 **ERRORS**

27729 The *pclose()* function shall fail if:

27730 [ECHILD] The status of the child process could not be obtained, as described above.

27731 **EXAMPLES**

27732 None.

27733 **APPLICATION USAGE**

27734 None.

27735 **RATIONALE**

27736 There is a requirement that *pclose()* not return before the child process terminates. This is  
 27737 intended to disallow implementations that return [EINTR] if a signal is received while waiting.  
 27738 If *pclose()* returned before the child terminated, there would be no way for the application to  
 27739 discover which child used to be associated with the stream, and it could not do the cleanup  
 27740 itself.

27741 If the stream pointed to by *stream* was not created by *popen()*, historical implementations of  
 27742 *pclose()* return  $-1$  without setting *errno*. To avoid requiring *pclose()* to set *errno* in this case,  
 27743 IEEE Std 1003.1-2001 makes the behavior unspecified. An application should not use *pclose()* to

27744 close any stream that was not created by *popen()*.

27745 Some historical implementations of *pclose()* either block or ignore the signals SIGINT, SIGQUIT,  
27746 and SIGHUP while waiting for the child process to terminate. Since this behavior is not  
27747 described for the *pclose()* function in IEEE Std 1003.1-2001, such implementations are not  
27748 conforming. Also, some historical implementations return [EINTR] if a signal is received, even  
27749 though the child process has not terminated. Such implementations are also considered non-  
27750 conforming.

27751 Consider, for example, an application that uses:

```
27752 popen("command", "r")
```

27753 to start *command*, which is part of the same application. The parent writes a prompt to its  
27754 standard output (presumably the terminal) and then reads from the *popen()*ed stream. The child  
27755 reads the response from the user, does some transformation on the response (pathname  
27756 expansion, perhaps) and writes the result to its standard output. The parent process reads the  
27757 result from the pipe, does something with it, and prints another prompt. The cycle repeats.  
27758 Assuming that both processes do appropriate buffer flushing, this would be expected to work.

27759 To conform to IEEE Std 1003.1-2001, *pclose()* must use *waitpid()*, or some similar function,  
27760 instead of *wait()*.

27761 The code sample below illustrates how the *pclose()* function might be implemented on a system  
27762 conforming to IEEE Std 1003.1-2001.

```
27763 int pclose(FILE *stream)
27764 {
27765 int stat;
27766 pid_t pid;
27767
27768 pid = <pid for process created for stream by popen() >
27769 (void) fclose(stream);
27770 while (waitpid(pid, &stat, 0) == -1) {
27771 if (errno != EINTR) {
27772 stat = -1;
27773 break;
27774 }
27775 }
27776 return(stat);
27777 }
```

#### 27777 FUTURE DIRECTIONS

27778 None.

#### 27779 SEE ALSO

27780 *fork()*, *popen()*, *waitpid()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

#### 27781 CHANGE HISTORY

27782 First released in Issue 1. Derived from Issue 1 of the SVID.

27783 **NAME**

27784 perror — write error messages to standard error

27785 **SYNOPSIS**

27786 #include &lt;stdio.h&gt;

27787 void perror(const char \*s);

27788 **DESCRIPTION**

27789 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 27790 conflict between the requirements described here and the ISO C standard is unintentional. This  
 27791 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

27792 The *perror()* function shall map the error number accessed through the symbol *errno* to a  
 27793 language-dependent error message, which shall be written to the standard error stream as  
 27794 follows:

- 27795 • First (if *s* is not a null pointer and the character pointed to by *s* is not the null byte), the string  
 27796 pointed to by *s* followed by a colon and a <space>.
- 27797 • Then an error message string followed by a <newline>.

27798 The contents of the error message strings shall be the same as those returned by *strerror()* with  
 27799 argument *errno*.

27800 cx The *perror()* function shall mark the file associated with the standard error stream as having  
 27801 been written (*st\_ctime*, *st\_mtime* marked for update) at some time between its successful  
 27802 completion and *exit()*, *abort()*, or the completion of *fflush()* or *fclose()* on *stderr*.

27803 The *perror()* function shall not change the orientation of the standard error stream.

27804 **RETURN VALUE**27805 The *perror()* function shall not return a value.27806 **ERRORS**

27807 No errors are defined.

27808 **EXAMPLES**27809 **Printing an Error Message for a Function**

27810 The following example replaces *bufptr* with a buffer that is the necessary size. If an error occurs,  
 27811 the *perror()* function prints a message and the program exits.

```

27812 #include <stdio.h>
27813 #include <stdlib.h>
27814 ...
27815 char *bufptr;
27816 size_t szbuf;
27817 ...
27818 if ((bufptr = malloc(szbuf)) == NULL) {
27819 perror("malloc"); exit(2);
27820 }
27821 ...

```

27822 **APPLICATION USAGE**

27823 None.

27824 **RATIONALE**

27825 None.

27826 **FUTURE DIRECTIONS**

27827 None.

27828 **SEE ALSO**27829 *strerror()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>27830 **CHANGE HISTORY**

27831 First released in Issue 1. Derived from Issue 1 of the SVID.

27832 **Issue 5**27833 A paragraph is added to the DESCRIPTION indicating that *perror()* does not change the  
27834 orientation of the standard error stream.27835 **Issue 6**

27836 Extensions beyond the ISO C standard are marked.

27837 **NAME**

27838 pipe — create an interprocess channel

27839 **SYNOPSIS**

27840 #include &lt;unistd.h&gt;

27841 int pipe(int *filides*[2]);27842 **DESCRIPTION**

27843 The *pipe()* function shall create a pipe and place two file descriptors, one each into the  
 27844 arguments *filides*[0] and *filides*[1], that refer to the open file descriptions for the read and write  
 27845 ends of the pipe. Their integer values shall be the two lowest available at the time of the *pipe()*  
 27846 call. The O\_NONBLOCK and FD\_CLOEXEC flags shall be clear on both file descriptors. (The  
 27847 *fcntl()* function can be used to set both these flags.)

27848 Data can be written to the file descriptor *filides*[1] and read from the file descriptor *filides*[0]. A  
 27849 read on the file descriptor *filides*[0] shall access data written to the file descriptor *filides*[1] on a  
 27850 first-in-first-out basis. It is unspecified whether *filides*[0] is also open for writing and whether  
 27851 *filides*[1] is also open for reading.

27852 A process has the pipe open for reading (correspondingly writing) if it has a file descriptor open  
 27853 that refers to the read end, *filides*[0] (write end, *filides*[1]).

27854 Upon successful completion, *pipe()* shall mark for update the *st\_atime*, *st\_ctime*, and *st\_mtime*  
 27855 fields of the pipe.

27856 **RETURN VALUE**

27857 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
 27858 indicate the error.

27859 **ERRORS**27860 The *pipe()* function shall fail if:

27861 [EMFILE] More than {OPEN\_MAX} minus two file descriptors are already in use by this  
 27862 process.

27863 [ENFILE] The number of simultaneously open files in the system would exceed a  
 27864 system-imposed limit.

27865 **EXAMPLES**

27866 None.

27867 **APPLICATION USAGE**

27868 None.

27869 **RATIONALE**

27870 The wording carefully avoids using the verb “to open” in order to avoid any implication of use  
 27871 of *open()*; see also *write()*.

27872 **FUTURE DIRECTIONS**

27873 None.

27874 **SEE ALSO**27875 *fcntl()*, *read()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, <*fcntl.h*>,  
 27876 <*unistd.h*>27877 **CHANGE HISTORY**

27878 First released in Issue 1. Derived from Issue 1 of the SVID.

27879 **Issue 6**

27880 The following new requirements on POSIX implementations derive from alignment with the  
27881 Single UNIX Specification:

- 27882 • The DESCRIPTION is updated to indicate that certain dispositions of *fildev[0]* and *fildev[1]*  
27883 are unspecified.

## 27884 NAME

27885 poll — input/output multiplexing

## 27886 SYNOPSIS

27887 XSI #include &lt;poll.h&gt;

27888 int poll(struct pollfd *fds*[], nfd\_t *nfds*, int *timeout*);

27889

## 27890 DESCRIPTION

27891 The *poll()* function provides applications with a mechanism for multiplexing input/output over  
 27892 a set of file descriptors. For each member of the array pointed to by *fds*, *poll()* shall examine the  
 27893 given file descriptor for the event(s) specified in *events*. The number of **pollfd** structures in the  
 27894 *fds* array is specified by *nfds*. The *poll()* function shall identify those file descriptors on which an  
 27895 application can read or write data, or on which certain events have occurred.

27896 The *fds* argument specifies the file descriptors to be examined and the events of interest for each  
 27897 file descriptor. It is a pointer to an array with one member for each open file descriptor of  
 27898 interest. The array's members are **pollfd** structures within which *fd* specifies an open file  
 27899 descriptor and *events* and *revents* are bitmasks constructed by OR'ing a combination of the  
 27900 following event flags:

|           |            |                                                                                        |
|-----------|------------|----------------------------------------------------------------------------------------|
| 27901     | POLLIN     | Data other than high-priority data may be read without blocking.                       |
| 27902 XSR |            | For STREAMS, this flag is set in <i>revents</i> even if the message is of zero length. |
| 27903     |            | This flag shall be equivalent to POLLRDNORM   POLLRDBAND.                              |
| 27904     | POLLRDNORM | Normal data may be read without blocking.                                              |
| 27905 XSR |            | For STREAMS, data on priority band 0 may be read without blocking. This                |
| 27906     |            | flag is set in <i>revents</i> even if the message is of zero length.                   |
| 27907     | POLLRDBAND | Priority data may be read without blocking.                                            |
| 27908 XSR |            | For STREAMS, data on priority bands greater than 0 may be read without                 |
| 27909     |            | blocking. This flag is set in <i>revents</i> even if the message is of zero length.    |
| 27910     | POLLPRI    | High-priority data may be read without blocking.                                       |
| 27911 XSR |            | For STREAMS, this flag is set in <i>revents</i> even if the message is of zero length. |
| 27912     | POLLOUT    | Normal data may be written without blocking.                                           |
| 27913 XSR |            | For STREAMS, data on priority band 0 may be written without blocking.                  |
| 27914     | POLLWRNORM | Equivalent to POLLOUT.                                                                 |
| 27915     | POLLWRBAND | Priority data may be written.                                                          |
| 27916 XSR |            | For STREAMS, data on priority bands greater than 0 may be written without              |
| 27917     |            | blocking. If any priority band has been written to on this STREAM, this event          |
| 27918     |            | only examines bands that have been written to at least once.                           |
| 27919     | POLLERR    | An error has occurred on the device or stream. This flag is only valid in the          |
| 27920     |            | <i>revents</i> bitmask; it shall be ignored in the <i>events</i> member.               |
| 27921     | POLLHUP    | The device has been disconnected. This event and POLLOUT are mutually-                 |
| 27922     |            | exclusive; a stream can never be writable if a hangup has occurred. However,           |
| 27923     |            | this event and POLLIN, POLLRDNORM, POLLRDBAND, or POLLPRI are not                      |
| 27924     |            | mutually-exclusive. This flag is only valid in the <i>revents</i> bitmask; it shall be |
| 27925     |            | ignored in the <i>events</i> member.                                                   |

|           |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-----------|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 27926     | POLLNVAL            | The specified <i>fd</i> value is invalid. This flag is only valid in the <i>revents</i> member; it shall be ignored in the <i>events</i> member.                                                                                                                                                                                                                                                                                                                                                                              |
| 27927     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27928     |                     | The significance and semantics of normal, priority, and high-priority data are file and device-specific.                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27929     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27930     |                     | If the value of <i>fd</i> is less than 0, <i>events</i> shall be ignored, and <i>revents</i> shall be set to 0 in that entry on return from <i>poll()</i> .                                                                                                                                                                                                                                                                                                                                                                   |
| 27931     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27932     |                     | In each <b>pollfd</b> structure, <i>poll()</i> shall clear the <i>revents</i> member, except that where the application requested a report on a condition by setting one of the bits of <i>events</i> listed above, <i>poll()</i> shall set the corresponding bit in <i>revents</i> if the requested condition is true. In addition, <i>poll()</i> shall set the POLLHUP, POLLERR, and POLLNVAL flag in <i>revents</i> if the condition is true, even if the application did not set the corresponding bit in <i>events</i> . |
| 27933     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27934     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27935     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27936     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27937     |                     | If none of the defined events have occurred on any selected file descriptor, <i>poll()</i> shall wait at least <i>timeout</i> milliseconds for an event to occur on any of the selected file descriptors. If the value of <i>timeout</i> is 0, <i>poll()</i> shall return immediately. If the value of <i>timeout</i> is -1, <i>poll()</i> shall block until a requested event occurs or until the call is interrupted.                                                                                                       |
| 27938     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27939     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27940     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27941     |                     | Implementations may place limitations on the granularity of timeout intervals. If the requested timeout interval requires a finer granularity than the implementation supports, the actual timeout interval shall be rounded up to the next supported value.                                                                                                                                                                                                                                                                  |
| 27942     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27943     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27944     |                     | The <i>poll()</i> function shall not be affected by the O_NONBLOCK flag.                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 27945     |                     | The <i>poll()</i> function shall support regular files, terminal and pseudo-terminal devices, FIFOs, pipes, sockets and STREAMS-based files. The behavior of <i>poll()</i> on elements of <i>fds</i> that refer to other types of file is unspecified.                                                                                                                                                                                                                                                                        |
| 27946 XSR |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27947     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27948     |                     | Regular files shall always poll TRUE for reading and writing.                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 27949     |                     | A file descriptor for a socket that is listening for connections shall indicate that it is ready for reading, once connections are available. A file descriptor for a socket that is connecting asynchronously shall indicate that it is ready for writing, once a connection has been established.                                                                                                                                                                                                                           |
| 27950     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27951     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27952     | <b>RETURN VALUE</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27953     |                     | Upon successful completion, <i>poll()</i> shall return a non-negative value. A positive value indicates the total number of file descriptors that have been selected (that is, file descriptors for which the <i>revents</i> member is non-zero). A value of 0 indicates that the call timed out and no file descriptors have been selected. Upon failure, <i>poll()</i> shall return -1 and set <i>errno</i> to indicate the error.                                                                                          |
| 27954     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27955     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27956     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27957     | <b>ERRORS</b>       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27958     |                     | The <i>poll()</i> function shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 27959     | [EAGAIN]            | The allocation of internal data structures failed but a subsequent request may succeed.                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 27960     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27961     | [EINTR]             | A signal was caught during <i>poll()</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 27962 XSR | [EINVAL]            | The <i>nfds</i> argument is greater than {OPEN_MAX}, or one of the <i>fd</i> members refers to a STREAM or multiplexer that is linked (directly or indirectly) downstream from a multiplexer.                                                                                                                                                                                                                                                                                                                                 |
| 27963     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 27964     |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

## 27965 EXAMPLES

27966 **Checking for Events on a Stream**

27967 The following example opens a pair of STREAMS devices and then waits for either one to  
 27968 become writable. This example proceeds as follows:

- 27969 1. Sets the *timeout* parameter to 500 milliseconds.
  - 27970 2. Opens the STREAMS devices */dev/dev0* and */dev/dev1*, and then polls them, specifying  
 27971 POLLOUT and POLLWRBAND as the events of interest.
- 27972 The STREAMS device names */dev/dev0* and */dev/dev1* are only examples of how  
 27973 STREAMS devices can be named; STREAMS naming conventions may vary among  
 27974 systems conforming to the IEEE Std 1003.1-2001.
- 27975 3. Uses the *ret* variable to determine whether an event has occurred on either of the two  
 27976 STREAMS. The *poll()* function is given 500 milliseconds to wait for an event to occur (if it  
 27977 has not occurred prior to the *poll()* call).
  - 27978 4. Checks the returned value of *ret*. If a positive value is returned, one of the following can  
 27979 be done:
    - 27980 a. Priority data can be written to the open STREAM on priority bands greater than 0,  
 27981 because the POLLWRBAND event occurred on the open STREAM (*fds[0]* or *fds[1]*).
    - 27982 b. Data can be written to the open STREAM on priority-band 0, because the POLLOUT  
 27983 event occurred on the open STREAM (*fds[0]* or *fds[1]*).
  - 27984 5. If the returned value is not a positive value, permission to write data to the open STREAM  
 27985 (on any priority band) is denied.
  - 27986 6. If the POLLHUP event occurs on the open STREAM (*fds[0]* or *fds[1]*), the device on the  
 27987 open STREAM has disconnected.

```

27988 #include <stropts.h>
27989 #include <poll.h>
27990 ...
27991 struct pollfd fds[2];
27992 int timeout_msecs = 500;
27993 int ret;
27994 int i;

27995 /* Open STREAMS device. */
27996 fds[0].fd = open("/dev/dev0", ...);
27997 fds[1].fd = open("/dev/dev1", ...);
27998 fds[0].events = POLLOUT | POLLWRBAND;
27999 fds[1].events = POLLOUT | POLLWRBAND;

28000 ret = poll(fds, 2, timeout_msecs);

28001 if (ret > 0) {
28002 /* An event on one of the fds has occurred. */
28003 for (i=0; i<2; i++) {
28004 if (fds[i].revents & POLLWRBAND) {
28005 /* Priority data may be written on device number i. */
28006 ...
28007 }
28008 if (fds[i].revents & POLLOUT) {

```

```
28009 /* Data may be written on device number i. */
28010 ...
28011 }
28012 if (fds[i].revents & POLLHUP) {
28013 /* A hangup has occurred on device number i. */
28014 ...
28015 }
28016 }
28017 }
```

**28018 APPLICATION USAGE**

28019 None.

**28020 RATIONALE**

28021 None.

**28022 FUTURE DIRECTIONS**

28023 None.

**28024 SEE ALSO**

28025 Section 2.6 (on page 38), *getmsg()*, *putmsg()*, *read()*, *select()*, *write()*, the Base Definitions volume  
28026 of IEEE Std 1003.1-2001, <**poll.h**>, <**stropts.h**>

**28027 CHANGE HISTORY**

28028 First released in Issue 4, Version 2.

**28029 Issue 5**

28030 Moved from X/OPEN UNIX extension to BASE.

28031 The description of POLLWRBAND is updated.

**28032 Issue 6**

28033 Text referring to sockets is added to the DESCRIPTION.

28034 Text relating to the XSI STREAMS Option Group is marked.

28035 The Open Group Corrigendum U055/3 is applied, updating the DESCRIPTION of  
28036 POLLWRBAND.

28037 **NAME**

28038 popen — initiate pipe streams to or from a process

28039 **SYNOPSIS**28040 cx `#include <stdio.h>`28041 `FILE *popen(const char *command, const char *mode);`

28042

28043 **DESCRIPTION**

28044 The *popen()* function shall execute the command specified by the string *command*. It shall create a  
 28045 pipe between the calling program and the executed command, and shall return a pointer to a  
 28046 stream that can be used to either read from or write to the pipe.

28047 The environment of the executed command shall be as if a child process were created within the  
 28048 *popen()* call using the *fork()* function, and the child invoked the *sh* utility using the call:

28049 `execl(shell_path, "sh", "-c", command, (char *)0);`28050 where *shell\_path* is an unspecified pathname for the *sh* utility.

28051 The *popen()* function shall ensure that any streams from previous *popen()* calls that remain open  
 28052 in the parent process are closed in the new child process.

28053 The *mode* argument to *popen()* is a string that specifies I/O mode:

- 28054 1. If *mode* is *r*, when the child process is started, its file descriptor `STDOUT_FILENO` shall be  
 28055 the writable end of the pipe, and the file descriptor *fileno(stream)* in the calling process,  
 28056 where *stream* is the stream pointer returned by *popen()*, shall be the readable end of the  
 28057 pipe.
- 28058 2. If *mode* is *w*, when the child process is started its file descriptor `STDIN_FILENO` shall be  
 28059 the readable end of the pipe, and the file descriptor *fileno(stream)* in the calling process,  
 28060 where *stream* is the stream pointer returned by *popen()*, shall be the writable end of the  
 28061 pipe.
- 28062 3. If *mode* is any other value, the result is undefined.

28063 After *popen()*, both the parent and the child process shall be capable of executing independently  
 28064 before either terminates.

28065 Pipe streams are byte-oriented.

28066 **RETURN VALUE**

28067 Upon successful completion, *popen()* shall return a pointer to an open stream that can be used to  
 28068 read or write to the pipe. Otherwise, it shall return a null pointer and may set *errno* to indicate  
 28069 the error.

28070 **ERRORS**28071 The *popen()* function may fail if:

28072 [EMFILE] {FOPEN\_MAX} or {STREAM\_MAX} streams are currently open in the calling  
 28073 process.

28074 [EINVAL] The *mode* argument is invalid.28075 The *popen()* function may also set *errno* values as described by *fork()* or *pipe()*.

28076 **EXAMPLES**

28077       None.

28078 **APPLICATION USAGE**28079       Since open files are shared, a mode *r* command can be used as an input filter and a mode *w*  
28080       command as an output filter.28081       Buffered reading before opening an input filter may leave the standard input of that filter  
28082       mispositioned. Similar problems with an output filter may be prevented by careful buffer  
28083       flushing; for example, with *fflush()*.28084       A stream opened by *popen()* should be closed by *pclose()*.28085       The behavior of *popen()* is specified for values of *mode* of *r* and *w*. Other modes such as *rb* and  
28086       *wb* might be supported by specific implementations, but these would not be portable features.  
28087       Note that historical implementations of *popen()* only check to see if the first character of *mode* is  
28088       *r*. Thus, a *mode* of *robert the robot* would be treated as *mode r*, and a *mode* of *anything else* would be  
28089       treated as *mode w*.28090       If the application calls *waitpid()* or *waitid()* with a *pid* argument greater than 0, and it still has a  
28091       stream that was called with *popen()* open, it must ensure that *pid* does not refer to the process  
28092       started by *popen()*.28093       To determine whether or not the environment specified in the Shell and Utilities volume of  
28094       IEEE Std 1003.1-2001 is present, use the function call:

28095       sysconf(\_SC\_2\_VERSION)

28096       (See *sysconf()*).28097 **RATIONALE**28098       The *popen()* function should not be used by programs that have set user (or group) ID privileges.  
28099       The *fork()* and *exec* family of functions (except *execlp()* and *execvp()*), should be used instead.  
28100       This prevents any unforeseen manipulation of the environment of the user that could cause  
28101       execution of commands not anticipated by the calling program.28102       If the original and *popen()*ed processes both intend to read or write or read and write a common  
28103       file, and either will be using FILE-type C functions (*fread()*, *fwrite()*, and so on), the rules for  
28104       sharing file handles must be observed (see Section 2.5.1 (on page 35)).28105 **FUTURE DIRECTIONS**

28106       None.

28107 **SEE ALSO**28108       *pclose()*, *pipe()*, *sysconf()*, *system()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
28109       <**stdio.h**>, the Shell and Utilities volume of IEEE Std 1003.1-2001, *sh*28110 **CHANGE HISTORY**

28111       First released in Issue 1. Derived from Issue 1 of the SVID.

28112 **Issue 5**

28113       A statement is added to the DESCRIPTION indicating that pipe streams are byte-oriented.

28114 **Issue 6**28115       The following new requirements on POSIX implementations derive from alignment with the  
28116       Single UNIX Specification:

- 28117
- The optional [EMFILE] error condition is added.

28118 **NAME**

28119 posix\_fadvise — file advisory information (**ADVANCED REALTIME**)

28120 **SYNOPSIS**

28121 ADV #include <fcntl.h>

28122 int posix\_fadvise(int fd, off\_t offset, size\_t len, int advice);

28123

28124 **DESCRIPTION**

28125 The *posix\_fadvise()* function shall advise the implementation on the expected behavior of the  
 28126 application with respect to the data in the file associated with the open file descriptor, *fd*,  
 28127 starting at *offset* and continuing for *len* bytes. The specified range need not currently exist in the  
 28128 file. If *len* is zero, all data following *offset* is specified. The implementation may use this  
 28129 information to optimize handling of the specified data. The *posix\_fadvise()* function shall have no  
 28130 effect on the semantics of other operations on the specified data, although it may affect the  
 28131 performance of other operations.

28132 The advice to be applied to the data is specified by the *advice* parameter and may be one of the  
 28133 following values:

28134 POSIX\_FADV\_NORMAL

28135 Specifies that the application has no advice to give on its behavior with respect to the  
 28136 specified data. It is the default characteristic if no advice is given for an open file.

28137 POSIX\_FADV\_SEQUENTIAL

28138 Specifies that the application expects to access the specified data sequentially from lower  
 28139 offsets to higher offsets.

28140 POSIX\_FADV\_RANDOM

28141 Specifies that the application expects to access the specified data in a random order.

28142 POSIX\_FADV\_WILLNEED

28143 Specifies that the application expects to access the specified data in the near future.

28144 POSIX\_FADV\_DONTNEED

28145 Specifies that the application expects that it will not access the specified data in the near  
 28146 future.

28147 POSIX\_FADV\_NOREUSE

28148 Specifies that the application expects to access the specified data once and then not reuse it  
 28149 thereafter.

28150 These values are defined in <fcntl.h>.

28151 **RETURN VALUE**

28152 Upon successful completion, *posix\_fadvise()* shall return zero; otherwise, an error number shall  
 28153 be returned to indicate the error.

28154 **ERRORS**

28155 The *posix\_fadvise()* function shall fail if:

28156 [EBADF] The *fd* argument is not a valid file descriptor.

28157 [EINVAL] The value of *advice* is invalid.

28158 [ESPIPE] The *fd* argument is associated with a pipe or FIFO.

28159 **EXAMPLES**

28160           None.

28161 **APPLICATION USAGE**

28162           The *posix\_fadvise()* function is part of the Advisory Information option and need not be provided  
28163           on all implementations.

28164 **RATIONALE**

28165           None.

28166 **FUTURE DIRECTIONS**

28167           None.

28168 **SEE ALSO**28169           *posix\_madvise()*, the Base Definitions volume of IEEE Std 1003.1-2001, <fcntl.h>28170 **CHANGE HISTORY**

28171           First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28172           In the SYNOPSIS, the inclusion of &lt;sys/types.h&gt; is no longer required.

28173 **NAME**

28174 posix\_fallocate — file space control (**ADVANCED REALTIME**)

28175 **SYNOPSIS**

28176 ADV `#include <fcntl.h>`

28177 `int posix_fallocate(int fd, off_t offset, size_t len);`

28178

28179 **DESCRIPTION**

28180 The *posix\_fallocate()* function shall ensure that any required storage for regular file data starting  
 28181 at *offset* and continuing for *len* bytes is allocated on the file system storage media. If  
 28182 *posix\_fallocate()* returns successfully, subsequent writes to the specified file data shall not fail  
 28183 due to the lack of free space on the file system storage media.

28184 If the *offset+len* is beyond the current file size, then *posix\_fallocate()* shall adjust the file size to  
 28185 *offset+len*. Otherwise, the file size shall not be changed.

28186 It is implementation-defined whether a previous *posix\_fadvise()* call influences allocation  
 28187 strategy.

28188 Space allocated via *posix\_fallocate()* shall be freed by a successful call to *creat()* or *open()* that  
 28189 truncates the size of the file. Space allocated via *posix\_fallocate()* may be freed by a successful call  
 28190 to *ftruncate()* that reduces the file size to a size smaller than *offset+len*.

28191 **RETURN VALUE**

28192 Upon successful completion, *posix\_fallocate()* shall return zero; otherwise, an error number shall  
 28193 be returned to indicate the error.

28194 **ERRORS**

28195 The *posix\_fallocate()* function shall fail if:

- 28196 [EBADF] The *fd* argument is not a valid file descriptor.
- 28197 [EBADF] The *fd* argument references a file that was opened without write permission.
- 28198 [EFBIG] The value of *offset+len* is greater than the maximum file size.
- 28199 [EINTR] A signal was caught during execution.
- 28200 [EINVAL] The *len* argument was zero or the *offset* argument was less than zero.
- 28201 [EIO] An I/O error occurred while reading from or writing to a file system.
- 28202 [ENODEV] The *fd* argument does not refer to a regular file.
- 28203 [ENOSPC] There is insufficient free space remaining on the file system storage media.
- 28204 [ESPIPE] The *fd* argument is associated with a pipe or FIFO.

28205 **EXAMPLES**

28206 None.

28207 **APPLICATION USAGE**

28208 The *posix\_fallocate()* function is part of the Advisory Information option and need not be  
 28209 provided on all implementations.

28210 **RATIONALE**

28211 None.

28212 **FUTURE DIRECTIONS**

28213       None.

28214 **SEE ALSO**28215       *creat()*, *truncate()*, *open()*, *unlink()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
28216       <fcntl.h>28217 **CHANGE HISTORY**

28218       First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28219       In the SYNOPSIS, the inclusion of &lt;sys/types.h&gt; is no longer required.

28220 **NAME**

28221 posix\_madvise — memory advisory information and alignment control (**ADVANCED**  
28222 **REALTIME**)

28223 **SYNOPSIS**

28224 ADV #include <sys/mman.h>

28225 int posix\_madvise(void \*addr, size\_t len, int advice);  
28226

28227 **DESCRIPTION**

28228 MF|SHM The *posix\_madvise()* function need only be supported if either the Memory Mapped Files or the  
28229 Shared Memory Objects options are supported.

28230 The *posix\_madvise()* function shall advise the implementation on the expected behavior of the  
28231 application with respect to the data in the memory starting at address *addr*, and continuing for  
28232 *len* bytes. The implementation may use this information to optimize handling of the specified  
28233 data. The *posix\_madvise()* function shall have no effect on the semantics of access to memory in  
28234 the specified range, although it may affect the performance of access.

28235 The implementation may require that *addr* be a multiple of the page size, which is the value  
28236 returned by *sysconf()* when the name value *\_SC\_PAGESIZE* is used.

28237 The advice to be applied to the memory range is specified by the *advice* parameter and may be  
28238 one of the following values:

28239 POSIX\_MADV\_NORMAL

28240 Specifies that the application has no advice to give on its behavior with respect to the  
28241 specified range. It is the default characteristic if no advice is given for a range of memory.

28242 POSIX\_MADV\_SEQUENTIAL

28243 Specifies that the application expects to access the specified range sequentially from lower  
28244 addresses to higher addresses.

28245 POSIX\_MADV\_RANDOM

28246 Specifies that the application expects to access the specified range in a random order.

28247 POSIX\_MADV\_WILLNEED

28248 Specifies that the application expects to access the specified range in the near future.

28249 POSIX\_MADV\_DONTNEED

28250 Specifies that the application expects that it will not access the specified range in the near  
28251 future.

28252 These values are defined in the <sys/mman.h> header.

28253 **RETURN VALUE**

28254 Upon successful completion, *posix\_madvise()* shall return zero; otherwise, an error number shall  
28255 be returned to indicate the error.

28256 **ERRORS**

28257 The *posix\_madvise()* function shall fail if:

28258 [EINVAL] The value of *advice* is invalid.

28259 [ENOMEM] Addresses in the range starting at *addr* and continuing for *len* bytes are partly  
28260 or completely outside the range allowed for the address space of the calling  
28261 process.

28262 The *posix\_madvise()* function may fail if:

28263 [EINVAL] The value of *addr* is not a multiple of the value returned by *sysconf()* when the  
28264 name value *\_SC\_PAGESIZE* is used.

28265 [EINVAL] The value of *len* is zero.

28266 **EXAMPLES**

28267 None.

28268 **APPLICATION USAGE**

28269 The *posix\_madvise()* function is part of the Advisory Information option and need not be  
28270 provided on all implementations.

28271 **RATIONALE**

28272 None.

28273 **FUTURE DIRECTIONS**

28274 None.

28275 **SEE ALSO**

28276 *mmap()*, *posix\_fadvise()*, *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
28277 <**sys/mman.h**>

28278 **CHANGE HISTORY**

28279 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28280 IEEE PASC Interpretation 1003.1 #102 is applied.

28281 **NAME**

28282 posix\_mem\_offset — find offset and length of a mapped typed memory block (**ADVANCED**  
28283 **REALTIME**)

28284 **SYNOPSIS**

28285 TYM #include <sys/mman.h>

```
28286 int posix_mem_offset(const void *restrict addr, size_t len,
28287 off_t *restrict off, size_t *restrict contig_len,
28288 int *restrict fildes);
28289
```

28290 **DESCRIPTION**

28291 The *posix\_mem\_offset()* function shall return in the variable pointed to by *off* a value that  
28292 identifies the offset (or location), within a memory object, of the memory block currently  
28293 mapped at *addr*. The function shall return in the variable pointed to by *fildes*, the descriptor used  
28294 (via *mmap()*) to establish the mapping which contains *addr*. If that descriptor was closed since  
28295 the mapping was established, the returned value of *fildes* shall be  $-1$ . The *len* argument specifies  
28296 the length of the block of the memory object the user wishes the offset for; upon return, the value  
28297 pointed to by *contig\_len* shall equal either *len*, or the length of the largest contiguous block of the  
28298 memory object that is currently mapped to the calling process starting at *addr*, whichever is  
28299 smaller.

28300 If the memory object mapped at *addr* is a typed memory object, then if the *off* and *contig\_len*  
28301 values obtained by calling *posix\_mem\_offset()* are used in a call to *mmap()* with a file descriptor  
28302 that refers to the same memory pool as *fildes* (either through the same port or through a different  
28303 port), and that was opened with neither the `POSIX_TYPED_MEM_ALLOCATE` nor the  
28304 `POSIX_TYPED_MEM_ALLOCATE_CONTIG` flag, the typed memory area that is mapped shall  
28305 be exactly the same area that was mapped at *addr* in the address space of the process that called  
28306 *posix\_mem\_offset()*.

28307 If the memory object specified by *fildes* is not a typed memory object, then the behavior of this  
28308 function is implementation-defined.

28309 **RETURN VALUE**

28310 Upon successful completion, the *posix\_mem\_offset()* function shall return zero; otherwise, the  
28311 corresponding error status value shall be returned.

28312 **ERRORS**

28313 The *posix\_mem\_offset()* function shall fail if:

28314 [EACCES] The process has not mapped a memory object supported by this function at  
28315 the given address *addr*.

28316 This function shall not return an error code of [EINTR].

28317 **EXAMPLES**

28318 None.

28319 **APPLICATION USAGE**

28320 None.

28321 **RATIONALE**

28322 None.

28323 **FUTURE DIRECTIONS**

28324           None.

28325 **SEE ALSO**

28326           *mmap()*, *posix\_typed\_mem\_open()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
28327           <**sys/mman.h**>

28328 **CHANGE HISTORY**

28329           First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

## 28330 NAME

28331 posix\_memalign — aligned memory allocation (**ADVANCED REALTIME**)

## 28332 SYNOPSIS

28333 ADV `#include <stdlib.h>`

28334 `int posix_memalign(void **memptr, size_t alignment, size_t size);`

28335

## 28336 DESCRIPTION

28337 The *posix\_memalign()* function shall allocate *size* bytes aligned on a boundary specified by  
 28338 *alignment*, and shall return a pointer to the allocated memory in *memptr*. The value of *alignment*  
 28339 shall be a multiple of *sizeof(void \*)*, that is also a power of two. Upon successful completion, the  
 28340 value pointed to by *memptr* shall be a multiple of *alignment*.

28341 CX The *free()* function shall deallocate memory that has previously been allocated by  
 28342 *posix\_memalign()*.

## 28343 RETURN VALUE

28344 Upon successful completion, *posix\_memalign()* shall return zero; otherwise, an error number  
 28345 shall be returned to indicate the error.

## 28346 ERRORS

28347 The *posix\_memalign()* function shall fail if:

28348 [EINVAL] The value of the alignment parameter is not a power of two multiple of  
 28349 *sizeof(void \*)*.

28350 [ENOMEM] There is insufficient memory available with the requested alignment.

## 28351 EXAMPLES

28352 None.

## 28353 APPLICATION USAGE

28354 The *posix\_memalign()* function is part of the Advisory Information option and need not be  
 28355 provided on all implementations.

## 28356 RATIONALE

28357 None.

## 28358 FUTURE DIRECTIONS

28359 None.

## 28360 SEE ALSO

28361 *free()*, *malloc()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<stdlib.h>`

## 28362 CHANGE HISTORY

28363 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28364 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

28365 **NAME**

28366        posix\_openpt — open a pseudo-terminal device

28367 **SYNOPSIS**

28368 XSI       #include &lt;stdlib.h&gt;

28369       #include &lt;fcntl.h&gt;

28370       int posix\_openpt(int oflag);

28371

28372 **DESCRIPTION**

28373        The *posix\_openpt()* function shall establish a connection between a master device for a pseudo-terminal and a file descriptor. The file descriptor is used by other I/O functions that refer to that pseudo-terminal.

28376        The file status flags and file access modes of the open file description shall be set according to the value of *oflag*.

28378        Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in <fcntl.h>:

28380        O\_RDWR        Open for reading and writing.

28381        O\_NOCTTY       If set *posix\_openpt()* shall not cause the terminal device to become the controlling terminal for the process.

28383        The behavior of other values for the *oflag* argument is unspecified.

28384 **RETURN VALUE**

28385        Upon successful completion, the *posix\_openpt()* function shall open a master pseudo-terminal device and return a non-negative integer representing the lowest numbered unused file descriptor. Otherwise, -1 shall be returned and *errno* set to indicate the error.

28388 **ERRORS**

28389        The *posix\_openpt()* function shall fail if:

28390        [EMFILE]       {OPEN\_MAX} file descriptors are currently open in the calling process.

28391        [ENFILE]       The maximum allowable number of files is currently open in the system.

28392        The *posix\_openpt()* function may fail if:

28393        [EINVAL]       The value of *oflag* is not valid.

28394        [EAGAIN]       Out of pseudo-terminal resources.

28395 XSR       [ENOSR]       Out of STREAMS resources.

28396 **EXAMPLES**

28397        **Opening a Pseudo-Terminal and Returning the Name of the Slave Device and a File Descriptor**

28399        #include <fcntl.h>

28400        #include <stdio.h>

28401        int masterfd, slavefd;

28402        char \*slavedevice;

28403        masterfd = posix\_openpt(O\_RDWR|O\_NOCTTY);

28404        if (masterfd == -1

28405            || grantpt(masterfd) == -1

```

28406 || unlockpt (masterfd) == -1
28407 || (slavedevice = ptsname (masterfd)) == NULL)
28408 return -1;

28409 printf("slave device is: %s\n", slavedevice);

28410 slavefd = open(slave, O_RDWR|O_NOCTTY);
28411 if (slavefd < 0)
28412 return -1;

```

## 28413 APPLICATION USAGE

28414 This function is a method for portably obtaining a file descriptor of a master terminal device for  
 28415 a pseudo-terminal. The *grantpt()* and *ptsname()* functions can be used to manipulate mode and  
 28416 ownership permissions, and to obtain the name of the slave device, respectively.

## 28417 RATIONALE

28418 The standard developers considered the matter of adding a special device for cloning master  
 28419 pseudo-terminals: the */dev/ptmx* device. However, consensus could not be reached, and it was  
 28420 felt that adding a new function would permit other implementations. The *posix\_openpt()*  
 28421 function is designed to complement the *grantpt()*, *ptsname()*, and *unlockpt()* functions.

28422 On implementations supporting the */dev/ptmx* clone device, opening the master device of a  
 28423 pseudo-terminal is simply:

```

28424 mfdp = open("/dev/ptmx", oflag);
28425 if (mfdp < 0)
28426 return -1;

```

## 28427 FUTURE DIRECTIONS

28428 None.

## 28429 SEE ALSO

28430 *grantpt()*, *open()*, *ptsname()*, *unlockpt()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
 28431 *<fcntl.h>*

## 28432 CHANGE HISTORY

28433 First released in Issue 6.

28434 **NAME**28435 posix\_spawn, posix\_spawnnp — spawn a process (**ADVANCED REALTIME**)28436 **SYNOPSIS**28437 SPN 

```
#include <spawn.h>
```

```
28438 int posix_spawn(pid_t *restrict pid, const char *restrict path,
28439 const posix_spawn_file_actions_t *file_actions,
28440 const posix_spawnattr_t *restrict attrp,
28441 char *const argv[restrict], char *const envp[restrict]);
28442 int posix_spawnnp(pid_t *restrict pid, const char *restrict file,
28443 const posix_spawn_file_actions_t *file_actions,
28444 const posix_spawnattr_t *restrict attrp,
28445 char *const argv[restrict], char * const envp[restrict]);
28446
```

28447 **DESCRIPTION**

28448 The *posix\_spawn()* and *posix\_spawnnp()* functions shall create a new process (child process) from  
 28449 the specified process image. The new process image shall be constructed from a regular  
 28450 executable file called the new process image file.

28451 When a C program is executed as the result of this call, it shall be entered as a C-language  
 28452 function call as follows:

```
28453 int main(int argc, char *argv[]);
```

28454 where *argc* is the argument count and *argv* is an array of character pointers to the arguments  
 28455 themselves. In addition, the following variable:

```
28456 extern char **environ;
```

28457 shall be initialized as a pointer to an array of character pointers to the environment strings.

28458 The argument *argv* is an array of character pointers to null-terminated strings. The last member  
 28459 of this array shall be a null pointer and is not counted in *argc*. These strings constitute the  
 28460 argument list available to the new process image. The value in *argv*[0] should point to a filename  
 28461 that is associated with the process image being started by the *posix\_spawn()* or *posix\_spawnnp()*  
 28462 function.

28463 The argument *envp* is an array of character pointers to null-terminated strings. These strings  
 28464 constitute the environment for the new process image. The environment array is terminated by a  
 28465 null pointer.

28466 The number of bytes available for the child process' combined argument and environment lists  
 28467 is {ARG\_MAX}. The implementation shall specify in the system documentation (see the Base  
 28468 Definitions volume of IEEE Std 1003.1-2001, Chapter 2, Conformance) whether any list  
 28469 overhead, such as length words, null terminators, pointers, or alignment bytes, is included in  
 28470 this total.

28471 The *path* argument to *posix\_spawn()* is a pathname that identifies the new process image file to  
 28472 execute.

28473 The *file* parameter to *posix\_spawnnp()* shall be used to construct a pathname that identifies the  
 28474 new process image file. If the *file* parameter contains a slash character, the *file* parameter shall be  
 28475 used as the pathname for the new process image file. Otherwise, the path prefix for this file shall  
 28476 be obtained by a search of the directories passed as the environment variable *PATH* (see the Base  
 28477 Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables). If this  
 28478 environment variable is not defined, the results of the search are implementation-defined.

28479 If *file\_actions* is a null pointer, then file descriptors open in the calling process shall remain open  
 28480 in the child process, except for those whose close-on-exec flag FD\_CLOEXEC is set (see *fcntl()*).  
 28481 For those file descriptors that remain open, all attributes of the corresponding open file  
 28482 descriptions, including file locks (see *fcntl()*), shall remain unchanged.

28483 If *file\_actions* is not NULL, then the file descriptors open in the child process shall be those open  
 28484 in the calling process as modified by the spawn file actions object pointed to by *file\_actions* and  
 28485 the FD\_CLOEXEC flag of each remaining open file descriptor after the spawn file actions have  
 28486 been processed. The effective order of processing the spawn file actions shall be:

- 28487 1. The set of open file descriptors for the child process shall initially be the same set as is  
 28488 open for the calling process. All attributes of the corresponding open file descriptions,  
 28489 including file locks (see *fcntl()*), shall remain unchanged.
- 28490 2. The signal mask, signal default actions, and the effective user and group IDs for the child  
 28491 process shall be changed as specified in the attributes object referenced by *attrp*.
- 28492 3. The file actions specified by the spawn file actions object shall be performed in the order in  
 28493 which they were added to the spawn file actions object.
- 28494 4. Any file descriptor that has its FD\_CLOEXEC flag set (see *fcntl()*) shall be closed.

28495 The **posix\_spawnattr\_t** spawn attributes object type is defined in **<spawn.h>**. It shall contain at  
 28496 least the attributes defined below.

28497 If the POSIX\_SPAWN\_SETPGROUP flag is set in the *spawn-flags* attribute of the object  
 28498 referenced by *attrp*, and the *spawn-pgroup* attribute of the same object is non-zero, then the  
 28499 child's process group shall be as specified in the *spawn-pgroup* attribute of the object referenced  
 28500 by *attrp*.

28501 As a special case, if the POSIX\_SPAWN\_SETPGROUP flag is set in the *spawn-flags* attribute of  
 28502 the object referenced by *attrp*, and the *spawn-pgroup* attribute of the same object is set to zero,  
 28503 then the child shall be in a new process group with a process group ID equal to its process ID.

28504 If the POSIX\_SPAWN\_SETPGROUP flag is not set in the *spawn-flags* attribute of the object  
 28505 referenced by *attrp*, the new child process shall inherit the parent's process group.

28506 PS If the POSIX\_SPAWN\_SETSCHEDPARAM flag is set in the *spawn-flags* attribute of the object  
 28507 referenced by *attrp*, but POSIX\_SPAWN\_SETSCHEDULER is not set, the new process image  
 28508 shall initially have the scheduling policy of the calling process with the scheduling parameters  
 28509 specified in the *spawn-schedparam* attribute of the object referenced by *attrp*.

28510 If the POSIX\_SPAWN\_SETSCHEDULER flag is set in the *spawn-flags* attribute of the object  
 28511 referenced by *attrp* (regardless of the setting of the POSIX\_SPAWN\_SETSCHEDPARAM flag),  
 28512 the new process image shall initially have the scheduling policy specified in the *spawn-*  
 28513 *schedpolicy* attribute of the object referenced by *attrp* and the scheduling parameters specified in  
 28514 the *spawn-schedparam* attribute of the same object.

28515 The POSIX\_SPAWN\_RESETEUIDS flag in the *spawn-flags* attribute of the object referenced by *attrp*  
 28516 governs the effective user ID of the child process. If this flag is not set, the child process shall  
 28517 inherit the parent process' effective user ID. If this flag is set, the child process' effective user ID  
 28518 shall be reset to the parent's real user ID. In either case, if the set-user-ID mode bit of the new  
 28519 process image file is set, the effective user ID of the child process shall become that file's owner  
 28520 ID before the new process image begins execution.

28521 The POSIX\_SPAWN\_RESETEUIDS flag in the *spawn-flags* attribute of the object referenced by *attrp*  
 28522 also governs the effective group ID of the child process. If this flag is not set, the child process  
 28523 shall inherit the parent process' effective group ID. If this flag is set, the child process' effective  
 28524 group ID shall be reset to the parent's real group ID. In either case, if the set-group-ID mode bit

28525 of the new process image file is set, the effective group ID of the child process shall become that  
28526 file's group ID before the new process image begins execution.

28527 If the POSIX\_SPAWN\_SETSIGMASK flag is set in the *spawn\_flags* attribute of the object  
28528 referenced by *attrp*, the child process shall initially have the signal mask specified in the *spawn-*  
28529 *sigmask* attribute of the object referenced by *attrp*.

28530 If the POSIX\_SPAWN\_SETSIGDEF flag is set in the *spawn\_flags* attribute of the object referenced  
28531 by *attrp*, the signals specified in the *spawn\_sigdefault* attribute of the same object shall be set to  
28532 their default actions in the child process. Signals set to the default action in the parent process  
28533 shall be set to the default action in the child process.

28534 Signals set to be caught by the calling process shall be set to the default action in the child  
28535 process.

28536 Except for SIGCHLD, signals set to be ignored by the calling process image shall be set to be  
28537 ignored by the child process, unless otherwise specified by the POSIX\_SPAWN\_SETSIGDEF flag  
28538 being set in the *spawn\_flags* attribute of the object referenced by *attrp* and the signals being  
28539 indicated in the *spawn\_sigdefault* attribute of the object referenced by *attrp*.

28540 If the SIGCHLD signal is set to be ignored by the calling process, it is unspecified whether the  
28541 SIGCHLD signal is set to be ignored or to the default action in the child process, unless  
28542 otherwise specified by the POSIX\_SPAWN\_SETSIGDEF flag being set in the *spawn\_flags*  
28543 attribute of the object referenced by *attrp* and the SIGCHLD signal being indicated in the  
28544 *spawn\_sigdefault* attribute of the object referenced by *attrp*.

28545 If the value of the *attrp* pointer is NULL, then the default values are used.

28546 All process attributes, other than those influenced by the attributes set in the object referenced  
28547 by *attrp* as specified above or by the file descriptor manipulations specified in *file\_actions*, shall  
28548 appear in the new process image as though *fork()* had been called to create a child process and  
28549 then a member of the *exec* family of functions had been called by the child process to execute the  
28550 new process image.

28551 THR It is implementation-defined whether the fork handlers are run when *posix\_spawn()* or  
28552 *posix\_spawnnp()* is called.

#### 28553 RETURN VALUE

28554 Upon successful completion, *posix\_spawn()* and *posix\_spawnnp()* shall return the process ID of the  
28555 child process to the parent process, in the variable pointed to by a non-NULL *pid* argument, and  
28556 shall return zero as the function return value. Otherwise, no child process shall be created, the  
28557 value stored into the variable pointed to by a non-NULL *pid* is unspecified, and an error number  
28558 shall be returned as the function return value to indicate the error. If the *pid* argument is a null  
28559 pointer, the process ID of the child is not returned to the caller.

#### 28560 ERRORS

28561 The *posix\_spawn()* and *posix\_spawnnp()* functions may fail if:

28562 [EINVAL] The value specified by *file\_actions* or *attrp* is invalid.

28563 If this error occurs after the calling process successfully returns from the *posix\_spawn()* or  
28564 *posix\_spawnnp()* function, the child process may exit with exit status 127.

28565 If *posix\_spawn()* or *posix\_spawnnp()* fail for any of the reasons that would cause *fork()* or one of  
28566 the *exec* family of functions to fail, an error value shall be returned as described by *fork()* and  
28567 *exec*, respectively (or, if the error occurs after the calling process successfully returns, the child  
28568 process shall exit with exit status 127).

28569 If POSIX\_SPAWN\_SETPGROUP is set in the *spawn-flags* attribute of the object referenced by  
 28570 *attrp*, and *posix\_spawn()* or *posix\_spawnnp()* fails while changing the child's process group, an  
 28571 error value shall be returned as described by *setpgid()* (or, if the error occurs after the calling  
 28572 process successfully returns, the child process shall exit with exit status 127).

28573 PS If POSIX\_SPAWN\_SETSCHEDPARAM is set and POSIX\_SPAWN\_SETSCHEDULER is not set  
 28574 in the *spawn-flags* attribute of the object referenced by *attrp*, then if *posix\_spawn()* or  
 28575 *posix\_spawnnp()* fails for any of the reasons that would cause *sched\_setparam()* to fail, an error  
 28576 value shall be returned as described by *sched\_setparam()* (or, if the error occurs after the calling  
 28577 process successfully returns, the child process shall exit with exit status 127).

28578 If POSIX\_SPAWN\_SETSCHEDULER is set in the *spawn-flags* attribute of the object referenced by  
 28579 *attrp*, and if *posix\_spawn()* or *posix\_spawnnp()* fails for any of the reasons that would cause  
 28580 *sched\_setscheduler()* to fail, an error value shall be returned as described by *sched\_setscheduler()*  
 28581 (or, if the error occurs after the calling process successfully returns, the child process shall exit  
 28582 with exit status 127).

28583 If the *file\_actions* argument is not NULL, and specifies any *close*, *dup2*, or *open* actions to be  
 28584 performed, and if *posix\_spawn()* or *posix\_spawnnp()* fails for any of the reasons that would cause  
 28585 *close()*, *dup2()*, or *open()* to fail, an error value shall be returned as described by *close()*,  
 28586 *dup2()*, and *open()*, respectively (or, if the error occurs after the calling process successfully returns, the  
 28587 child process shall exit with exit status 127). An open file action may, by itself, result in any of  
 28588 the errors described by *close()* or *dup2()*, in addition to those described by *open()*.

28589 **EXAMPLES**

28590 None.

28591 **APPLICATION USAGE**

28592 These functions are part of the Spawn option and need not be provided on all implementations.

28593 **RATIONALE**

28594 The *posix\_spawn()* function and its close relation *posix\_spawnnp()* have been introduced to  
 28595 overcome the following perceived difficulties with *fork()*: the *fork()* function is difficult or  
 28596 impossible to implement without swapping or dynamic address translation.

- 28597 • Swapping is generally too slow for a realtime environment.
- 28598 • Dynamic address translation is not available everywhere that POSIX might be useful.
- 28599 • Processes are too useful to simply option out of POSIX whenever it must run without  
 28600 address translation or other MMU services.

28601 Thus, POSIX needs process creation and file execution primitives that can be efficiently  
 28602 implemented without address translation or other MMU services.

28603 The *posix\_spawn()* function is implementable as a library routine, but both *posix\_spawn()* and  
 28604 *posix\_spawnnp()* are designed as kernel operations. Also, although they may be an efficient  
 28605 replacement for many *fork()/exec* pairs, their goal is to provide useful process creation  
 28606 primitives for systems that have difficulty with *fork()*, not to provide drop-in replacements for  
 28607 *fork()/exec*.

28608 This view of the role of *posix\_spawn()* and *posix\_spawnnp()* influenced the design of their API. It  
 28609 does not attempt to provide the full functionality of *fork()/exec* in which arbitrary user-specified  
 28610 operations of any sort are permitted between the creation of the child process and the execution  
 28611 of the new process image; any attempt to reach that level would need to provide a programming  
 28612 language as parameters. Instead, *posix\_spawn()* and *posix\_spawnnp()* are process creation  
 28613 primitives like the *Start\_Process* and *Start\_Process\_Search* Ada language bindings package  
 28614 *POSIX\_Process\_Primitives* and also like those in many operating systems that are not UNIX

28615 systems, but with some POSIX-specific additions.

28616 To achieve its coverage goals, *posix\_spawn()* and *posix\_spawnp()* have control of six types of  
 28617 inheritance: file descriptors, process group ID, user and group ID, signal mask, scheduling, and  
 28618 whether each signal ignored in the parent will remain ignored in the child, or be reset to its  
 28619 default action in the child.

28620 Control of file descriptors is required to allow an independently written child process image to  
 28621 access data streams opened by and even generated or read by the parent process without being  
 28622 specifically coded to know which parent files and file descriptors are to be used. Control of the  
 28623 process group ID is required to control how the child process' job control relates to that of the  
 28624 parent.

28625 Control of the signal mask and signal defaulting is sufficient to support the implementation of  
 28626 *system()*. Although support for *system()* is not explicitly one of the goals for *posix\_spawn()* and  
 28627 *posix\_spawnp()*, it is covered under the "at least 50%" coverage goal.

28628 The intention is that the normal file descriptor inheritance across *fork()*, the subsequent effect of  
 28629 the specified spawn file actions, and the normal file descriptor inheritance across one of the *exec*  
 28630 family of functions should fully specify open file inheritance. The implementation need make no  
 28631 decisions regarding the set of open file descriptors when the child process image begins  
 28632 execution, those decisions having already been made by the caller and expressed as the set of  
 28633 open file descriptors and their *FD\_CLOEXEC* flags at the time of the call and the spawn file  
 28634 actions object specified in the call. We have been assured that in cases where the POSIX  
 28635 *Start\_Process* Ada primitives have been implemented in a library, this method of controlling file  
 28636 descriptor inheritance may be implemented very easily.

28637 We can identify several problems with *posix\_spawn()* and *posix\_spawnp()*, but there does not  
 28638 appear to be a solution that introduces fewer problems. Environment modification for child  
 28639 process attributes not specifiable via the *attrp* or *file\_actions* arguments must be done in the  
 28640 parent process, and since the parent generally wants to save its context, it is more costly than  
 28641 similar functionality with *fork()/exec*. It is also complicated to modify the environment of a  
 28642 multi-threaded process temporarily, since all threads must agree when it is safe for the  
 28643 environment to be changed. However, this cost is only borne by those invocations of  
 28644 *posix\_spawn()* and *posix\_spawnp()* that use the additional functionality. Since extensive  
 28645 modifications are not the usual case, and are particularly unlikely in time-critical code, keeping  
 28646 much of the environment control out of *posix\_spawn()* and *posix\_spawnp()* is appropriate design.

28647 The *posix\_spawn()* and *posix\_spawnp()* functions do not have all the power of *fork()/exec*. This is  
 28648 to be expected. The *fork()* function is a wonderfully powerful operation. We do not expect to  
 28649 duplicate its functionality in a simple, fast function with no special hardware requirements. It is  
 28650 worth noting that *posix\_spawn()* and *posix\_spawnp()* are very similar to the process creation  
 28651 operations on many operating systems that are not UNIX systems.

## 28652 Requirements

28653 The requirements for *posix\_spawn()* and *posix\_spawnp()* are:

- 28654 • They must be implementable without an MMU or unusual hardware.
- 28655 • They must be compatible with existing POSIX standards.

28656 Additional goals are:

- 28657 • They should be efficiently implementable.
- 28658 • They should be able to replace at least 50% of typical executions of *fork()*.

28659           • A system with *posix\_spawn()* and *posix\_spawnp()* and without *fork()* should be useful, at least  
28660 for realtime applications.

28661           • A system with *fork()* and the *exec* family should be able to implement *posix\_spawn()* and  
28662 *posix\_spawnp()* as library routines.

28663           **Two-Syntax**

28664           POSIX *exec* has several calling sequences with approximately the same functionality. These  
28665 appear to be required for compatibility with existing practice. Since the existing practice for the  
28666 *posix\_spawn\*()* functions is otherwise substantially unlike POSIX, we feel that simplicity  
28667 outweighs compatibility. There are, therefore, only two names for the *posix\_spawn\*()* functions.

28668           The parameter list does not differ between *posix\_spawn()* and *posix\_spawnp()*; *posix\_spawnp()*  
28669 interprets the second parameter more elaborately than *posix\_spawn()*.

28670           **Compatibility with POSIX.5 (Ada)**

28671           The *Start\_Process* and *Start\_Process\_Search* procedures from the *POSIX\_Process\_Primitives*  
28672 package from the Ada language binding to POSIX.1 encapsulate *fork()* and *exec* functionality in a  
28673 manner similar to that of *posix\_spawn()* and *posix\_spawnp()*. Originally, in keeping with our  
28674 simplicity goal, the standard developers had limited the capabilities of *posix\_spawn()* and  
28675 *posix\_spawnp()* to a subset of the capabilities of *Start\_Process* and *Start\_Process\_Search*; certain  
28676 non-default capabilities were not supported. However, based on suggestions by the ballot group  
28677 to improve file descriptor mapping or drop it, and on the advice of an Ada Language Bindings  
28678 working group member, the standard developers decided that *posix\_spawn()* and *posix\_spawnp()*  
28679 should be sufficiently powerful to implement *Start\_Process* and *Start\_Process\_Search*. The  
28680 rationale is that if the Ada language binding to such a primitive had already been approved as  
28681 an IEEE standard, there can be little justification for not approving the functionally-equivalent  
28682 parts of a C binding. The only three capabilities provided by *posix\_spawn()* and *posix\_spawnp()*  
28683 that are not provided by *Start\_Process* and *Start\_Process\_Search* are optionally specifying the  
28684 child's process group ID, the set of signals to be reset to default signal handling in the child  
28685 process, and the child's scheduling policy and parameters.

28686           For the Ada language binding for *Start\_Process* to be implemented with *posix\_spawn()*, that  
28687 binding would need to explicitly pass an empty signal mask and the parent's environment to  
28688 *posix\_spawn()* whenever the caller of *Start\_Process* allowed these arguments to default, since  
28689 *posix\_spawn()* does not provide such defaults. The ability of *Start\_Process* to mask user-specified  
28690 signals during its execution is functionally unique to the Ada language binding and must be  
28691 dealt with in the binding separately from the call to *posix\_spawn()*.

28692           **Process Group**

28693           The process group inheritance field can be used to join the child process with an existing process  
28694 group. By assigning a value of zero to the *spawn-pgroup* attribute of the object referenced by  
28695 *attrp*, the *setpgid()* mechanism will place the child process in a new process group.

28696 **Threads**

28697 Without the *posix\_spawn()* and *posix\_spawnp()* functions, systems without address translation  
28698 can still use threads to give an abstraction of concurrency. In many cases, thread creation  
28699 suffices, but it is not always a good substitute. The *posix\_spawn()* and *posix\_spawnp()* functions  
28700 are considerably “heavier” than thread creation. Processes have several important attributes that  
28701 threads do not. Even without address translation, a process may have base-and-bound memory  
28702 protection. Each process has a process environment including security attributes and file  
28703 capabilities, and powerful scheduling attributes. Processes abstract the behavior of non-  
28704 uniform-memory-architecture multi-processors better than threads, and they are more  
28705 convenient to use for activities that are not closely linked.

28706 The *posix\_spawn()* and *posix\_spawnp()* functions may not bring support for multiple processes to  
28707 every configuration. Process creation is not the only piece of operating system support required  
28708 to support multiple processes. The total cost of support for multiple processes may be quite high  
28709 in some circumstances. Existing practice shows that support for multiple processes is  
28710 uncommon and threads are common among “tiny kernels”. There should, therefore, probably  
28711 continue to be AEPs for operating systems with only one process.

28712 **Asynchronous Error Notification**

28713 A library implementation of *posix\_spawn()* or *posix\_spawnp()* may not be able to detect all  
28714 possible errors before it forks the child process. IEEE Std 1003.1-2001 provides for an error  
28715 indication returned from a child process which could not successfully complete the spawn  
28716 operation via a special exit status which may be detected using the status value returned by  
28717 *wait()* and *waitpid()*.

28718 The *stat\_val* interface and the macros used to interpret it are not well suited to the purpose of  
28719 returning API errors, but they are the only path available to a library implementation. Thus, an  
28720 implementation may cause the child process to exit with exit status 127 for any error detected  
28721 during the spawn process after the *posix\_spawn()* or *posix\_spawnp()* function has successfully  
28722 returned.

28723 The standard developers had proposed using two additional macros to interpret *stat\_val*. The  
28724 first, WIFSPAWNFAIL, would have detected a status that indicated that the child exited because  
28725 of an error detected during the *posix\_spawn()* or *posix\_spawnp()* operations rather than during  
28726 actual execution of the child process image; the second, WSPAWNERRNO, would have  
28727 extracted the error value if WIFSPAWNFAIL indicated a failure. Unfortunately, the ballot group  
28728 strongly opposed this because it would make a library implementation of *posix\_spawn()* or  
28729 *posix\_spawnp()* dependent on kernel modifications to *waitpid()* to be able to embed special  
28730 information in *stat\_val* to indicate a spawn failure.

28731 The 8 bits of child process exit status that are guaranteed by IEEE Std 1003.1-2001 to be  
28732 accessible to the waiting parent process are insufficient to disambiguate a spawn error from any  
28733 other kind of error that may be returned by an arbitrary process image. No other bits of the exit  
28734 status are required to be visible in *stat\_val*, so these macros could not be strictly implemented at  
28735 the library level. Reserving an exit status of 127 for such spawn errors is consistent with the use  
28736 of this value by *system()* and *popen()* to signal failures in these operations that occur after the  
28737 function has returned but before a shell is able to execute. The exit status of 127 does not  
28738 uniquely identify this class of error, nor does it provide any detailed information on the nature  
28739 of the failure. Note that a kernel implementation of *posix\_spawn()* or *posix\_spawnp()* is permitted  
28740 (and encouraged) to return any possible error as the function value, thus providing more  
28741 detailed failure information to the parent process.

28742 Thus, no special macros are available to isolate asynchronous *posix\_spawn()* or *posix\_spawnp()*  
28743 errors. Instead, errors detected by the *posix\_spawn()* or *posix\_spawnp()* operations in the context

28744 of the child process before the new process image executes are reported by setting the child's  
 28745 exit status to 127. The calling process may use the WIFEXITED and WEXITSTATUS macros on  
 28746 the *stat\_val* stored by the *wait()* or *waitpid()* functions to detect spawn failures to the extent that  
 28747 other status values with which the child process image may exit (before the parent can  
 28748 conclusively determine that the child process image has begun execution) are distinct from exit  
 28749 status 127.

28750 **FUTURE DIRECTIONS**

28751 None.

28752 **SEE ALSO**

28753 *alarm()*, *chmod()*, *close()*, *dup()*, *exec*, *exit()*, *fcntl()*, *fork()*, *kill()*, *open()*,  
 28754 *posix\_spawn\_file\_actions\_addclose()*, *posix\_spawn\_file\_actions\_adddup2()*,  
 28755 *posix\_spawn\_file\_actions\_addopen()*, *posix\_spawn\_file\_actions\_destroy()*, <REFERENCE  
 28756 UNDEFINED>( *posix\_spawn\_file\_actions\_init()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_init()*,  
 28757 *posix\_spawnattr\_getsigdefault()*, *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getpgroup()*,  
 28758 *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*, *posix\_spawnattr\_getsigmask()*,  
 28759 *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setflags()*, *posix\_spawnattr\_setpgroup()*,  
 28760 *posix\_spawnattr\_setschedparam()*, *posix\_spawnattr\_setschedpolicy()*, *posix\_spawnattr\_setsigmask()*,  
 28761 *sched\_setparam()*, *sched\_setscheduler()*, *setpgid()*, *setuid()*, *stat()*, *times()*, *wait()*, the Base  
 28762 Definitions volume of IEEE Std 1003.1-2001, <**spawn.h**>

28763 **CHANGE HISTORY**

28764 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28765 IEEE PASC Interpretation 1003.1 #103 is applied, noting that the signal default actions are  
 28766 changed as well as the signal mask in step 2.

28767 IEEE PASC Interpretation 1003.1 #132 is applied.

28768 **NAME**

28769 posix\_spawn\_file\_actions\_addclose, posix\_spawn\_file\_actions\_addopen — add close or open  
 28770 action to spawn file actions object (**ADVANCED REALTIME**)

28771 **SYNOPSIS**

```
28772 SPN #include <spawn.h>
28773
28774 int posix_spawn_file_actions_addclose(posix_spawn_file_actions_t *
28775 file_actions, int fildes);
28776 int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *
28777 restrict file_actions, int fildes,
28778 const char *restrict path, int oflag, mode_t mode);
```

28779 **DESCRIPTION**

28780 These functions shall add or delete a close or open action to a spawn file actions object.

28781 A spawn file actions object is of type **posix\_spawn\_file\_actions\_t** (defined in `<spawn.h>`) and is  
 28782 used to specify a series of actions to be performed by a `posix_spawn()` or `posix_spawnp()`  
 28783 operation in order to arrive at the set of open file descriptors for the child process given the set of  
 28784 open file descriptors of the parent. IEEE Std 1003.1-2001 does not define comparison or  
 28785 assignment operators for the type **posix\_spawn\_file\_actions\_t**.

28786 A spawn file actions object, when passed to `posix_spawn()` or `posix_spawnp()`, shall specify how  
 28787 the set of open file descriptors in the calling process is transformed into a set of potentially open  
 28788 file descriptors for the spawned process. This transformation shall be as if the specified sequence  
 28789 of actions was performed exactly once, in the context of the spawned process (prior to execution  
 28790 of the new process image), in the order in which the actions were added to the object;  
 28791 additionally, when the new process image is executed, any file descriptor (from this new set)  
 28792 which has its `FD_CLOEXEC` flag set shall be closed (see `posix_spawn()`).

28793 The `posix_spawn_file_actions_addclose()` function shall add a *close* action to the object referenced  
 28794 by *file\_actions* that shall cause the file descriptor *fildes* to be closed (as if `close(fildes)` had been  
 28795 called) when a new process is spawned using this file actions object.

28796 The `posix_spawn_file_actions_addopen()` function shall add an *open* action to the object referenced  
 28797 by *file\_actions* that shall cause the file named by *path* to be opened (as if `open(path, oflag, mode)`  
 28798 had been called, and the returned file descriptor, if not *fildes*, had been changed to *fildes*) when a  
 28799 new process is spawned using this file actions object. If *fildes* was already an open file descriptor,  
 28800 it shall be closed before the new file is opened.

28801 The string described by *path* shall be copied by the `posix_spawn_file_actions_addopen()` function.

28802 **RETURN VALUE**

28803 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
 28804 be returned to indicate the error.

28805 **ERRORS**

28806 These functions shall fail if:

28807 [EBADF] The value specified by *fildes* is negative or greater than or equal to  
 28808 {OPEN\_MAX}.

28809 These functions may fail if:

28810 [EINVAL] The value specified by *file\_actions* is invalid.

28811 [ENOMEM] Insufficient memory exists to add to the spawn file actions object.

28812 It shall not be considered an error for the *files* argument passed to these functions to specify a  
 28813 file descriptor for which the specified operation could not be performed at the time of the call.  
 28814 Any such error will be detected when the associated file actions object is later used during a  
 28815 *posix\_spawn()* or *posix\_spawnnp()* operation.

28816 **EXAMPLES**

28817 None.

28818 **APPLICATION USAGE**

28819 These functions are part of the Spawn option and need not be provided on all implementations.

28820 **RATIONALE**

28821 A spawn file actions object may be initialized to contain an ordered sequence of *close()*, *dup2()*,  
 28822 and *open()* operations to be used by *posix\_spawn()* or *posix\_spawnnp()* to arrive at the set of open  
 28823 file descriptors inherited by the spawned process from the set of open file descriptors in the  
 28824 parent at the time of the *posix\_spawn()* or *posix\_spawnnp()* call. It had been suggested that the  
 28825 *close()* and *dup2()* operations alone are sufficient to rearrange file descriptors, and that files  
 28826 which need to be opened for use by the spawned process can be handled either by having the  
 28827 calling process open them before the *posix\_spawn()* or *posix\_spawnnp()* call (and close them after),  
 28828 or by passing filenames to the spawned process (in *argv*) so that it may open them itself. The  
 28829 standard developers recommend that applications use one of these two methods when practical,  
 28830 since detailed error status on a failed open operation is always available to the application this  
 28831 way. However, the standard developers feel that allowing a spawn file actions object to specify  
 28832 open operations is still appropriate because:

- 28833 1. It is consistent with equivalent POSIX.5 (Ada) functionality.
- 28834 2. It supports the I/O redirection paradigm commonly employed by POSIX programs  
 28835 designed to be invoked from a shell. When such a program is the child process, it may not  
 28836 be designed to open files on its own.
- 28837 3. It allows file opens that might otherwise fail or violate file ownership/access rights if  
 28838 executed by the parent process.

28839 Regarding 2. above, note that the spawn open file action provides to *posix\_spawn()* and  
 28840 *posix\_spawnnp()* the same capability that the shell redirection operators provide to *system()*, only  
 28841 without the intervening execution of a shell; for example:

```
28842 system ("myprog <file1 3<file2");
```

28843 Regarding 3. above, note that if the calling process needs to open one or more files for access by  
 28844 the spawned process, but has insufficient spare file descriptors, then the open action is necessary  
 28845 to allow the *open()* to occur in the context of the child process after other file descriptors have  
 28846 been closed (that must remain open in the parent).

28847 Additionally, if a parent is executed from a file having a “set-user-id” mode bit set and the  
 28848 POSIX\_SPAWN\_RESETEUIDS flag is set in the spawn attributes, a file created within the parent  
 28849 process will (possibly incorrectly) have the parent’s effective user ID as its owner, whereas a file  
 28850 created via an *open()* action during *posix\_spawn()* or *posix\_spawnnp()* will have the parent’s real  
 28851 ID as its owner; and an open by the parent process may successfully open a file to which the real  
 28852 user should not have access or fail to open a file to which the real user should have access.

28853 **File Descriptor Mapping**

28854 The standard developers had originally proposed using an array which specified the mapping of  
28855 child file descriptors back to those of the parent. It was pointed out by the ballot group that it is  
28856 not possible to reshuffle file descriptors arbitrarily in a library implementation of *posix\_spawn()*  
28857 or *posix\_spawnnp()* without provision for one or more spare file descriptor entries (which simply  
28858 may not be available). Such an array requires that an implementation develop a complex  
28859 strategy to achieve the desired mapping without inadvertently closing the wrong file descriptor  
28860 at the wrong time.

28861 It was noted by a member of the Ada Language Bindings working group that the approved Ada  
28862 Language *Start\_Process* family of POSIX process primitives use a caller-specified set of file  
28863 actions to alter the normal *fork()/exec* semantics for inheritance of file descriptors in a very  
28864 flexible way, yet no such problems exist because the burden of determining how to achieve the  
28865 final file descriptor mapping is completely on the application. Furthermore, although the file  
28866 actions interface appears frightening at first glance, it is actually quite simple to implement in  
28867 either a library or the kernel.

28868 **FUTURE DIRECTIONS**

28869 None.

28870 **SEE ALSO**

28871 *close()*, *dup()*, *open()*, *posix\_spawn()*, *posix\_spawn\_file\_actions\_adddup2()*,  
28872 *posix\_spawn\_file\_actions\_destroy()*, *posix\_spawnnp()*, the Base Definitions volume of  
28873 IEEE Std 1003.1-2001, <spawn.h>

28874 **CHANGE HISTORY**

28875 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28876 IEEE PASC Interpretation 1003.1 #105 is applied, adding a note to the DESCRIPTION that the  
28877 string pointed to by *path* is copied by the *posix\_spawn\_file\_actions\_addopen()* function.

28878 **NAME**

28879 posix\_spawn\_file\_actions\_adddup2 — add dup2 action to spawn file actions object  
 28880 (ADVANCED REALTIME)

28881 **SYNOPSIS**

```
28882 SPN #include <spawn.h>
28883
28884 int posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t *
28885 file_actions, int fildes, int newfildes);
```

28886 **DESCRIPTION**

28887 The *posix\_spawn\_file\_actions\_adddup2()* function shall add a *dup2()* action to the object  
 28888 referenced by *file\_actions* that shall cause the file descriptor *fildes* to be duplicated as *newfildes* (as  
 28889 if *dup2(fildes, newfildes)* had been called) when a new process is spawned using this file actions  
 28890 object.

28891 A spawn file actions object is as defined in *posix\_spawn\_file\_actions\_addclose()*.

28892 **RETURN VALUE**

28893 Upon successful completion, the *posix\_spawn\_file\_actions\_adddup2()* function shall return zero;  
 28894 otherwise, an error number shall be returned to indicate the error.

28895 **ERRORS**

28896 The *posix\_spawn\_file\_actions\_adddup2()* function shall fail if:

28897 [EBADF] The value specified by *fildes* or *newfildes* is negative or greater than or equal to  
 28898 {OPEN\_MAX}.

28899 [ENOMEM] Insufficient memory exists to add to the spawn file actions object.

28900 The *posix\_spawn\_file\_actions\_adddup2()* function may fail if:

28901 [EINVAL] The value specified by *file\_actions* is invalid.

28902 It shall not be considered an error for the *fildes* argument passed to the  
 28903 *posix\_spawn\_file\_actions\_adddup2()* function to specify a file descriptor for which the specified  
 28904 operation could not be performed at the time of the call. Any such error will be detected when  
 28905 the associated file actions object is later used during a *posix\_spawn()* or *posix\_spawnnp()*  
 28906 operation.

28907 **EXAMPLES**

28908 None.

28909 **APPLICATION USAGE**

28910 The *posix\_spawn\_file\_actions\_adddup2()* function is part of the Spawn option and need not be  
 28911 provided on all implementations.

28912 **RATIONALE**

28913 Refer to the RATIONALE in *posix\_spawn\_file\_actions\_addclose()*.

28914 **FUTURE DIRECTIONS**

28915 None.

28916 **SEE ALSO**

28917 *dup()*, *posix\_spawn()*, *posix\_spawn\_file\_actions\_addclose()*, *posix\_spawn\_file\_actions\_destroy()*,  
 28918 *posix\_spawnnp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <spawn.h>

28919 **CHANGE HISTORY**

28920 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28921 IEEE PASC Interpretation 1003.1 #104 is applied, noting that the [EBADF] error can apply to the

28922 *newfildes* argument in addition to *fildes*.

28923 **NAME**

28924 posix\_spawn\_file\_actions\_addopen — add open action to spawn file actions object  
28925 (ADVANCED REALTIME)

28926 **SYNOPSIS**

```
28927 SPN #include <spawn.h>

28928 int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *
28929 restrict file_actions, int fildes,
28930 const char *restrict path, int oflag, mode_t mode);
28931
```

28932 **DESCRIPTION**

28933 Refer to *posix\_spawn\_file\_actions\_addclose()*.

28934 **NAME**

28935 posix\_spawn\_file\_actions\_destroy, posix\_spawn\_file\_actions\_init — destroy and initialize  
 28936 spawn file actions object (**ADVANCED REALTIME**)

28937 **SYNOPSIS**

```
28938 SPN #include <spawn.h>

28939 int posix_spawn_file_actions_destroy(posix_spawn_file_actions_t *
28940 file_actions);
28941 int posix_spawn_file_actions_init(posix_spawn_file_actions_t *
28942 file_actions);
28943
```

28944 **DESCRIPTION**

28945 The *posix\_spawn\_file\_actions\_destroy()* function shall destroy the object referenced by *file\_actions*;  
 28946 the object becomes, in effect, uninitialized. An implementation may cause  
 28947 *posix\_spawn\_file\_actions\_destroy()* to set the object referenced by *file\_actions* to an invalid value. A  
 28948 destroyed spawn file actions object can be reinitialized using *posix\_spawn\_file\_actions\_init()*; the  
 28949 results of otherwise referencing the object after it has been destroyed are undefined.

28950 The *posix\_spawn\_file\_actions\_init()* function shall initialize the object referenced by *file\_actions* to  
 28951 contain no file actions for *posix\_spawn()* or *posix\_spawnnp()* to perform.

28952 A spawn file actions object is as defined in *posix\_spawn\_file\_actions\_addclose()*.

28953 The effect of initializing an already initialized spawn file actions object is undefined.

28954 **RETURN VALUE**

28955 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
 28956 be returned to indicate the error.

28957 **ERRORS**

28958 The *posix\_spawn\_file\_actions\_init()* function shall fail if:

28959 [ENOMEM] Insufficient memory exists to initialize the spawn file actions object.

28960 The *posix\_spawn\_file\_actions\_destroy()* function may fail if:

28961 [EINVAL] The value specified by *file\_actions* is invalid.

28962 **EXAMPLES**

28963 None.

28964 **APPLICATION USAGE**

28965 These functions are part of the Spawn option and need not be provided on all implementations.

28966 **RATIONALE**

28967 Refer to the RATIONALE in *posix\_spawn\_file\_actions\_addclose()*.

28968 **FUTURE DIRECTIONS**

28969 None.

28970 **SEE ALSO**

28971 *posix\_spawn()*, *posix\_spawnnp()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<spawn.h>**

28972 **CHANGE HISTORY**

28973 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28974 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

## 28975 NAME

28976 `posix_spawnattr_destroy`, `posix_spawnattr_init` — destroy and initialize spawn attributes object  
 28977 (ADVANCED REALTIME)

## 28978 SYNOPSIS

28979 SPN `#include <spawn.h>`

28980 `int posix_spawnattr_destroy(posix_spawnattr_t *attr);`

28981 `int posix_spawnattr_init(posix_spawnattr_t *attr);`

28982

## 28983 DESCRIPTION

28984 The `posix_spawnattr_destroy()` function shall destroy a spawn attributes object. A destroyed `attr`  
 28985 attributes object can be reinitialized using `posix_spawnattr_init()`; the results of otherwise  
 28986 referencing the object after it has been destroyed are undefined. An implementation may cause  
 28987 `posix_spawnattr_destroy()` to set the object referenced by `attr` to an invalid value.

28988 The `posix_spawnattr_init()` function shall initialize a spawn attributes object `attr` with the default  
 28989 value for all of the individual attributes used by the implementation. Results are undefined if  
 28990 `posix_spawnattr_init()` is called specifying an already initialized `attr` attributes object.

28991 A spawn attributes object is of type **posix\_spawnattr\_t** (defined in `<spawn.h>`) and is used to  
 28992 specify the inheritance of process attributes across a spawn operation. IEEE Std 1003.1-2001 does  
 28993 not define comparison or assignment operators for the type **posix\_spawnattr\_t**.

28994 Each implementation shall document the individual attributes it uses and their default values  
 28995 unless these values are defined by IEEE Std 1003.1-2001. Attributes not defined by  
 28996 IEEE Std 1003.1-2001, their default values, and the names of the associated functions to get and  
 28997 set those attribute values are implementation-defined.

28998 The resulting spawn attributes object (possibly modified by setting individual attribute values),  
 28999 is used to modify the behavior of `posix_spawn()` or `posix_spawnp()`. After a spawn attributes  
 29000 object has been used to spawn a process by a call to a `posix_spawn()` or `posix_spawnp()`, any  
 29001 function affecting the attributes object (including destruction) shall not affect any process that  
 29002 has been spawned in this way.

## 29003 RETURN VALUE

29004 Upon successful completion, `posix_spawnattr_destroy()` and `posix_spawnattr_init()` shall return  
 29005 zero; otherwise, an error number shall be returned to indicate the error.

## 29006 ERRORS

29007 The `posix_spawnattr_init()` function shall fail if:

29008 [ENOMEM] Insufficient memory exists to initialize the spawn attributes object.

29009 The `posix_spawnattr_destroy()` function may fail if:

29010 [EINVAL] The value specified by `attr` is invalid.

## 29011 EXAMPLES

29012 None.

## 29013 APPLICATION USAGE

29014 These functions are part of the Spawn option and need not be provided on all implementations.

## 29015 RATIONALE

29016 The original spawn interface proposed in IEEE Std 1003.1-2001 defined the attributes that specify  
 29017 the inheritance of process attributes across a spawn operation as a structure. In order to be able  
 29018 to separate optional individual attributes under their appropriate options (that is, the `spawn-`  
 29019 `schedparam` and `spawn-schedpolicy` attributes depending upon the Process Scheduling option), and

29020 also for extensibility and consistency with the newer POSIX interfaces, the attributes interface  
29021 has been changed to an opaque data type. This interface now consists of the type  
29022 **posix\_spawnattr\_t**, representing a spawn attributes object, together with associated functions to  
29023 initialize or destroy the attributes object, and to set or get each individual attribute. Although the  
29024 new object-oriented interface is more verbose than the original structure, it is simple to use,  
29025 more extensible, and easy to implement.

**29026 FUTURE DIRECTIONS**

29027 None.

**29028 SEE ALSO**

29029 *posix\_spawn()*, *posix\_spawnattr\_getsigdefault()*, *posix\_spawnattr\_getflags()*,  
29030 *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*,  
29031 *posix\_spawnattr\_getsigmask()*, *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setflags()*,  
29032 *posix\_spawnattr\_setpgroup()*, *posix\_spawnattr\_setsigmask()*, *posix\_spawnattr\_setschedpolicy()*,  
29033 *posix\_spawnattr\_setschedparam()*, *posix\_spawn()*, the Base Definitions volume of  
29034 IEEE Std 1003.1-2001, <spawn.h>

**29035 CHANGE HISTORY**

29036 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

29037 IEEE PASC Interpretation 1003.1 #106 is applied, noting that the effect of initializing an already  
29038 initialized spawn attributes option is undefined.

29039 **NAME**

29040 posix\_spawnattr\_getflags, posix\_spawnattr\_setflags — get and set the spawn-flags attribute of a  
 29041 spawn attributes object (**ADVANCED REALTIME**)

29042 **SYNOPSIS**

```
29043 SPN #include <spawn.h>
29044 int posix_spawnattr_getflags(const posix_spawnattr_t *restrict attr,
29045 short *restrict flags);
29046 int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
29047
```

29048 **DESCRIPTION**

29049 The *posix\_spawnattr\_getflags()* function shall obtain the value of the *spawn-flags* attribute from  
 29050 the attributes object referenced by *attr*.

29051 The *posix\_spawnattr\_setflags()* function shall set the *spawn-flags* attribute in an initialized  
 29052 attributes object referenced by *attr*.

29053 The *spawn-flags* attribute is used to indicate which process attributes are to be changed in the  
 29054 new process image when invoking *posix\_spawn()* or *posix\_spawnp()*. It is the bitwise-inclusive  
 29055 OR of zero or more of the following flags:

- 29056 POSIX\_SPAWN\_RESETEIDS
- 29057 POSIX\_SPAWN\_SETPGROUP
- 29058 POSIX\_SPAWN\_SETSIGDEF
- 29059 POSIX\_SPAWN\_SETSIGMASK
- 29060 PS POSIX\_SPAWN\_SETSCHEDPARAM
- 29061 POSIX\_SPAWN\_SETSCHEDULER
- 29062

29063 These flags are defined in **<spawn.h>**. The default value of this attribute shall be as if no flags  
 29064 were set.

29065 **RETURN VALUE**

29066 Upon successful completion, *posix\_spawnattr\_getflags()* shall return zero and store the value of  
 29067 the *spawn-flags* attribute of *attr* into the object referenced by the *flags* parameter; otherwise, an  
 29068 error number shall be returned to indicate the error.

29069 Upon successful completion, *posix\_spawnattr\_setflags()* shall return zero; otherwise, an error  
 29070 number shall be returned to indicate the error.

29071 **ERRORS**

29072 These functions may fail if:

29073 [EINVAL] The value specified by *attr* is invalid.

29074 The *posix\_spawnattr\_setflags()* function may fail if:

29075 [EINVAL] The value of the attribute being set is not valid.

29076 **EXAMPLES**

29077 None.

29078 **APPLICATION USAGE**

29079 These functions are part of the Spawn option and need not be provided on all implementations.

29080 **RATIONALE**

29081 None.

29082 **FUTURE DIRECTIONS**

29083 None.

29084 **SEE ALSO**

29085 *posix\_spawn()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_init()*, *posix\_spawnattr\_getsigdefault()*,  
29086 *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*,  
29087 *posix\_spawnattr\_getsigmask()*, *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setpgroup()*,  
29088 *posix\_spawnattr\_setschedparam()*, *posix\_spawnattr\_setschedpolicy()*, *posix\_spawnattr\_setsigmask()*,  
29089 *posix\_spawnnp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <spawn.h>

29090 **CHANGE HISTORY**

29091 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

## 29092 NAME

29093 `posix_spawnattr_getpgroup`, `posix_spawnattr_setpgroup` — get and set the spawn-pgroup  
 29094 attribute of a spawn attributes object (**ADVANCED REALTIME**)

## 29095 SYNOPSIS

```
29096 SPN #include <spawn.h>
29097
29097 int posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict attr,
29098 pid_t *restrict pgroup);
29099 int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
29100
```

## 29101 DESCRIPTION

29102 The `posix_spawnattr_getpgroup()` function shall obtain the value of the `spawn-pgroup` attribute  
 29103 from the attributes object referenced by `attr`.

29104 The `posix_spawnattr_setpgroup()` function shall set the `spawn-pgroup` attribute in an initialized  
 29105 attributes object referenced by `attr`.

29106 The `spawn-pgroup` attribute represents the process group to be joined by the new process image  
 29107 in a spawn operation (if `POSIX_SPAWN_SETPGROUP` is set in the `spawn-flags` attribute). The  
 29108 default value of this attribute shall be zero.

## 29109 RETURN VALUE

29110 Upon successful completion, `posix_spawnattr_getpgroup()` shall return zero and store the value of  
 29111 the `spawn-pgroup` attribute of `attr` into the object referenced by the `pgroup` parameter; otherwise,  
 29112 an error number shall be returned to indicate the error.

29113 Upon successful completion, `posix_spawnattr_setpgroup()` shall return zero; otherwise, an error  
 29114 number shall be returned to indicate the error.

## 29115 ERRORS

29116 These functions may fail if:

29117 [EINVAL] The value specified by `attr` is invalid.

29118 The `posix_spawnattr_setpgroup()` function may fail if:

29119 [EINVAL] The value of the attribute being set is not valid.

## 29120 EXAMPLES

29121 None.

## 29122 APPLICATION USAGE

29123 These functions are part of the Spawn option and need not be provided on all implementations.

## 29124 RATIONALE

29125 None.

## 29126 FUTURE DIRECTIONS

29127 None.

## 29128 SEE ALSO

29129 `posix_spawn()`, `posix_spawnattr_destroy()`, `posix_spawnattr_init()`, `posix_spawnattr_getsigdefault()`,  
 29130 `posix_spawnattr_getflags()`, `posix_spawnattr_getschedparam()`, `posix_spawnattr_getschedpolicy()`,  
 29131 `posix_spawnattr_getsigmask()`, `posix_spawnattr_setsigdefault()`, `posix_spawnattr_setflags()`,  
 29132 `posix_spawnattr_setschedparam()`, `posix_spawnattr_setschedpolicy()`, `posix_spawnattr_setsigmask()`,  
 29133 `posix_spawnnp()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<spawn.h>`

29134 **CHANGE HISTORY**

29135 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

29136 **NAME**

29137 posix\_spawnattr\_getschedparam, posix\_spawnattr\_setschedparam — get and set the spawn-  
 29138 schedparam attribute of a spawn attributes object (**ADVANCED REALTIME**)

29139 **SYNOPSIS**

```
29140 SPN PS #include <spawn.h>
29141 #include <sched.h>

29142 int posix_spawnattr_getschedparam(const posix_spawnattr_t *
29143 restrict attr, struct sched_param *restrict schedparam);
29144 int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,
29145 const struct sched_param *restrict schedparam);
29146
```

29147 **DESCRIPTION**

29148 The *posix\_spawnattr\_getschedparam()* function shall obtain the value of the *spawn-schedparam*  
 29149 attribute from the attributes object referenced by *attr*.

29150 The *posix\_spawnattr\_setschedparam()* function shall set the *spawn-schedparam* attribute in an  
 29151 initialized attributes object referenced by *attr*.

29152 The *spawn-schedparam* attribute represents the scheduling parameters to be assigned to the new  
 29153 process image in a spawn operation (if POSIX\_SPAWN\_SETSCHEDULER or  
 29154 POSIX\_SPAWN\_SETSCHEDPARAM is set in the *spawn-flags* attribute). The default value of this  
 29155 attribute is unspecified.

29156 **RETURN VALUE**

29157 Upon successful completion, *posix\_spawnattr\_getschedparam()* shall return zero and store the  
 29158 value of the *spawn-schedparam* attribute of *attr* into the object referenced by the *schedparam*  
 29159 parameter; otherwise, an error number shall be returned to indicate the error.

29160 Upon successful completion, *posix\_spawnattr\_setschedparam()* shall return zero; otherwise, an  
 29161 error number shall be returned to indicate the error.

29162 **ERRORS**

29163 These functions may fail if:

29164 [EINVAL] The value specified by *attr* is invalid.

29165 The *posix\_spawnattr\_setschedparam()* function may fail if:

29166 [EINVAL] The value of the attribute being set is not valid.

29167 **EXAMPLES**

29168 None.

29169 **APPLICATION USAGE**

29170 These functions are part of the Spawn and Process Scheduling options and need not be provided  
 29171 on all implementations.

29172 **RATIONALE**

29173 None.

29174 **FUTURE DIRECTIONS**

29175 None.

29176 **SEE ALSO**

29177 *posix\_spawn()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_init()*, *posix\_spawnattr\_getsigdefault()*,  
 29178 *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedpolicy()*,  
 29179 *posix\_spawnattr\_getsigmask()*, *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setflags()*,

29180 *posix\_spawnattr\_setpgroup()*, *posix\_spawnattr\_setschedpolicy()*, *posix\_spawnattr\_setsigmask()*,  
29181 *posix\_spawnnp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sched.h>, <spawn.h>

29182 **CHANGE HISTORY**

29183 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

29184 **NAME**

29185 posix\_spawnattr\_getschedpolicy, posix\_spawnattr\_setschedpolicy — get and set the spawn-  
 29186 schedpolicy attribute of a spawn attributes object (**ADVANCED REALTIME**)

29187 **SYNOPSIS**

```
29188 SPN PS #include <spawn.h>
29189 #include <sched.h>

29190 int posix_spawnattr_getschedpolicy(const posix_spawnattr_t *
29191 restrict attr, int *restrict schedpolicy);
29192 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
29193 int schedpolicy);
29194
```

29195 **DESCRIPTION**

29196 The *posix\_spawnattr\_getschedpolicy()* function shall obtain the value of the *spawn-schedpolicy*  
 29197 attribute from the attributes object referenced by *attr*.

29198 The *posix\_spawnattr\_setschedpolicy()* function shall set the *spawn-schedpolicy* attribute in an  
 29199 initialized attributes object referenced by *attr*.

29200 The *spawn-schedpolicy* attribute represents the scheduling policy to be assigned to the new  
 29201 process image in a spawn operation (if POSIX\_SPAWN\_SETSCHEDULER is set in the *spawn-*  
 29202 *flags* attribute). The default value of this attribute is unspecified.

29203 **RETURN VALUE**

29204 Upon successful completion, *posix\_spawnattr\_getschedpolicy()* shall return zero and store the  
 29205 value of the *spawn-schedpolicy* attribute of *attr* into the object referenced by the *schedpolicy*  
 29206 parameter; otherwise, an error number shall be returned to indicate the error.

29207 Upon successful completion, *posix\_spawnattr\_setschedpolicy()* shall return zero; otherwise, an  
 29208 error number shall be returned to indicate the error.

29209 **ERRORS**

29210 These functions may fail if:

29211 [EINVAL] The value specified by *attr* is invalid.

29212 The *posix\_spawnattr\_setschedpolicy()* function may fail if:

29213 [EINVAL] The value of the attribute being set is not valid.

29214 **EXAMPLES**

29215 None.

29216 **APPLICATION USAGE**

29217 These functions are part of the Spawn and Process Scheduling options and need not be provided  
 29218 on all implementations.

29219 **RATIONALE**

29220 None.

29221 **FUTURE DIRECTIONS**

29222 None.

29223 **SEE ALSO**

29224 *posix\_spawn()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_init()*, *posix\_spawnattr\_getsigdefault()*,  
 29225 *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*,  
 29226 *posix\_spawnattr\_getsigmask()*, *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setflags()*,  
 29227 *posix\_spawnattr\_setpgroup()*, *posix\_spawnattr\_setschedparam()*, *posix\_spawnattr\_setsigmask()*,

- 29228            *posix\_spawnp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sched.h>, <spawn.h>
- 29229 **CHANGE HISTORY**
- 29230            First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

## 29231 NAME

29232 `posix_spawnattr_getsigdefault`, `posix_spawnattr_setsigdefault` — get and set the spawn-  
 29233 sigdefault attribute of a spawn attributes object (**ADVANCED REALTIME**)

## 29234 SYNOPSIS

```
29235 SPN #include <signal.h>
29236 #include <spawn.h>

29237 int posix_spawnattr_getsigdefault(const posix_spawnattr_t *
29238 restrict attr, sigset_t *restrict sigdefault);
29239 int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,
29240 const sigset_t *restrict sigdefault);
29241
```

## 29242 DESCRIPTION

29243 The `posix_spawnattr_getsigdefault()` function shall obtain the value of the `spawn-sigdefault`  
 29244 attribute from the attributes object referenced by `attr`.

29245 The `posix_spawnattr_setsigdefault()` function shall set the `spawn-sigdefault` attribute in an  
 29246 initialized attributes object referenced by `attr`.

29247 The `spawn-sigdefault` attribute represents the set of signals to be forced to default signal handling  
 29248 in the new process image (if `POSIX_SPAWN_SETSIGDEF` is set in the `spawn-flags` attribute) by a  
 29249 spawn operation. The default value of this attribute shall be an empty signal set.

## 29250 RETURN VALUE

29251 Upon successful completion, `posix_spawnattr_getsigdefault()` shall return zero and store the value  
 29252 of the `spawn-sigdefault` attribute of `attr` into the object referenced by the `sigdefault` parameter;  
 29253 otherwise, an error number shall be returned to indicate the error.

29254 Upon successful completion, `posix_spawnattr_setsigdefault()` shall return zero; otherwise, an error  
 29255 number shall be returned to indicate the error.

## 29256 ERRORS

29257 These functions may fail if:

29258 [EINVAL] The value specified by `attr` is invalid.

29259 The `posix_spawnattr_setsigdefault()` function may fail if:

29260 [EINVAL] The value of the attribute being set is not valid.

## 29261 EXAMPLES

29262 None.

## 29263 APPLICATION USAGE

29264 These functions are part of the Spawn option and need not be provided on all implementations.

## 29265 RATIONALE

29266 None.

## 29267 FUTURE DIRECTIONS

29268 None.

## 29269 SEE ALSO

29270 `posix_spawn()`, `posix_spawnattr_destroy()`, `posix_spawnattr_init()`, `posix_spawnattr_getflags()`,  
 29271 `posix_spawnattr_getpgroup()`, `posix_spawnattr_getschedparam()`, `posix_spawnattr_getschedpolicy()`,  
 29272 `posix_spawnattr_getsigmask()`, `posix_spawnattr_setflags()`, `posix_spawnattr_setpgroup()`,  
 29273 `posix_spawnattr_setschedparam()`, `posix_spawnattr_setschedpolicy()`, `posix_spawnattr_setsigmask()`,  
 29274 `posix_spawnnp()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<signal.h>`, `<spawn.h>`

29275 **CHANGE HISTORY**

29276 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

## 29277 NAME

29278 `posix_spawnattr_getsigmask`, `posix_spawnattr_setsigmask` — get and set the spawn-sigmask  
 29279 attribute of a spawn attributes object (**ADVANCED REALTIME**)

## 29280 SYNOPSIS

```
29281 SPN #include <signal.h>
29282 #include <spawn.h>

29283 int posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict attr,
29284 sigset_t *restrict sigmask);
29285 int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,
29286 const sigset_t *restrict sigmask);
29287
```

## 29288 DESCRIPTION

29289 The `posix_spawnattr_getsigmask()` function shall obtain the value of the `spawn-sigmask` attribute  
 29290 from the attributes object referenced by `attr`.

29291 The `posix_spawnattr_setsigmask()` function shall set the `spawn-sigmask` attribute in an initialized  
 29292 attributes object referenced by `attr`.

29293 The `spawn-sigmask` attribute represents the signal mask in effect in the new process image of a  
 29294 spawn operation (if `POSIX_SPAWN_SETSIGMASK` is set in the `spawn-flags` attribute). The  
 29295 default value of this attribute is unspecified.

## 29296 RETURN VALUE

29297 Upon successful completion, `posix_spawnattr_getsigmask()` shall return zero and store the value  
 29298 of the `spawn-sigmask` attribute of `attr` into the object referenced by the `sigmask` parameter;  
 29299 otherwise, an error number shall be returned to indicate the error.

29300 Upon successful completion, `posix_spawnattr_setsigmask()` shall return zero; otherwise, an error  
 29301 number shall be returned to indicate the error.

## 29302 ERRORS

29303 These functions may fail if:

29304 [EINVAL] The value specified by `attr` is invalid.

29305 The `posix_spawnattr_setsigmask()` function may fail if:

29306 [EINVAL] The value of the attribute being set is not valid.

## 29307 EXAMPLES

29308 None.

## 29309 APPLICATION USAGE

29310 These functions are part of the Spawn option and need not be provided on all implementations.

## 29311 RATIONALE

29312 None.

## 29313 FUTURE DIRECTIONS

29314 None.

## 29315 SEE ALSO

29316 `posix_spawn()`, `posix_spawnattr_destroy()`, `posix_spawnattr_init()`, `posix_spawnattr_getsigdefault()`,  
 29317 `posix_spawnattr_getflags()`, `posix_spawnattr_getpgroup()`, `posix_spawnattr_getschedparam()`,  
 29318 `posix_spawnattr_getschedpolicy()`, `posix_spawnattr_setsigdefault()`, `posix_spawnattr_setflags()`,  
 29319 `posix_spawnattr_setpgroup()`, `posix_spawnattr_setschedparam()`, `posix_spawnattr_setschedpolicy()`,  
 29320 `posix_spawnnp()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<signal.h>`, `<spawn.h>`

29321 **CHANGE HISTORY**

29322 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

29323 **NAME**

29324        `posix_spawnattr_init` — initialize the spawn attributes object (**ADVANCED REALTIME**)

29325 **SYNOPSIS**

29326 SPN     `#include <spawn.h>`

29327        `int posix_spawnattr_init(posix_spawnattr_t *attr);`

29328

29329 **DESCRIPTION**

29330        Refer to `posix_spawnattr_destroy()`.

29331 **NAME**

29332 `posix_spawnattr_setflags` — set the spawn-flags attribute of a spawn attributes object  
29333 (**ADVANCED REALTIME**)

29334 **SYNOPSIS**

29335 SPN `#include <spawn.h>`

29336 `int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);`

29337

29338 **DESCRIPTION**

29339 Refer to `posix_spawnattr_getflags()`.

29340 **NAME**

29341 posix\_spawnattr\_setpgroup — set the spawn-pgroup attribute of a spawn attributes object  
29342 (ADVANCED REALTIME)

29343 **SYNOPSIS**

29344 SPN `#include <spawn.h>`

29345 `int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);`  
29346

29347 **DESCRIPTION**

29348 Refer to *posix\_spawnattr\_getpgroup()*.

29349 **NAME**

29350 `posix_spawnattr_setschedparam` — set the spawn-schedparam attribute of a spawn attributes  
29351 object (**ADVANCED REALTIME**)

29352 **SYNOPSIS**

29353 SPN PS `#include <sched.h>`

29354 `#include <spawn.h>`

```
29355 int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,
29356 const struct sched_param *restrict schedparam);
```

29357

29358 **DESCRIPTION**

29359 Refer to `posix_spawnattr_getschedparam()`.

29360 **NAME**

29361        posix\_spawnattr\_setschedpolicy — set the spawn-schedpolicy attribute of a spawn attributes  
29362        object (**ADVANCED REALTIME**)

29363 **SYNOPSIS**

29364 SPN PS #include <sched.h>

29365        #include <spawn.h>

```
29366 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
29367 int schedpolicy);
```

29368

29369 **DESCRIPTION**

29370        Refer to *posix\_spawnattr\_getschedpolicy()*.

29371 **NAME**

29372 `posix_spawnattr_setsigdefault` — set the spawn-sigdefault attribute of a spawn attributes object  
29373 (**ADVANCED REALTIME**)

29374 **SYNOPSIS**

29375 SPN `#include <signal.h>`

29376 `#include <spawn.h>`

```
29377 int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,
29378 const sigset_t *restrict sigdefault);
```

29379

29380 **DESCRIPTION**

29381 Refer to `posix_spawnattr_getsigdefault()`.

29382 **NAME**

29383        posix\_spawnattr\_setsigmask — set the spawn-sigmask attribute of a spawn attributes object  
29384        (ADVANCED REALTIME)

29385 **SYNOPSIS**

```
29386 SPN #include <signal.h>
29387 #include <spawn.h>

29388 int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,
29389 const sigset_t *restrict sigmask);
29390
```

29391 **DESCRIPTION**

29392        Refer to *posix\_spawnattr\_getsigmask()*.

29393 **NAME**29394        posix\_spawn — spawn a process (**ADVANCED REALTIME**)29395 **SYNOPSIS**

29396 SPN     #include &lt;spawn.h&gt;

```
29397 int posix_spawn(pid_t *restrict pid, const char *restrict file,
29398 const posix_spawn_file_actions_t *file_actions,
29399 const posix_spawnattr_t *restrict attrp,
29400 char *const argv[restrict], char *const envp[restrict]);
```

29401

29402 **DESCRIPTION**29403        Refer to *posix\_spawn()*.

29404 **NAME**

29405 posix\_trace\_attr\_destroy, posix\_trace\_attr\_init — destroy and initialize the trace stream  
 29406 attributes object (**TRACING**)

29407 **SYNOPSIS**

29408 TRC `#include <trace.h>`

29409 `int posix_trace_attr_destroy(trace_attr_t *attr);`

29410 `int posix_trace_attr_init(trace_attr_t *attr);`

29411

29412 **DESCRIPTION**

29413 The *posix\_trace\_attr\_destroy()* function shall destroy an initialized trace attributes object. A  
 29414 destroyed *attr* attributes object can be reinitialized using *posix\_trace\_attr\_init()*; the results of  
 29415 otherwise referencing the object after it has been destroyed are undefined.

29416 The *posix\_trace\_attr\_init()* function shall initialize a trace attributes object *attr* with the default  
 29417 value for all of the individual attributes used by a given implementation. The read-only  
 29418 *generation-version* and *clock-resolution* attributes of the newly initialized trace attributes object  
 29419 shall be set to their appropriate values (see Section 2.11.1.2 (on page 75)).

29420 Results are undefined if *posix\_trace\_attr\_init()* is called specifying an already initialized *attr*  
 29421 attributes object.

29422 Implementations may add extensions to the trace attributes object structure as permitted in the  
 29423 Base Definitions volume of IEEE Std 1003.1-2001, Chapter 2, Conformance.

29424 The resulting attributes object (possibly modified by setting individual attributes values), when  
 29425 used by *posix\_trace\_create()*, defines the attributes of the trace stream created. A single attributes  
 29426 object can be used in multiple calls to *posix\_trace\_create()*. After one or more trace streams have  
 29427 been created using an attributes object, any function affecting that attributes object, including  
 29428 destruction, shall not affect any trace stream previously created. An initialized attributes object  
 29429 also serves to receive the attributes of an existing trace stream or trace log when calling the  
 29430 *posix\_trace\_get\_attr()* function.

29431 **RETURN VALUE**

29432 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 29433 return the corresponding error number.

29434 **ERRORS**

29435 The *posix\_trace\_attr\_destroy()* function may fail if:

29436 [EINVAL] The value of *attr* is invalid.

29437 The *posix\_trace\_attr\_init()* function shall fail if:

29438 [ENOMEM] Insufficient memory exists to initialize the trace attributes object.

29439 **EXAMPLES**

29440 None.

29441 **APPLICATION USAGE**

29442 None.

29443 **RATIONALE**

29444 None.

29445 **FUTURE DIRECTIONS**

29446       None.

29447 **SEE ALSO**29448       *posix\_trace\_create()*, *posix\_trace\_get\_attr()*, *uname()*, the Base Definitions volume of  
29449       IEEE Std 1003.1-2001, <**trace.h**>29450 **CHANGE HISTORY**

29451       First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

29452       IEEE PASC Interpretation 1003.1 #123 is applied.

29453 **NAME**

29454 posix\_trace\_attr\_getclockres, posix\_trace\_attr\_getcreatetime, posix\_trace\_attr\_getgenversion,  
29455 posix\_trace\_attr\_getname, posix\_trace\_attr\_setname — retrieve and set information about a  
29456 trace stream (**TRACING**)

29457 **SYNOPSIS**

```
29458 TRC #include <time.h>
29459 #include <trace.h>

29460 int posix_trace_attr_getclockres(const trace_attr_t *attr,
29461 struct timespec *resolution);
29462 int posix_trace_attr_getcreatetime(const trace_attr_t *attr,
29463 struct timespec *createtime);

29464 #include <trace.h>

29465 int posix_trace_attr_getgenversion(const trace_attr_t *attr,
29466 char *genversion);
29467 int posix_trace_attr_getname(const trace_attr_t *attr,
29468 char *tracename);
29469 int posix_trace_attr_setname(trace_attr_t *attr,
29470 const char *tracename);
29471
```

29472 **DESCRIPTION**

29473 The *posix\_trace\_attr\_getclockres()* function shall copy the clock resolution of the clock used to  
29474 generate timestamps from the *clock-resolution* attribute of the attributes object pointed to by the  
29475 *attr* argument into the structure pointed to by the *resolution* argument.

29476 The *posix\_trace\_attr\_getcreatetime()* function shall copy the trace stream creation time from the  
29477 *creation-time* attribute of the attributes object pointed to by the *attr* argument into the structure  
29478 pointed to by the *createtime* argument. The *creation-time* attribute shall represent the time of  
29479 creation of the trace stream.

29480 The *posix\_trace\_attr\_getgenversion()* function shall copy the string containing version information  
29481 from the *generation-version* attribute of the attributes object pointed to by the *attr* argument into  
29482 the string pointed to by the *genversion* argument. The *genversion* argument shall be the address of  
29483 a character array which can store at least {TRACE\_NAME\_MAX} characters.

29484 The *posix\_trace\_attr\_getname()* function shall copy the string containing the trace name from the  
29485 *trace-name* attribute of the attributes object pointed to by the *attr* argument into the string  
29486 pointed to by the *tracename* argument. The *tracename* argument shall be the address of a character  
29487 array which can store at least {TRACE\_NAME\_MAX} characters.

29488 The *posix\_trace\_attr\_setname()* function shall set the name in the *trace-name* attribute of the  
29489 attributes object pointed to by the *attr* argument, using the trace name string supplied by the  
29490 *tracename* argument. If the supplied string contains more than {TRACE\_NAME\_MAX}  
29491 characters, the name copied into the *trace-name* attribute may be truncated to one less than the  
29492 length of {TRACE\_NAME\_MAX} characters. The default value is a null string.

29493 **RETURN VALUE**

29494 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
29495 return the corresponding error number.

29496 If successful, the *posix\_trace\_attr\_getclockres()* function stores the *clock-resolution* attribute value  
29497 in the object pointed to by *resolution*. Otherwise, the content of this object is unspecified.

29498 If successful, the *posix\_trace\_attr\_getcreatetime()* function stores the trace stream creation time in  
29499 the object pointed to by *createtime*. Otherwise, the content of this object is unspecified.

29500 If successful, the *posix\_trace\_attr\_getgenversion()* function stores the trace version information in  
29501 the string pointed to by *genversion*. Otherwise, the content of this string is unspecified.

29502 If successful, the *posix\_trace\_attr\_getname()* function stores the trace name in the string pointed  
29503 to by *tracename*. Otherwise, the content of this string is unspecified.

#### 29504 ERRORS

29505 The *posix\_trace\_attr\_getclockres()*, *posix\_trace\_attr\_getcreatetime()*, *posix\_trace\_attr\_getgenversion()*,  
29506 and *posix\_trace\_attr\_getname()* functions may fail if:

29507 [EINVAL] The value specified by one of the arguments is invalid.

#### 29508 EXAMPLES

29509 None.

#### 29510 APPLICATION USAGE

29511 None.

#### 29512 RATIONALE

29513 None.

#### 29514 FUTURE DIRECTIONS

29515 None.

#### 29516 SEE ALSO

29517 *posix\_trace\_attr\_init()*, *posix\_trace\_create()*, *posix\_trace\_get\_attr()*, *uname()*, the Base Definitions  
29518 volume of IEEE Std 1003.1-2001, <time.h>, <trace.h>

#### 29519 CHANGE HISTORY

29520 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

## 29521 NAME

29522 `posix_trace_attr_getinherited`, `posix_trace_attr_getlogfullpolicy`,  
 29523 `posix_trace_attr_getstreamfullpolicy`, `posix_trace_attr_setinherited`,  
 29524 `posix_trace_attr_setlogfullpolicy`, `posix_trace_attr_setstreamfullpolicy` — retrieve and set the  
 29525 behavior of a trace stream (**TRACING**)

## 29526 SYNOPSIS

```
29527 TRC #include <trace.h>
29528 TRC TRI int posix_trace_attr_getinherited(const trace_attr_t *restrict attr,
29529 int *restrict inheritancepolicy);
29530 TRC TRL int posix_trace_attr_getlogfullpolicy(const trace_attr_t *restrict attr,
29531 int *restrict logpolicy);
29532 TRC int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *attr,
29533 int *streampolicy);
29534 TRC TRI int posix_trace_attr_setinherited(trace_attr_t *attr,
29535 int inheritancepolicy);
29536 TRC TRL int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
29537 int logpolicy);
29538 TRC int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,
29539 int streampolicy);
29540
```

## 29541 DESCRIPTION

29542 TRI The `posix_trace_attr_getinherited()` and `posix_trace_attr_setinherited()` functions, respectively, shall  
 29543 get and set the inheritance policy stored in the *inheritance* attribute for traced processes across  
 29544 the `fork()` and `spawn()` operations. The *inheritance* attribute of the attributes object pointed to by  
 29545 the *attr* argument shall be set to one of the following values defined by manifest constants in the  
 29546 **<trace.h>** header:

### 29547 POSIX\_TRACE\_CLOSE\_FOR\_CHILD

29548 After a `fork()` or `spawn()` operation, the child shall not be traced, and tracing of the parent  
 29549 shall continue.

### 29550 POSIX\_TRACE\_INHERITED

29551 After a `fork()` or `spawn()` operation, if the parent is being traced, its child shall be  
 29552 concurrently traced using the same trace stream.

29553 The default value for the *inheritance* attribute is `POSIX_TRACE_CLOSE_FOR_CHILD`.

29554 TRL The `posix_trace_attr_getlogfullpolicy()` and `posix_trace_attr_setlogfullpolicy()` functions,  
 29555 respectively, shall get and set the trace log full policy stored in the *log-full-policy* attribute of the  
 29556 attributes object pointed to by the *attr* argument.

29557 The *log-full-policy* attribute shall be set to one of the following values defined by manifest  
 29558 constants in the **<trace.h>** header:

### 29559 POSIX\_TRACE\_LOOP

29560 The trace log shall loop until the associated trace stream is stopped. This policy means that  
 29561 when the trace log gets full, the file system shall reuse the resources allocated to the oldest  
 29562 trace events that were recorded. In this way, the trace log will always contain the most  
 29563 recent trace events flushed.

### 29564 POSIX\_TRACE\_UNTIL\_FULL

29565 The trace stream shall be flushed to the trace log until the trace log is full. This condition can  
 29566 be deduced from the *posix\_log\_full\_status* member status (see the **posix\_trace\_status\_info**  
 29567 structure defined in **<trace.h>**). The last recorded trace event shall be the

|           |                                                                                                                         |
|-----------|-------------------------------------------------------------------------------------------------------------------------|
| 29568     | POSIX_TRACE_STOP trace event.                                                                                           |
| 29569     | POSIX_TRACE_APPEND                                                                                                      |
| 29570     | The associated trace stream shall be flushed to the trace log without log size limitation. If                           |
| 29571     | the application specifies POSIX_TRACE_APPEND, the implementation shall ignore the                                       |
| 29572     | <i>log-max-size</i> attribute.                                                                                          |
| 29573     | The default value for the <i>log-full-policy</i> attribute is POSIX_TRACE_LOOP.                                         |
| 29574     | The <i>posix_trace_attr_getstreamfullpolicy()</i> and <i>posix_trace_attr_setstreamfullpolicy()</i> functions,          |
| 29575     | respectively, shall get and set the trace stream full policy stored in the <i>stream-full-policy</i> attribute          |
| 29576     | of the attributes object pointed to by the <i>attr</i> argument.                                                        |
| 29577     | The <i>stream-full-policy</i> attribute shall be set to one of the following values defined by manifest                 |
| 29578     | constants in the <trace.h> header:                                                                                      |
| 29579     | POSIX_TRACE_LOOP                                                                                                        |
| 29580     | The trace stream shall loop until explicitly stopped by the <i>posix_trace_stop()</i> function. This                    |
| 29581     | policy means that when the trace stream is full, the trace system shall reuse the resources                             |
| 29582     | allocated to the oldest trace events recorded. In this way, the trace stream will always                                |
| 29583     | contain the most recent trace events recorded.                                                                          |
| 29584     | POSIX_TRACE_UNTIL_FULL                                                                                                  |
| 29585     | The trace stream will run until the trace stream resources are exhausted. Then the trace                                |
| 29586     | stream will stop. This condition can be deduced from <i>posix_stream_status</i> and                                     |
| 29587     | <i>posix_stream_full_status</i> (see the <b>posix_trace_status_info</b> structure defined in <trace.h>).                |
| 29588     | When this trace stream is read, a POSIX_TRACE_STOP trace event shall be reported after                                  |
| 29589     | reporting the last recorded trace event. The trace system shall reuse the resources allocated                           |
| 29590     | to any trace events already reported—see the <i>posix_trace_getnext_event()</i> ,                                       |
| 29591     | <i>posix_trace_trygetnext_event()</i> , and <i>posix_trace_timedgetnext_event()</i> functions—or already                |
| 29592     | flushed for an active trace stream with log if the Trace Log option is supported; see the                               |
| 29593     | <i>posix_trace_flush()</i> function. The trace system shall restart the trace stream when it is empty                   |
| 29594     | and may restart it sooner. A POSIX_TRACE_START trace event shall be reported before                                     |
| 29595     | reporting the next recorded trace event.                                                                                |
| 29596 TRL | POSIX_TRACE_FLUSH                                                                                                       |
| 29597     | If the Trace Log option is supported, this policy is identical to the                                                   |
| 29598     | POSIX_TRACE_UNTIL_FULL trace stream full policy except that the trace stream shall be                                   |
| 29599     | flushed regularly as if <i>posix_trace_flush()</i> had been explicitly called. Defining this policy for                 |
| 29600     | an active trace stream without log shall be invalid.                                                                    |
| 29601     | The default value for the <i>stream-full-policy</i> attribute shall be POSIX_TRACE_LOOP for an active                   |
| 29602     | trace stream without log.                                                                                               |
| 29603 TRL | If the Trace Log option is supported, the default value for the <i>stream-full-policy</i> attribute shall be            |
| 29604     | POSIX_TRACE_FLUSH for an active trace stream with log.                                                                  |
| 29605     | <b>RETURN VALUE</b>                                                                                                     |
| 29606     | Upon successful completion, these functions shall return a value of zero. Otherwise, they shall                         |
| 29607     | return the corresponding error number.                                                                                  |
| 29608 TRI | If successful, the <i>posix_trace_attr_getinherited()</i> function shall store the <i>inheritance</i> attribute value   |
| 29609     | in the object pointed to by <i>inheritancepolicy</i> . Otherwise, the content of this object is undefined.              |
| 29610 TRL | If successful, the <i>posix_trace_attr_getlogfullpolicy()</i> function shall store the <i>log-full-policy</i> attribute |
| 29611     | value in the object pointed to by <i>logpolicy</i> . Otherwise, the content of this object is undefined.                |
| 29612     | If successful, the <i>posix_trace_attr_getstreamfullpolicy()</i> function shall store the <i>stream-full-policy</i>     |
| 29613     | attribute value in the object pointed to by <i>streampolicy</i> . Otherwise, the content of this object is              |

29614 undefined.

## 29615 ERRORS

29616 These functions may fail if:

29617 [EINVAL] The value specified by at least one of the arguments is invalid.

## 29618 EXAMPLES

29619 None.

## 29620 APPLICATION USAGE

29621 None.

## 29622 RATIONALE

29623 None.

## 29624 FUTURE DIRECTIONS

29625 None.

## 29626 SEE ALSO

29627 *fork()*, *posix\_trace\_attr\_init()*, *posix\_trace\_create()*, *posix\_trace\_flush()*, *posix\_trace\_get\_attr()*,  
29628 *posix\_trace\_getnext\_event()*, *posix\_trace\_start()*, *posix\_trace\_timedgetnext\_event()*, the Base  
29629 Definitions volume of IEEE Std 1003.1-2001, <trace.h>

## 29630 CHANGE HISTORY

29631 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

29632 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/39 is applied, adding the TRL and TRC |  
29633 margin codes to the *posix\_trace\_attr\_setlogfullpolicy()* function. |

## 29634 NAME

29635 posix\_trace\_attr\_getlogsize, posix\_trace\_attr\_getmaxdatasize,  
 29636 posix\_trace\_attr\_getmaxsystemeventsize, posix\_trace\_attr\_getmaxusereventsize,  
 29637 posix\_trace\_attr\_getstreamsize, posix\_trace\_attr\_setlogsize, posix\_trace\_attr\_setmaxdatasize,  
 29638 posix\_trace\_attr\_setstreamsize — retrieve and set trace stream size attributes (TRACING)

## 29639 SYNOPSIS

```
29640 TRC #include <sys/types.h>
29641 #include <trace.h>

29642 TRC TRL int posix_trace_attr_getlogsize(const trace_attr_t *restrict attr,
29643 size_t *restrict logsize);
29644 TRC int posix_trace_attr_getmaxdatasize(const trace_attr_t *restrict attr,
29645 size_t *restrict maxdatasize);
29646 int posix_trace_attr_getmaxsystemeventsize(
29647 const trace_attr_t *restrict attr,
29648 size_t *restrict eventsize);
29649 int posix_trace_attr_getmaxusereventsize(
29650 const trace_attr_t *restrict attr,
29651 size_t data_len, size_t *restrict eventsize);
29652 int posix_trace_attr_getstreamsize(const trace_attr_t *restrict attr,
29653 size_t *restrict streamsize);
29654 TRC TRL int posix_trace_attr_setlogsize(trace_attr_t *attr,
29655 size_t logsize);
29656 TRC int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,
29657 size_t maxdatasize);
29658 int posix_trace_attr_setstreamsize(trace_attr_t *attr,
29659 size_t streamsize);
29660
```

## 29661 DESCRIPTION

29662 TRL The *posix\_trace\_attr\_getlogsize()* function shall copy the log size, in bytes, from the *log-max-size*  
 29663 attribute of the attributes object pointed to by the *attr* argument into the variable pointed to by  
 29664 the *logsize* argument. This log size is the maximum total of bytes that shall be allocated for  
 29665 system and user trace events in the trace log. The default value for the *log-max-size* attribute is  
 29666 implementation-defined.

29667 The *posix\_trace\_attr\_setlogsize()* function shall set the maximum allowed size, in bytes, in the  
 29668 *log-max-size* attribute of the attributes object pointed to by the *attr* argument, using the size value  
 29669 supplied by the *logsize* argument.

29670 The trace log size shall be used if the *log-full-policy* attribute is set to *POSIX\_TRACE\_LOOP* or  
 29671 *POSIX\_TRACE\_UNTIL\_FULL*. If the *log-full-policy* attribute is set to *POSIX\_TRACE\_APPEND*,  
 29672 the implementation shall ignore the *log-max-size* attribute.

29673 The *posix\_trace\_attr\_getmaxdatasize()* function shall copy the maximum user trace event data  
 29674 size, in bytes, from the *max-data-size* attribute of the attributes object pointed to by the *attr*  
 29675 argument into the variable pointed to by the *maxdatasize* argument. The default value for the  
 29676 *max-data-size* attribute is implementation-defined.

29677 The *posix\_trace\_attr\_getmaxsystemeventsize()* function shall calculate the maximum memory size,  
 29678 in bytes, required to store a single system trace event. This value is calculated for the trace  
 29679 stream attributes object pointed to by the *attr* argument and is returned in the variable pointed  
 29680 to by the *eventsize* argument.

29681 The values returned as the maximum memory sizes of the user and system trace events shall be  
 29682 such that if the sum of the maximum memory sizes of a set of the trace events that may be  
 29683 recorded in a trace stream is less than or equal to the *stream-min-size* attribute of that trace  
 29684 stream, the system provides the necessary resources for recording all those trace events, without  
 29685 loss.

29686 The *posix\_trace\_attr\_getmaxusereventsize()* function shall calculate the maximum memory size, in  
 29687 bytes, required to store a single user trace event generated by a call to *posix\_trace\_event()* with a  
 29688 *data\_len* parameter equal to the *data\_len* value specified in this call. This value is calculated for  
 29689 the trace stream attributes object pointed to by the *attr* argument and is returned in the variable  
 29690 pointed to by the *eventsize* argument.

29691 The *posix\_trace\_attr\_getstreamsize()* function shall copy the stream size, in bytes, from the  
 29692 *stream-min-size* attribute of the attributes object pointed to by the *attr* argument into the variable  
 29693 pointed to by the *streamsize* argument.

29694 This stream size is the current total memory size reserved for system and user trace events in the  
 29695 trace stream. The default value for the *stream-min-size* attribute is implementation-defined. The  
 29696 stream size refers to memory used to store trace event records. Other stream data (for example,  
 29697 trace attribute values) shall not be included in this size.

29698 The *posix\_trace\_attr\_setmaxdatasize()* function shall set the maximum allowed size, in bytes, in  
 29699 the *max-data-size* attribute of the attributes object pointed to by the *attr* argument, using the size  
 29700 value supplied by the *maxdatasize* argument. This maximum size is the maximum allowed size  
 29701 for the user data argument which may be passed to *posix\_trace\_event()*. The implementation  
 29702 shall be allowed to truncate data passed to *trace\_user\_event* which is longer than *maxdatasize*.

29703 The *posix\_trace\_attr\_setstreamsize()* function shall set the minimum allowed size, in bytes, in the  
 29704 *stream-min-size* attribute of the attributes object pointed to by the *attr* argument, using the size  
 29705 value supplied by the *streamsize* argument.

29706 **RETURN VALUE**

29707 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 29708 return the corresponding error number.

29709 TRL The *posix\_trace\_attr\_getlogsize()* function stores the maximum trace log allowed size in the object  
 29710 pointed to by *logsize*, if successful.

29711 The *posix\_trace\_attr\_getmaxdatasize()* function stores the maximum trace event record memory  
 29712 size in the object pointed to by *maxdatasize*, if successful.

29713 The *posix\_trace\_attr\_getmaxsystemeventsize()* function stores the maximum memory size to store  
 29714 a single system trace event in the object pointed to by *eventsize*, if successful.

29715 The *posix\_trace\_attr\_getmaxusereventsize()* function stores the maximum memory size to store a  
 29716 single user trace event in the object pointed to by *eventsize*, if successful.

29717 The *posix\_trace\_attr\_getstreamsize()* function stores the maximum trace stream allowed size in  
 29718 the object pointed to by *streamsize*, if successful.

29719 **ERRORS**

29720 These functions may fail if:

29721 [EINVAL] The value specified by one of the arguments is invalid.

29722 **EXAMPLES**

29723 None.

29724 **APPLICATION USAGE**

29725 None.

29726 **RATIONALE**

29727 None.

29728 **FUTURE DIRECTIONS**

29729 None.

29730 **SEE ALSO**29731 *posix\_trace\_attr\_init()*, *posix\_trace\_create()*, *posix\_trace\_event()*, *posix\_trace\_get\_attr()*, the Base

29732 Definitions volume of IEEE Std 1003.1-2001, &lt;sys/types.h&gt;, &lt;trace.h&gt;

29733 **CHANGE HISTORY**

29734 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

29735 **NAME**

29736        posix\_trace\_attr\_getname — retrieve and set information about a trace stream (**TRACING**)

29737 **SYNOPSIS**

29738 TRC     #include <trace.h>

```
29739 int posix_trace_attr_getname(const trace_attr_t *attr,
29740 char *tracename);
```

29741

29742 **DESCRIPTION**

29743        Refer to *posix\_trace\_attr\_getclockres()*.

29744 **NAME**

29745 `posix_trace_attr_getstreamfullpolicy` — retrieve and set the behavior of a trace stream  
29746 **(TRACING)**

29747 **SYNOPSIS**

29748 TRC `#include <trace.h>`

```
29749 int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *attr,
29750 int *streampolicy);
```

29751

29752 **DESCRIPTION**

29753 Refer to `posix_trace_attr_getinherited()`.

29754 **NAME**

29755        posix\_trace\_attr\_getstreamsize — retrieve and set trace stream size attributes (**TRACING**)

29756 **SYNOPSIS**

29757 TRC     #include <sys/types.h>

29758        #include <trace.h>

```
29759 int posix_trace_attr_getstreamsize(const trace_attr_t *restrict attr,
29760 size_t *restrict streamsize);
```

29761

29762 **DESCRIPTION**

29763        Refer to *posix\_trace\_attr\_getlogsize()*.

29764 **NAME**

29765        `posix_trace_attr_init` — initialize the trace stream attributes object (**TRACING**)

29766 **SYNOPSIS**

29767 TRC        `#include <trace.h>`

29768        `int posix_trace_attr_init(trace_attr_t *attr);`

29769

29770 **DESCRIPTION**

29771        Refer to `posix_trace_attr_destroy()`.

29772 **NAME**

29773        posix\_trace\_attr\_setinherited, posix\_trace\_attr\_setlogfullpolicy — retrieve and set the behavior  
29774        of a trace stream (**TRACING**)

29775 **SYNOPSIS**

```
29776 TRC #include <trace.h>
29777 TRC TRI int posix_trace_attr_setinherited(trace_attr_t *attr,
29778 int inheritancepolicy);
29779 TRC TRL int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
29780 int logpolicy);
29781
```

29782 **DESCRIPTION**

29783        Refer to *posix\_trace\_attr\_getinherited()*.

29784 **NAME**

29785 posix\_trace\_attr\_setlogsize, posix\_trace\_attr\_setmaxdatasize — retrieve and set trace stream  
29786 size attributes (**TRACING**)

29787 **SYNOPSIS**

29788 TRC #include <sys/types.h>

29789 #include <trace.h>

29790 TRC TRL int posix\_trace\_attr\_setlogsize(trace\_attr\_t \*attr,  
29791 size\_t logsize);

29792 TRC int posix\_trace\_attr\_setmaxdatasize(trace\_attr\_t \*attr,  
29793 size\_t maxdatasize);

29794

29795 **DESCRIPTION**

29796 Refer to *posix\_trace\_attr\_getlogsize()*.

29797 **NAME**

29798        posix\_trace\_attr\_setname — retrieve and set information about a trace stream (**TRACING**)

29799 **SYNOPSIS**

29800 TRC     #include <trace.h>

```
29801 int posix_trace_attr_setname(trace_attr_t *attr,
29802 const char *tracename);
```

29803

29804 **DESCRIPTION**

29805        Refer to *posix\_trace\_attr\_getclockres()*.

29806 **NAME**

29807 `posix_trace_attr_setstreamfullpolicy` — retrieve and set the behavior of a trace stream  
29808 **(TRACING)**

29809 **SYNOPSIS**

29810 TRC `#include <trace.h>`

```
29811 int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,
29812 int streampolicy); |
```

29813

29814 **DESCRIPTION**

29815 Refer to `posix_trace_attr_getinherited()`.

29816 **NAME**

29817        posix\_trace\_attr\_setstreamsize — retrieve and set trace stream size attributes (**TRACING**)

29818 **SYNOPSIS**

29819 TRC     #include <sys/types.h>

29820        #include <trace.h>

```
29821 int posix_trace_attr_setstreamsize(trace_attr_t *attr,
29822 size_t streamsize);
```

29823

29824 **DESCRIPTION**

29825        Refer to *posix\_trace\_attr\_getlogsize()*.

29826 **NAME**29827        posix\_trace\_clear — clear trace stream and trace log (**TRACING**)29828 **SYNOPSIS**

29829 TRC        #include &lt;sys/types.h&gt;

29830        #include &lt;trace.h&gt;

29831        int posix\_trace\_clear(trace\_id\_t trid);

29832

29833 **DESCRIPTION**

29834        The *posix\_trace\_clear()* function shall reinitialize the trace stream identified by the argument *trid* as if it were returning from the *posix\_trace\_create()* function, except that the same allocated resources shall be reused, the mapping of trace event type identifiers to trace event names shall be unchanged, and the trace stream status shall remain unchanged (that is, if it was running, it remains running and if it was suspended, it remains suspended).

29839        All trace events in the trace stream recorded before the call to *posix\_trace\_clear()* shall be lost. The *posix\_stream\_full\_status* status shall be set to `POSIX_TRACE_NOT_FULL`. There is no guarantee that all trace events that occurred during the *posix\_trace\_clear()* call are recorded; the behavior with respect to trace points that may occur during this call is unspecified.

29843 TRL        If the Trace Log option is supported and the trace stream has been created with a log, the *posix\_trace\_clear()* function shall reinitialize the trace stream with the same behavior as if the trace stream was created without the log, plus it shall reinitialize the trace log associated with the trace stream identified by the argument *trid* as if it were returning from the *posix\_trace\_create\_withlog()* function, except that the same allocated resources, for the trace log, may be reused and the associated trace stream status remains unchanged. The first trace event recorded in the trace log after the call to *posix\_trace\_clear()* shall be the same as the first trace event recorded in the active trace stream after the call to *posix\_trace\_clear()*. The *posix\_log\_full\_status* status shall be set to `POSIX_TRACE_NOT_FULL`. There is no guarantee that all trace events that occurred during the *posix\_trace\_clear()* call are recorded in the trace log; the behavior with respect to trace points that may occur during this call is unspecified. If the log full policy is `POSIX_TRACE_APPEND`, the effect of a call to this function is unspecified for the trace log associated with the trace stream identified by the *trid* argument.

29856 **RETURN VALUE**

29857        Upon successful completion, the *posix\_trace\_clear()* function shall return a value of zero. Otherwise, it shall return the corresponding error number.

29859 **ERRORS**29860        The *posix\_trace\_clear()* function shall fail if:29861        [EINVAL]        The value of the *trid* argument does not correspond to an active trace stream.29862 **EXAMPLES**

29863        None.

29864 **APPLICATION USAGE**

29865        None.

29866 **RATIONALE**

29867        None.

29868 **FUTURE DIRECTIONS**

29869        None.

29870 **SEE ALSO**

29871 *posix\_trace\_attr\_init()*, *posix\_trace\_create()*, *posix\_trace\_flush()*, *posix\_trace\_get\_attr()*, the Base  
29872 Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <trace.h>

29873 **CHANGE HISTORY**

29874 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

29875 IEEE PASC Interpretation 1003.1 #123 is applied.

29876 **NAME**29877        posix\_trace\_close, posix\_trace\_open, posix\_trace\_rewind — trace log management (**TRACING**)29878 **SYNOPSIS**

29879 TRC TRL #include &lt;trace.h&gt;

29880        int posix\_trace\_close(trace\_id\_t trid);

29881        int posix\_trace\_open(int file\_desc, trace\_id\_t \*trid);

29882        int posix\_trace\_rewind(trace\_id\_t trid);

29883

29884 **DESCRIPTION**

29885        The *posix\_trace\_close()* function shall deallocate the trace log identifier indicated by *trid*, and all  
 29886        of its associated resources. If there is no valid trace log pointed to by the *trid*, this function shall  
 29887        fail.

29888        The *posix\_trace\_open()* function shall allocate the necessary resources and establish the  
 29889        connection between a trace log identified by the *file\_desc* argument and a trace stream identifier  
 29890        identified by the object pointed to by the *trid* argument. The *file\_desc* argument should be a valid  
 29891        open file descriptor that corresponds to a trace log. The *file\_desc* argument shall be open for  
 29892        reading. The current trace event timestamp, which specifies the timestamp of the trace event  
 29893        that will be read by the next call to *posix\_trace\_getnext\_event()*, shall be set to the timestamp of  
 29894        the oldest trace event recorded in the trace log identified by *trid*.

29895        The *posix\_trace\_open()* function shall return a trace stream identifier in the variable pointed to by  
 29896        the *trid* argument, that may only be used by the following functions:

|       |                                               |                                    |
|-------|-----------------------------------------------|------------------------------------|
| 29897 | <i>posix_trace_close()</i>                    | <i>posix_trace_get_attr()</i>      |
| 29898 | <i>posix_trace_eventid_equal()</i>            | <i>posix_trace_get_status()</i>    |
| 29899 | <i>posix_trace_eventid_get_name()</i>         | <i>posix_trace_getnext_event()</i> |
| 29900 | <i>posix_trace_eventtypelist_getnext_id()</i> | <i>posix_trace_rewind()</i>        |
| 29901 | <i>posix_trace_eventtypelist_rewind()</i>     |                                    |

29902        In particular, notice that the operations normally used by a trace controller process, such as  
 29903        *posix\_trace\_start()*, *posix\_trace\_stop()*, or *posix\_trace\_shutdown()*, cannot be invoked using the  
 29904        trace stream identifier returned by the *posix\_trace\_open()* function.

29905        The *posix\_trace\_rewind()* function shall reset the current trace event timestamp, which specifies  
 29906        the timestamp of the trace event that will be read by the next call to *posix\_trace\_getnext\_event()*,  
 29907        to the timestamp of the oldest trace event recorded in the trace log identified by *trid*.

29908 **RETURN VALUE**

29909        Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 29910        return the corresponding error number.

29911        If successful, the *posix\_trace\_open()* function stores the trace stream identifier value in the object  
 29912        pointed to by *trid*.

29913 **ERRORS**

29914        The *posix\_trace\_open()* function shall fail if:

29915        [EINTR]        The operation was interrupted by a signal and thus no trace log was opened.

29916        [EINVAL]       The object pointed to by *file\_desc* does not correspond to a valid trace log.

29917        The *posix\_trace\_close()* and *posix\_trace\_rewind()* functions may fail if:

29918        [EINVAL]       The object pointed to by *trid* does not correspond to a valid trace log.

29919 **EXAMPLES**

29920 None.

29921 **APPLICATION USAGE**

29922 None.

29923 **RATIONALE**

29924 None.

29925 **FUTURE DIRECTIONS**

29926 None.

29927 **SEE ALSO**29928 *posix\_trace\_get\_attr()*, *posix\_trace\_get\_filter()*, *posix\_trace\_getnext\_event()*, the Base Definitions  
29929 volume of IEEE Std 1003.1-2001, <**trace.h**>29930 **CHANGE HISTORY**

29931 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

29932 IEEE PASC Interpretation 1003.1 #123 is applied.

## 29933 NAME

29934 posix\_trace\_create, posix\_trace\_create\_withlog, posix\_trace\_flush, posix\_trace\_shutdown —  
 29935 trace stream initialization, flush, and shutdown from a process (TRACING)

## 29936 SYNOPSIS

```
29937 TRC #include <sys/types.h>
29938 #include <trace.h>

29939 int posix_trace_create(pid_t pid,
29940 const trace_attr_t *restrict attr,
29941 trace_id_t *restrict trid);
29942 TRC TRL int posix_trace_create_withlog(pid_t pid,
29943 const trace_attr_t *restrict attr, int file_desc,
29944 trace_id_t *restrict trid);
29945 int posix_trace_flush(trace_id_t trid);
29946 TRC int posix_trace_shutdown(trace_id_t trid);
29947
```

## 29948 DESCRIPTION

29949 The *posix\_trace\_create()* function shall create an active trace stream. It allocates all the resources  
 29950 needed by the trace stream being created for tracing the process specified by *pid* in accordance  
 29951 with the *attr* argument. The *attr* argument represents the initial attributes of the trace stream and  
 29952 shall have been initialized by the function *posix\_trace\_attr\_init()* prior to the *posix\_trace\_create()*  
 29953 call. If the argument *attr* is NULL, the default attributes shall be used. The *attr* attributes object  
 29954 shall be manipulated through a set of functions described in the *posix\_trace\_attr* family of  
 29955 functions. If the attributes of the object pointed to by *attr* are modified later, the attributes of the  
 29956 trace stream shall not be affected. The *creation-time* attribute of the newly created trace stream  
 29957 shall be set to the value of the system clock, if the Timers option is not supported, or to the value  
 29958 of the CLOCK\_REALTIME clock, if the Timers option is supported.

29959 The *pid* argument represents the target process to be traced. If the process executing this  
 29960 function does not have appropriate privileges to trace the process identified by *pid*, an error shall  
 29961 be returned. If the *pid* argument is zero, the calling process shall be traced.

29962 The *posix\_trace\_create()* function shall store the trace stream identifier of the new trace stream in  
 29963 the object pointed to by the *trid* argument. This trace stream identifier shall be used in  
 29964 subsequent calls to control tracing. The *trid* argument may only be used by the following  
 29965 functions:

|       |                                               |                                         |
|-------|-----------------------------------------------|-----------------------------------------|
| 29966 | <i>posix_trace_clear()</i>                    | <i>posix_trace_getnext_event()</i>      |
| 29967 | <i>posix_trace_eventid_equal()</i>            | <i>posix_trace_shutdown()</i>           |
| 29968 | <i>posix_trace_eventid_get_name()</i>         | <i>posix_trace_start()</i>              |
| 29969 | <i>posix_trace_eventtypelist_getnext_id()</i> | <i>posix_trace_stop()</i>               |
| 29970 | <i>posix_trace_eventtypelist_rewind()</i>     | <i>posix_trace_timedgetnext_event()</i> |
| 29971 | <i>posix_trace_get_attr()</i>                 | <i>posix_trace_trid_eventid_open()</i>  |
| 29972 | <i>posix_trace_get_status()</i>               | <i>posix_trace_trygetnext_event()</i>   |

29973 TEF If the Trace Event Filter option is supported, the following additional functions may use the *trid*  
 29974 argument:

```
29975 posix_trace_get_filter() posix_trace_set_filter()
```

29976

29977 In particular, notice that the operations normally used by a trace analyzer process, such as  
 29978 *posix\_trace\_rewind()* or *posix\_trace\_close()*, cannot be invoked using the trace stream identifier  
 29979 returned by the *posix\_trace\_create()* function.

29980 TEF A trace stream shall be created in a suspended state. If the Trace Event Filter option is  
 29981 supported, its trace event type filter shall be empty.

29982 The *posix\_trace\_create()* function may be called multiple times from the same or different  
 29983 processes, with the system-wide limit indicated by the runtime invariant value  
 29984 {TRACE\_SYS\_MAX}, which has the minimum value {\_POSIX\_TRACE\_SYS\_MAX}.

29985 The trace stream identifier returned by the *posix\_trace\_create()* function in the argument pointed  
 29986 to by *trid* is valid only in the process that made the function call. If it is used from another  
 29987 process, that is a child process, in functions defined in IEEE Std 1003.1-2001, these functions shall  
 29988 return with the error [EINVAL].

29989 TRL The *posix\_trace\_create\_withlog()* function shall be equivalent to *posix\_trace\_create()*, except that it  
 29990 associates a trace log with this stream. The *file\_desc* argument shall be the file descriptor  
 29991 designating the trace log destination. The function shall fail if this file descriptor refers to a file  
 29992 with a file type that is not compatible with the log policy associated with the trace log. The list of  
 29993 the appropriate file types that are compatible with each log policy is implementation-defined.

29994 The *posix\_trace\_create\_withlog()* function shall return in the parameter pointed to by *trid* the trace  
 29995 stream identifier, which uniquely identifies the newly created trace stream, and shall be used in  
 29996 subsequent calls to control tracing. The *trid* argument may only be used by the following  
 29997 functions:

|       |                                               |                                         |
|-------|-----------------------------------------------|-----------------------------------------|
| 29998 | <i>posix_trace_clear()</i>                    | <i>posix_trace_getnext_event()</i>      |
| 29999 | <i>posix_trace_eventid_equal()</i>            | <i>posix_trace_shutdown()</i>           |
| 30000 | <i>posix_trace_eventid_get_name()</i>         | <i>posix_trace_start()</i>              |
| 30001 | <i>posix_trace_eventtypelist_getnext_id()</i> | <i>posix_trace_stop()</i>               |
| 30002 | <i>posix_trace_eventtypelist_rewind()</i>     | <i>posix_trace_timedgetnext_event()</i> |
| 30003 | <i>posix_trace_flush()</i>                    | <i>posix_trace_trid_eventid_open()</i>  |
| 30004 | <i>posix_trace_get_attr()</i>                 | <i>posix_trace_trygetnext_event()</i>   |
| 30005 | <i>posix_trace_get_status()</i>               |                                         |

30006

30007 TRL TEF If the Trace Event Filter option is supported, the following additional functions may use the *trid*  
 30008 argument:

|       |                                 |                                 |
|-------|---------------------------------|---------------------------------|
| 30009 | <i>posix_trace_get_filter()</i> | <i>posix_trace_set_filter()</i> |
|-------|---------------------------------|---------------------------------|

30010

30011 TRL In particular, notice that the operations normally used by a trace analyzer process, such as  
 30012 *posix\_trace\_rewind()* or *posix\_trace\_close()*, cannot be invoked using the trace stream identifier  
 30013 returned by the *posix\_trace\_create\_withlog()* function.

30014 The *posix\_trace\_flush()* function shall initiate a flush operation which copies the contents of the  
 30015 trace stream identified by the argument *trid* into the trace log associated with the trace stream at  
 30016 the creation time. If no trace log has been associated with the trace stream pointed to by *trid*, this  
 30017 function shall return an error. The termination of the flush operation can be polled by the  
 30018 *posix\_trace\_get\_status()* function. During the flush operation, it shall be possible to trace new  
 30019 trace events up to the point when the trace stream becomes full. After flushing is completed, the  
 30020 space used by the flushed trace events shall be available for tracing new trace events.

- 30021 If flushing the trace stream causes the resulting trace log to become full, the trace log full policy  
30022 shall be applied. If the trace *log-full-policy* attribute is set, the following occurs:
- 30023 **POSIX\_TRACE\_UNTIL\_FULL**  
30024 The trace events that have not yet been flushed shall be discarded.
- 30025 **POSIX\_TRACE\_LOOP**  
30026 The trace events that have not yet been flushed shall be written to the beginning of the trace  
30027 log, overwriting previous trace events stored there.
- 30028 **POSIX\_TRACE\_APPEND**  
30029 The trace events that have not yet been flushed shall be appended to the trace log.  
30030
- 30031 The *posix\_trace\_shutdown()* function shall stop the tracing of trace events in the trace stream  
30032 identified by *trid*, as if *posix\_trace\_stop()* had been invoked. The *posix\_trace\_shutdown()* function  
30033 shall free all the resources associated with the trace stream.
- 30034 The *posix\_trace\_shutdown()* function shall not return until all the resources associated with the  
30035 trace stream have been freed. When the *posix\_trace\_shutdown()* function returns, the *trid*  
30036 argument becomes an invalid trace stream identifier. A call to this function shall unconditionally  
30037 deallocate the resources regardless of whether all trace events have been retrieved by the  
30038 analyzer process. Any thread blocked on one of the *trace\_getnext\_event()* functions (which  
30039 specified this *trid*) before this call is unblocked with the error [EINVAL].
- 30040 If the process exits, invokes a member of the *exec* family of functions, or is terminated, the trace  
30041 streams that the process had created and that have not yet been shut down, shall be  
30042 automatically shut down as if an explicit call were made to the *posix\_trace\_shutdown()* function.
- 30043 TRL For an active trace stream with log, when the *posix\_trace\_shutdown()* function is called, all trace  
30044 events that have not yet been flushed to the trace log shall be flushed, as in the  
30045 *posix\_trace\_flush()* function, and the trace log shall be closed.
- 30046 When a trace log is closed, all the information that may be retrieved later from the trace log  
30047 through the trace interface shall have been written to the trace log. This information includes the  
30048 trace attributes, the list of trace event types (with the mapping between trace event names and  
30049 trace event type identifiers), and the trace status.
- 30050 In addition, unspecified information shall be written to the trace log to allow detection of a valid  
30051 trace log during the *posix\_trace\_open()* operation.
- 30052 The *posix\_trace\_shutdown()* function shall not return until all trace events have been flushed.
- 30053 **RETURN VALUE**
- 30054 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
30055 return the corresponding error number.
- 30056 TRL The *posix\_trace\_create()* and *posix\_trace\_create\_withlog()* functions store the trace stream  
30057 identifier value in the object pointed to by *trid*, if successful.
- 30058 **ERRORS**
- 30059 TRL The *posix\_trace\_create()* and *posix\_trace\_create\_withlog()* functions shall fail if:
- 30060 [EAGAIN] No more trace streams can be started now. {TRACE\_SYS\_MAX} has been  
30061 exceeded.
- 30062 [EINTR] The operation was interrupted by a signal. No trace stream was created.
- 30063 [EINVAL] One or more of the trace parameters specified by the *attr* parameter is invalid.

|       |          |                                                                                                                                                  |
|-------|----------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| 30064 | [ENOMEM] | The implementation does not currently have sufficient memory to create the trace stream with the specified parameters.                           |
| 30065 |          |                                                                                                                                                  |
| 30066 | [EPERM]  | The caller does not have appropriate privilege to trace the process specified by <i>pid</i> .                                                    |
| 30067 |          |                                                                                                                                                  |
| 30068 | [ESRCH]  | The <i>pid</i> argument does not refer to an existing process.                                                                                   |
| 30069 | TRL      | The <i>posix_trace_create_withlog()</i> function shall fail if:                                                                                  |
| 30070 | [EBADF]  | The <i>file_desc</i> argument is not a valid file descriptor open for writing.                                                                   |
| 30071 | [EINVAL] | The <i>file_desc</i> argument refers to a file with a file type that does not support the log policy associated with the trace log.              |
| 30072 |          |                                                                                                                                                  |
| 30073 | [ENOSPC] | No space left on device. The device corresponding to the argument <i>file_desc</i> does not contain the space required to create this trace log. |
| 30074 |          |                                                                                                                                                  |
| 30075 |          |                                                                                                                                                  |
| 30076 | TRL      | The <i>posix_trace_flush()</i> and <i>posix_trace_shutdown()</i> functions shall fail if:                                                        |
| 30077 | [EINVAL] | The value of the <i>trid</i> argument does not correspond to an active trace stream with log.                                                    |
| 30078 |          |                                                                                                                                                  |
| 30079 | [EFBIG]  | The trace log file has attempted to exceed an implementation-defined maximum file size.                                                          |
| 30080 |          |                                                                                                                                                  |
| 30081 | [ENOSPC] | No space left on device.                                                                                                                         |
| 30082 |          |                                                                                                                                                  |

30083 **EXAMPLES**

30084 None.

30085 **APPLICATION USAGE**

30086 None.

30087 **RATIONALE**

30088 None.

30089 **FUTURE DIRECTIONS**

30090 None.

30091 **SEE ALSO**

30092 *clock\_getres()*, *exec*, *posix\_trace\_attr\_init()*, *posix\_trace\_clear()*, *posix\_trace\_close()*,  
 30093 *posix\_trace\_eventid\_equal()*, *posix\_trace\_eventtypelist\_getnext\_id()*, *posix\_trace\_flush()*,  
 30094 *posix\_trace\_get\_attr()*, *posix\_trace\_get\_filter()*, *posix\_trace\_get\_status()*, *posix\_trace\_getnext\_event()*,  
 30095 *posix\_trace\_open()*, *posix\_trace\_set\_filter()*, *posix\_trace\_shutdown()*, *posix\_trace\_start()*,  
 30096 *posix\_trace\_timedgetnext\_event()*, *posix\_trace\_trid\_eventid\_open()*, *posix\_trace\_start()*, *time()*, the  
 30097 Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <trace.h>

30098 **CHANGE HISTORY**

30099 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

## 30100 NAME

30101 posix\_trace\_event, posix\_trace\_eventid\_open — trace functions for instrumenting application  
 30102 code (**TRACING**)

## 30103 SYNOPSIS

```
30104 TRC #include <sys/types.h>
```

```
30105 #include <trace.h>
```

```
30106 void posix_trace_event(trace_event_id_t event_id,
30107 const void *restrict data_ptr, size_t data_len);
30108 int posix_trace_eventid_open(const char *restrict event_name,
30109 trace_event_id_t *restrict event_id);
30110
```

## 30111 DESCRIPTION

30112 The *posix\_trace\_event()* function shall record the *event\_id* and the user data pointed to by *data\_ptr*  
 30113 in the trace stream into which the calling process is being traced and in which *event\_id* is not  
 30114 filtered out. If the total size of the user trace event data represented by *data\_len* is not greater  
 30115 than the declared maximum size for user trace event data, then the *truncation-status* attribute of  
 30116 the trace event recorded is POSIX\_TRACE\_NOT\_TRUNCATED. Otherwise, the user trace event  
 30117 data is truncated to this declared maximum size and the *truncation-status* attribute of the trace  
 30118 event recorded is POSIX\_TRACE\_TRUNCATED\_RECORD.

30119 If there is no trace stream created for the process or if the created trace stream is not running, or  
 30120 if the trace event specified by *event\_id* is filtered out in the trace stream, the *posix\_trace\_event()*  
 30121 function shall have no effect.

30122 The *posix\_trace\_eventid\_open()* function shall associate a user trace event name with a trace event  
 30123 type identifier for the calling process. The trace event name is the string pointed to by the  
 30124 argument *event\_name*. It shall have a maximum of {TRACE\_EVENT\_NAME\_MAX} characters  
 30125 (which has the minimum value {POSIX\_TRACE\_EVENT\_NAME\_MAX}). The number of user  
 30126 trace event type identifiers that can be defined for any given process is limited by the maximum  
 30127 value {TRACE\_USER\_EVENT\_MAX}, which has the minimum value  
 30128 {POSIX\_TRACE\_USER\_EVENT\_MAX}.

30129 If the Trace Inherit option is not supported, the *posix\_trace\_eventid\_open()* function shall  
 30130 associate the user trace event name pointed to by the *event\_name* argument with a trace event  
 30131 type identifier that is unique for the traced process, and is returned in the variable pointed to by  
 30132 the *event\_id* argument. If the user trace event name has already been mapped for the traced  
 30133 process, then the previously assigned trace event type identifier shall be returned. If the per-  
 30134 process user trace event name limit represented by {TRACE\_USER\_EVENT\_MAX} has been  
 30135 reached, the pre-defined POSIX\_TRACE\_UNNAMED\_USEREVENT (see Table 2-7 (on page 79))  
 30136 user trace event shall be returned.

30137 TRI If the Trace Inherit option is supported, the *posix\_trace\_eventid\_open()* function shall associate the  
 30138 user trace event name pointed to by the *event\_name* argument with a trace event type identifier  
 30139 that is unique for all the processes being traced in this same trace stream, and is returned in the  
 30140 variable pointed to by the *event\_id* argument. If the user trace event name has already been  
 30141 mapped for the traced processes, then the previously assigned trace event type identifier shall be  
 30142 returned. If the per-process user trace event name limit represented by  
 30143 {TRACE\_USER\_EVENT\_MAX} has been reached, the pre-defined  
 30144 POSIX\_TRACE\_UNNAMED\_USEREVENT (Table 2-7 (on page 79)) user trace event shall be  
 30145 returned.

30146 **Note:** The above procedure, together with the fact that multiple processes can only be traced into the  
 30147 same trace stream by inheritance, ensure that all the processes that are traced into a trace  
 30148 stream have the same mapping of trace event names to trace event type identifiers.

30149

30150 If there is no trace stream created, the *posix\_trace\_eventid\_open()* function shall store this  
30151 information for future trace streams created for this process.

30152 **RETURN VALUE**

30153 No return value is defined for the *posix\_trace\_event()* function.

30154 Upon successful completion, the *posix\_trace\_eventid\_open()* function shall return a value of zero.  
30155 Otherwise, it shall return the corresponding error number. The *posix\_trace\_eventid\_open()*  
30156 function stores the trace event type identifier value in the object pointed to by *event\_id*, if  
30157 successful.

30158 **ERRORS**

30159 The *posix\_trace\_eventid\_open()* function shall fail if:

30160 [ENAMETOOLONG]

30161 The size of the name pointed to by the *event\_name* argument was longer than  
30162 the implementation-defined value {TRACE\_EVENT\_NAME\_MAX}.

30163 **EXAMPLES**

30164 None.

30165 **APPLICATION USAGE**

30166 None.

30167 **RATIONALE**

30168 None.

30169 **FUTURE DIRECTIONS**

30170 None.

30171 **SEE ALSO**

30172 Table 2-7 (on page 79), *posix\_trace\_start()*, *posix\_trace\_trid\_eventid\_open()*, the Base Definitions  
30173 volume of IEEE Std 1003.1-2001, <sys/types.h>, <trace.h>

30174 **CHANGE HISTORY**

30175 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30176 IEEE PASC Interpretation 1003.1 #123 is applied.

30177 IEEE PASC Interpretation 1003.1 #127 is applied, correcting some editorial errors in the names of  
30178 the *posix\_trace\_eventid\_open()* function and the *event\_id* argument.

30179 **NAME**

30180 posix\_trace\_eventid\_equal, posix\_trace\_eventid\_get\_name, posix\_trace\_trid\_eventid\_open —  
 30181 manipulate the trace event type identifier (**TRACING**)

30182 **SYNOPSIS**

```
30183 TRC #include <trace.h>

30184 int posix_trace_eventid_equal(trace_id_t trid, trace_event_id_t event1,
30185 trace_event_id_t event2);
30186 int posix_trace_eventid_get_name(trace_id_t trid,
30187 trace_event_id_t event, char *event_name);
30188 TRC TEF int posix_trace_trid_eventid_open(trace_id_t trid,
30189 const char *restrict event_name,
30190 trace_event_id_t *restrict event);
30191
```

30192 **DESCRIPTION**

30193 The *posix\_trace\_eventid\_equal()* function shall compare the trace event type identifiers *event1* and  
 30194 *event2* from the same trace stream or the same trace log identified by the *trid* argument. If the  
 30195 trace event type identifiers *event1* and *event2* are from different trace streams, the return value  
 30196 shall be unspecified.

30197 The *posix\_trace\_eventid\_get\_name()* function shall return, in the argument pointed to by  
 30198 *event\_name*, the trace event name associated with the trace event type identifier identified by the  
 30199 argument *event*, for the trace stream or for the trace log identified by the *trid* argument. The  
 30200 name of the trace event shall have a maximum of {TRACE\_EVENT\_NAME\_MAX} characters  
 30201 (which has the minimum value {\_POSIX\_TRACE\_EVENT\_NAME\_MAX}). Successive calls to  
 30202 this function with the same trace event type identifier and the same trace stream identifier shall  
 30203 return the same event name.

30204 TEF The *posix\_trace\_trid\_eventid\_open()* function shall associate a user trace event name with a trace  
 30205 event type identifier for a given trace stream. The trace stream is identified by the *trid* argument,  
 30206 and it shall be an active trace stream. The trace event name is the string pointed to by the  
 30207 argument *event\_name*. It shall have a maximum of {TRACE\_EVENT\_NAME\_MAX} characters  
 30208 (which has the minimum value {\_POSIX\_TRACE\_EVENT\_NAME\_MAX}). The number of user  
 30209 trace event type identifiers that can be defined for any given process is limited by the maximum  
 30210 value {TRACE\_USER\_EVENT\_MAX}, which has the minimum value  
 30211 {\_POSIX\_TRACE\_USER\_EVENT\_MAX}.

30212 If the Trace Inherit option is not supported, the *posix\_trace\_trid\_eventid\_open()* function shall  
 30213 associate the user trace event name pointed to by the *event\_name* argument with a trace event  
 30214 type identifier that is unique for the process being traced in the trace stream identified by the *trid*  
 30215 argument, and is returned in the variable pointed to by the *event* argument. If the user trace  
 30216 event name has already been mapped for the traced process, then the previously assigned trace  
 30217 event type identifier shall be returned. If the per-process user trace event name limit represented  
 30218 by {TRACE\_USER\_EVENT\_MAX} has been reached, the pre-defined  
 30219 POSIX\_TRACE\_UNNAMED\_USEREVENT (see Table 2-7 (on page 79)) user trace event shall be  
 30220 returned.

30221 TEF TRI If the Trace Inherit option is supported, the *posix\_trace\_trid\_eventid\_open()* function shall  
 30222 associate the user trace event name pointed to by the *event\_name* argument with a trace event  
 30223 type identifier that is unique for all the processes being traced in the trace stream identified by  
 30224 the *trid* argument, and is returned in the variable pointed to by the *event* argument. If the user  
 30225 trace event name has already been mapped for the traced processes, then the previously  
 30226 assigned trace event type identifier shall be returned. If the per-process user trace event name  
 30227 limit represented by {TRACE\_USER\_EVENT\_MAX} has been reached, the pre-defined

30228            POSIX\_TRACE\_UNNAMED\_USEREVENT (see Table 2-7 (on page 79)) user trace event shall be  
 30229            returned.

30230 **RETURN VALUE**

30231 TEF        Upon successful completion, the *posix\_trace\_eventid\_get\_name()* and  
 30232            *posix\_trace\_trid\_eventid\_open()* functions shall return a value of zero. Otherwise, they shall return  
 30233            the corresponding error number.

30234            The *posix\_trace\_eventid\_equal()* function shall return a non-zero value if *event1* and *event2* are  
 30235            equal; otherwise, a value of zero shall be returned. No errors are defined. If either *event1* or  
 30236            *event2* are not valid trace event type identifiers for the trace stream specified by *trid* or if the *trid*  
 30237            is invalid, the behavior shall be unspecified.

30238            The *posix\_trace\_eventid\_get\_name()* function stores the trace event name value in the object  
 30239            pointed to by *event\_name*, if successful.

30240 TEF        The *posix\_trace\_trid\_eventid\_open()* function stores the trace event type identifier value in the  
 30241            object pointed to by *event*, if successful.

30242 **ERRORS**

30243 TEF        The *posix\_trace\_eventid\_get\_name()* and *posix\_trace\_trid\_eventid\_open()* functions shall fail if:

30244            [EINVAL]        The *trid* argument was not a valid trace stream identifier.

30245 TEF        The *posix\_trace\_trid\_eventid\_open()* function shall fail if:

30246 TEF        [ENAMETOOLONG]        The size of the name pointed to by the *event\_name* argument was longer than  
 30247            the implementation-defined value {TRACE\_EVENT\_NAME\_MAX}.  
 30248

30249            The *posix\_trace\_eventid\_get\_name()* function shall fail if:

30250            [EINVAL]        The trace event type identifier *event* was not associated with any name.

30251 **EXAMPLES**

30252            None.

30253 **APPLICATION USAGE**

30254            None.

30255 **RATIONALE**

30256            None.

30257 **FUTURE DIRECTIONS**

30258            None.

30259 **SEE ALSO**

30260            Table 2-7 (on page 79), *posix\_trace\_event()*, *posix\_trace\_getnext\_event()*, the Base Definitions  
 30261            volume of IEEE Std 1003.1-2001, <trace.h>

30262 **CHANGE HISTORY**

30263            First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30264            IEEE PASC Interpretations 1003.1 #123 and #129 are applied.

30265 **NAME**30266        posix\_trace\_eventid\_open — trace functions for instrumenting application code (**TRACING**)30267 **SYNOPSIS**

30268 TRC     #include &lt;sys/types.h&gt;

30269        #include &lt;trace.h&gt;

30270        int posix\_trace\_eventid\_open(const char \*restrict event\_name,

30271                                    trace\_event\_id\_t \*restrict event\_id);

30272

30273 **DESCRIPTION**30274        Refer to *posix\_trace\_event()*.

## 30275 NAME

30276 `posix_trace_eventset_add`, `posix_trace_eventset_del`, `posix_trace_eventset_empty`,  
 30277 `posix_trace_eventset_fill`, `posix_trace_eventset_ismember` — manipulate trace event type sets  
 30278 (TRACING)

## 30279 SYNOPSIS

```
30280 TRC TEF #include <trace.h>

30281 int posix_trace_eventset_add(trace_event_id_t event_id,
30282 trace_event_set_t *set);
30283 int posix_trace_eventset_del(trace_event_id_t event_id,
30284 trace_event_set_t *set);
30285 int posix_trace_eventset_empty(trace_event_set_t *set);
30286 int posix_trace_eventset_fill(trace_event_set_t *set, int what);
30287 int posix_trace_eventset_ismember(trace_event_id_t event_id,
30288 const trace_event_set_t *restrict set, int *restrict ismember);
30289
```

## 30290 DESCRIPTION

30291 These primitives manipulate sets of trace event types. They operate on data objects addressable  
 30292 by the application, not on the current trace event filter of any trace stream.

30293 The `posix_trace_eventset_add()` and `posix_trace_eventset_del()` functions, respectively, shall add or  
 30294 delete the individual trace event type specified by the value of the argument `event_id` to or from  
 30295 the trace event type set pointed to by the argument `set`. Adding a trace event type already in the  
 30296 set or deleting a trace event type not in the set shall not be considered an error.

30297 The `posix_trace_eventset_empty()` function shall initialize the trace event type set pointed to by  
 30298 the `set` argument such that all trace event types defined, both system and user, shall be excluded  
 30299 from the set.

30300 The `posix_trace_eventset_fill()` function shall initialize the trace event type set pointed to by the  
 30301 argument `set`, such that the set of trace event types defined by the argument `what` shall be  
 30302 included in the set. The value of the argument `what` shall consist of one of the following values,  
 30303 as defined in the `<trace.h>` header:

30304 **POSIX\_TRACE\_WOPID\_EVENTS**

30305 All the process-independent implementation-defined system trace event types are included  
 30306 in the set.

30307 **POSIX\_TRACE\_SYSTEM\_EVENTS**

30308 All the implementation-defined system trace event types are included in the set, as are those  
 30309 defined in IEEE Std 1003.1-2001.

30310 **POSIX\_TRACE\_ALL\_EVENTS**

30311 All trace event types defined, both system and user, are included in the set.

30312 Applications shall call either `posix_trace_eventset_empty()` or `posix_trace_eventset_fill()` at least  
 30313 once for each object of type `trace_event_set_t` prior to any other use of that object. If such an  
 30314 object is not initialized in this way, but is nonetheless supplied as an argument to any of the  
 30315 `posix_trace_eventset_add()`, `posix_trace_eventset_del()`, or `posix_trace_eventset_ismember()` functions,  
 30316 the results are undefined.

30317 The `posix_trace_eventset_ismember()` function shall test whether the trace event type specified by  
 30318 the value of the argument `event_id` is a member of the set pointed to by the argument `set`. The  
 30319 value returned in the object pointed to by `ismember` argument is zero if the trace event type  
 30320 identifier is not a member of the set and a value different from zero if it is a member of the set.

**30321 RETURN VALUE**

30322       Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
30323       return the corresponding error number.

**30324 ERRORS**

30325       These functions may fail if:

30326       [EINVAL]       The value of one of the arguments is invalid.

**30327 EXAMPLES**

30328       None.

**30329 APPLICATION USAGE**

30330       None.

**30331 RATIONALE**

30332       None.

**30333 FUTURE DIRECTIONS**

30334       None.

**30335 SEE ALSO**

30336       *posix\_trace\_set\_filter()*, *posix\_trace\_trid\_eventid\_open()*, the Base Definitions volume of  
30337       IEEE Std 1003.1-2001, <trace.h>

**30338 CHANGE HISTORY**

30339       First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30340 **NAME**

30341 posix\_trace\_eventtypelist\_getnext\_id, posix\_trace\_eventtypelist\_rewind — iterate over a  
30342 mapping of trace event types (**TRACING**)

30343 **SYNOPSIS**

```
30344 TRC #include <trace.h>
30345 int posix_trace_eventtypelist_getnext_id(trace_id_t trid,
30346 trace_event_id_t *restrict event, int *restrict unavailable);
30347 int posix_trace_eventtypelist_rewind(trace_id_t trid);
30348
```

30349 **DESCRIPTION**

30350 The first time *posix\_trace\_eventtypelist\_getnext\_id()* is called, the function shall return in the  
30351 variable pointed to by *event* the first trace event type identifier of the list of trace events of the  
30352 trace stream identified by the *trid* argument. Successive calls to  
30353 *posix\_trace\_eventtypelist\_getnext\_id()* return in the variable pointed to by *event* the next trace  
30354 event type identifier in that same list. Each time a trace event type identifier is successfully  
30355 written into the variable pointed to by the *event* argument, the variable pointed to by the  
30356 *unavailable* argument shall be set to zero. When no more trace event type identifiers are  
30357 available, and so none is returned, the variable pointed to by the *unavailable* argument shall be  
30358 set to a value different from zero.

30359 The *posix\_trace\_eventtypelist\_rewind()* function shall reset the next trace event type identifier to  
30360 be read to the first trace event type identifier from the list of trace events used in the trace stream  
30361 identified by *trid*.

30362 **RETURN VALUE**

30363 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
30364 return the corresponding error number.

30365 The *posix\_trace\_eventtypelist\_getnext\_id()* function stores the trace event type identifier value in  
30366 the object pointed to by *event*, if successful.

30367 **ERRORS**

30368 These functions shall fail if:

30369 [EINVAL] The *trid* argument was not a valid trace stream identifier.

30370 **EXAMPLES**

30371 None.

30372 **APPLICATION USAGE**

30373 None.

30374 **RATIONALE**

30375 None.

30376 **FUTURE DIRECTIONS**

30377 None.

30378 **SEE ALSO**

30379 *posix\_trace\_event()*, *posix\_trace\_getnext\_event()*, *posix\_trace\_trid\_eventid\_open()*, the Base  
30380 Definitions volume of IEEE Std 1003.1-2001, <trace.h>

30381 **CHANGE HISTORY**

30382 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30383 IEEE PASC Interpretations 1003.1 #123 and #129 are applied.

30384 **NAME**30385        posix\_trace\_flush — trace stream flush from a process (**TRACING**)30386 **SYNOPSIS**

30387 TRC     #include &lt;sys/types.h&gt;

30388        #include &lt;trace.h&gt;

30389 TRC TRL int posix\_trace\_flush(trace\_id\_t trid);

30390

30391 **DESCRIPTION**30392        Refer to *posix\_trace\_create()*.

30393 **NAME**

30394 posix\_trace\_get\_attr, posix\_trace\_get\_status — retrieve the trace attributes or trace status  
 30395 (TRACING)

30396 **SYNOPSIS**

```
30397 TRC #include <trace.h>
30398
30398 int posix_trace_get_attr(trace_id_t trid, trace_attr_t *attr);
30399 int posix_trace_get_status(trace_id_t trid,
30400 struct posix_trace_status_info *statusinfo);
30401
```

30402 **DESCRIPTION**

30403 The *posix\_trace\_get\_attr()* function shall copy the attributes of the active trace stream identified  
 30404 TRL by *trid* into the object pointed to by the *attr* argument. If the Trace Log option is supported, *trid*  
 30405 may represent a pre-recorded trace log.

30406 The *posix\_trace\_get\_status()* function shall return, in the structure pointed to by the *statusinfo*  
 30407 argument, the current trace status for the trace stream identified by the *trid* argument. These  
 30408 status values returned in the structure pointed to by *statusinfo* shall have been appropriately  
 30409 TRL read to ensure that the returned values are consistent. If the Trace Log option is supported and  
 30410 the *trid* argument refers to a pre-recorded trace stream, the status shall be the status of the  
 30411 completed trace stream.

30412 Each time the *posix\_trace\_get\_status()* function is used, the overrun status of the trace stream  
 30413 TRL shall be reset to POSIX\_TRACE\_NO\_OVERRUN immediately after the call completes. If the  
 30414 Trace Log option is supported, the *posix\_trace\_get\_status()* function shall behave the same as  
 30415 when the option is not supported except for the following differences:

- 30416 • If the *trid* argument refers to a trace stream with log, each time the *posix\_trace\_get\_status()*  
 30417 function is used, the log overrun status of the trace stream shall be reset to  
 30418 POSIX\_TRACE\_NO\_OVERRUN and the *flush\_error* status shall be reset to zero immediately  
 30419 after the call completes.
- 30420 • If the *trid* argument refers to a pre-recorded trace stream, the status returned shall be the  
 30421 status of the completed trace stream and the status values of the trace stream shall not be  
 30422 reset.

30424 **RETURN VALUE**

30425 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 30426 return the corresponding error number.

30427 The *posix\_trace\_get\_attr()* function stores the trace attributes in the object pointed to by *attr*, if  
 30428 successful.

30429 The *posix\_trace\_get\_status()* function stores the trace status in the object pointed to by *statusinfo*,  
 30430 if successful.

30431 **ERRORS**

30432 These functions shall fail if:

30433 [EINVAL] The trace stream argument *trid* does not correspond to a valid active trace  
 30434 stream or a valid trace log.

30435 **EXAMPLES**

30436 None.

30437 **APPLICATION USAGE**

30438 None.

30439 **RATIONALE**

30440 None.

30441 **FUTURE DIRECTIONS**

30442 None.

30443 **SEE ALSO**

30444 *posix\_trace\_attr\_destroy()*, *posix\_trace\_attr\_init()*, *posix\_trace\_create()*, *posix\_trace\_open()*, the Base  
30445 Definitions volume of IEEE Std 1003.1-2001, <trace.h>

30446 **CHANGE HISTORY**

30447 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30448 IEEE PASC Interpretation 1003.1 #123 is applied.

30449 **NAME**

30450 posix\_trace\_get\_filter, posix\_trace\_set\_filter — retrieve and set the filter of an initialized trace  
30451 stream (**TRACING**)

30452 **SYNOPSIS**

```
30453 TRC TEF #include <trace.h>
30454
30454 int posix_trace_get_filter(trace_id_t trid, trace_event_set_t *set);
30455 int posix_trace_set_filter(trace_id_t trid,
30456 const trace_event_set_t *set, int how);
30457
```

30458 **DESCRIPTION**

30459 The *posix\_trace\_get\_filter()* function shall retrieve, into the argument pointed to by *set*, the actual  
30460 trace event filter from the trace stream specified by *trid*.

30461 The *posix\_trace\_set\_filter()* function shall change the set of filtered trace event types after a trace  
30462 stream identified by the *trid* argument is created. This function may be called prior to starting  
30463 the trace stream, or while the trace stream is active. By default, if no call is made to  
30464 *posix\_trace\_set\_filter()*, all trace events shall be recorded (that is, none of the trace event types are  
30465 filtered out).

30466 If this function is called while the trace is in progress, a special system trace event,  
30467 POSIX\_TRACE\_FILTER, shall be recorded in the trace indicating both the old and the new sets  
30468 of filtered trace event types (see Table 2-4 (on page 78) and Table 2-6 (on page 79)).

30469 If the *posix\_trace\_set\_filter()* function is interrupted by a signal, an error shall be returned and the  
30470 filter shall not be changed. In this case, the state of the trace stream shall not be changed.

30471 The value of the argument *how* indicates the manner in which the set is to be changed and shall  
30472 have one of the following values, as defined in the **<trace.h>** header:

30473 POSIX\_TRACE\_SET\_EVENTSET

30474 The resulting set of trace event types to be filtered shall be the trace event type set pointed  
30475 to by the argument *set*.

30476 POSIX\_TRACE\_ADD\_EVENTSET

30477 The resulting set of trace event types to be filtered shall be the union of the current set and  
30478 the trace event type set pointed to by the argument *set*.

30479 POSIX\_TRACE\_SUB\_EVENTSET

30480 The resulting set of trace event types to be filtered shall be all trace event types in the  
30481 current set that are not in the set pointed to by the argument *set*; that is, remove each  
30482 element of the specified set from the current filter.

30483 **RETURN VALUE**

30484 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
30485 return the corresponding error number.

30486 The *posix\_trace\_get\_filter()* function stores the set of filtered trace event types in *set*, if successful.

30487 **ERRORS**

30488 These functions shall fail if:

30489 [EINVAL] The value of the *trid* argument does not correspond to an active trace stream  
30490 or the value of the argument pointed to by *set* is invalid.

30491 [EINTR] The operation was interrupted by a signal.

30492 **EXAMPLES**

30493 None.

30494 **APPLICATION USAGE**

30495 None.

30496 **RATIONALE**

30497 None.

30498 **FUTURE DIRECTIONS**

30499 None.

30500 **SEE ALSO**30501 Table 2-4 (on page 78), Table 2-6 (on page 79), *posix\_trace\_eventset\_add()*, the Base Definitions  
30502 volume of IEEE Std 1003.1-2001, <**trace.h**>30503 **CHANGE HISTORY**

30504 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30505 IEEE PASC Interpretation 1003.1 #123 is applied.

30506 **NAME**

30507        posix\_trace\_get\_status — retrieve the trace status (**TRACING**)

30508 **SYNOPSIS**

30509 TRC     #include <trace.h>

```
30510 int posix_trace_get_status(trace_id_t trid,
30511 struct posix_trace_status_info *statusinfo);
30512
```

30513 **DESCRIPTION**

30514        Refer to *posix\_trace\_get\_attr()*.

30515 **NAME**

30516 posix\_trace\_getnext\_event, posix\_trace\_timedgetnext\_event, posix\_trace\_trygetnext\_event —  
 30517 retrieve a trace event (**TRACING**)

30518 **SYNOPSIS**

```
30519 TRC #include <sys/types.h>
30520 #include <trace.h>

30521 int posix_trace_getnext_event(trace_id_t trid,
30522 struct posix_trace_event_info *restrict event,
30523 void *restrict data, size_t num_bytes,
30524 size_t *restrict data_len, int *restrict unavailable);
30525 TRC TMO int posix_trace_timedgetnext_event(trace_id_t trid,
30526 struct posix_trace_event_info *restrict event,
30527 void *restrict data, size_t num_bytes,
30528 size_t *restrict data_len, int *restrict unavailable,
30529 const struct timespec *restrict abs_timeout);
30530 TRC int posix_trace_trygetnext_event(trace_id_t trid,
30531 struct posix_trace_event_info *restrict event,
30532 void *restrict data, size_t num_bytes,
30533 size_t *restrict data_len, int *restrict unavailable);
30534
```

30535 **DESCRIPTION**

30536 The *posix\_trace\_getnext\_event()* function shall report a recorded trace event either from an active  
 30537 TRL trace stream without log or a pre-recorded trace stream identified by the *trid* argument. The  
 30538 *posix\_trace\_trygetnext\_event()* function shall report a recorded trace event from an active trace  
 30539 stream without log identified by the *trid* argument.

30540 The trace event information associated with the recorded trace event shall be copied by the  
 30541 function into the structure pointed to by the argument *event* and the data associated with the  
 30542 trace event shall be copied into the buffer pointed to by the *data* argument.

30543 The *posix\_trace\_getnext\_event()* function shall block if the *trid* argument identifies an active trace  
 30544 stream and there is currently no trace event ready to be retrieved. When returning, if a recorded  
 30545 trace event was reported, the variable pointed to by the *unavailable* argument shall be set to zero.  
 30546 Otherwise, the variable pointed to by the *unavailable* argument shall be set to a value different  
 30547 from zero.

30548 TMO The *posix\_trace\_timedgetnext\_event()* function shall attempt to get another trace event from an  
 30549 active trace stream without log, as in the *posix\_trace\_getnext\_event()* function. However, if no  
 30550 trace event is available from the trace stream, the implied wait shall be terminated when the  
 30551 timeout specified by the argument *abs\_timeout* expires, and the function shall return the error  
 30552 [ETIMEDOUT].

30553 The timeout shall expire when the absolute time specified by *abs\_timeout* passes, as measured by  
 30554 the clock upon which timeouts are based (that is, when the value of that clock equals or exceeds  
 30555 *abs\_timeout*), or if the absolute time specified by *abs\_timeout* has already passed at the time of the  
 30556 call.

30557 TMO TMR If the Timers option is supported, the timeout shall be based on the CLOCK\_REALTIME clock;  
 30558 if the Timers option is not supported, the timeout shall be based on the system clock as returned  
 30559 by the *time()* function. The resolution of the timeout shall be the resolution of the clock on which  
 30560 it is based. The **timespec** data type is defined in the **<time.h>** header.

30561 TMO Under no circumstance shall the function fail with a timeout if a trace event is immediately  
 30562 available from the trace stream. The validity of the *abs\_timeout* argument need not be checked if

30563 a trace event is immediately available from the trace stream.

30564 The behavior of this function for a pre-recorded trace stream is unspecified.

30565 TRL The *posix\_trace\_trygetnext\_event()* function shall not block. This function shall return an error if  
 30566 the *trid* argument identifies a pre-recorded trace stream. If a recorded trace event was reported,  
 30567 the variable pointed to by the *unavailable* argument shall be set to zero. Otherwise, if no trace  
 30568 event was reported, the variable pointed to by the *unavailable* argument shall be set to a value  
 30569 different from zero.

30570 The argument *num\_bytes* shall be the size of the buffer pointed to by the *data* argument. The  
 30571 argument *data\_len* reports to the application the length in bytes of the data record just  
 30572 transferred. If *num\_bytes* is greater than or equal to the size of the data associated with the trace  
 30573 event pointed to by the *event* argument, all the recorded data shall be transferred. In this case, the  
 30574 *truncation-status* member of the trace event structure shall be either  
 30575 POSIX\_TRACE\_NOT\_TRUNCATED, if the trace event data was recorded without truncation  
 30576 while tracing, or POSIX\_TRACE\_TRUNCATED\_RECORD, if the trace event data was truncated  
 30577 when it was recorded. If the *num\_bytes* argument is less than the length of recorded trace event  
 30578 data, the data transferred shall be truncated to a length of *num\_bytes*, the value stored in the  
 30579 variable pointed to by *data\_len* shall be equal to *num\_bytes*, and the *truncation-status* member of  
 30580 the *event* structure argument shall be set to POSIX\_TRACE\_TRUNCATED\_READ (see the  
 30581 **posix\_trace\_event\_info** structure defined in <trace.h>).

30582 The report of a trace event shall be sequential starting from the oldest recorded trace event. Trace  
 30583 events shall be reported in the order in which they were generated, up to an implementation-  
 30584 defined time resolution that causes the ordering of trace events occurring very close to each  
 30585 other to be unknown. Once reported, a trace event cannot be reported again from an active trace  
 30586 stream. Once a trace event is reported from an active trace stream without log, the trace stream  
 30587 shall make the resources associated with that trace event available to record future generated  
 30588 trace events.

30589 **RETURN VALUE**

30590 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 30591 return the corresponding error number.

30592 If successful, these functions store:

- 30593 • The recorded trace event in the object pointed to by *event*
- 30594 • The trace event information associated with the recorded trace event in the object pointed to  
 30595 by *data*
- 30596 • The length of this trace event information in the object pointed to by *data\_len*
- 30597 • The value of zero in the object pointed to by *unavailable*

30598 **ERRORS**

30599 These functions shall fail if:

30600 [EINVAL] The trace stream identifier argument *trid* is invalid.

30601 The *posix\_trace\_getnext\_event()* and *posix\_trace\_timedgetnext\_event()* functions shall fail if:

30602 [EINTR] The operation was interrupted by a signal, and so the call had no effect.

30603 The *posix\_trace\_trygetnext\_event()* function shall fail if:

30604 [EINVAL] The trace stream identifier argument *trid* does not correspond to an active  
 30605 trace stream.

- 30606 TMO The *posix\_trace\_timedgetnext\_event()* function shall fail if:
- 30607 [EINVAL] There is no trace event immediately available from the trace stream, and the  
30608 *timeout* argument is invalid.
- 30609 [ETIMEDOUT] No trace event was available from the trace stream before the specified  
30610 *timeout* *timeout* expired.  
30611
- 30612 **EXAMPLES**
- 30613 None.
- 30614 **APPLICATION USAGE**
- 30615 None.
- 30616 **RATIONALE**
- 30617 None.
- 30618 **FUTURE DIRECTIONS**
- 30619 None.
- 30620 **SEE ALSO**
- 30621 *posix\_trace\_create()*, *posix\_trace\_open()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
30622 *<sys/types.h>*, *<trace.h>*
- 30623 **CHANGE HISTORY**
- 30624 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.
- 30625 IEEE PASC Interpretation 1003.1 #123 is applied.

30626 **NAME**

30627        posix\_trace\_open, posix\_trace\_rewind — trace log management (**TRACING**)

30628 **SYNOPSIS**

30629 TRC TRL #include <trace.h>

30630        int posix\_trace\_open(int *file\_desc*, trace\_id\_t \**trid*);

30631        int posix\_trace\_rewind(trace\_id\_t *trid*);

30632

30633 **DESCRIPTION**

30634        Refer to *posix\_trace\_close()*.

30635 **NAME**

30636        posix\_trace\_set\_filter — set filter of an initialized trace stream (**TRACING**)

30637 **SYNOPSIS**

30638 TRC TEF #include <trace.h>

```
30639 int posix_trace_set_filter(trace_id_t trid,
30640 const trace_event_set_t *set, int how);
```

30641

30642 **DESCRIPTION**

30643        Refer to *posix\_trace\_get\_filter()*.

30644 **NAME**

30645        posix\_trace\_shutdown — trace stream shutdown from a process (**TRACING**)

30646 **SYNOPSIS**

30647 TRC     #include <sys/types.h>

30648        #include <trace.h>

30649        int posix\_trace\_shutdown(trace\_id\_t *trid*);

30650

30651 **DESCRIPTION**

30652        Refer to *posix\_trace\_create()*.

30653 **NAME**30654            posix\_trace\_start, posix\_trace\_stop — trace start and stop (**TRACING**)30655 **SYNOPSIS**

30656 TRC        #include &lt;trace.h&gt;

30657               int posix\_trace\_start(trace\_id\_t trid);

30658               int posix\_trace\_stop (trace\_id\_t trid);

30659

30660 **DESCRIPTION**30661            The *posix\_trace\_start()* and *posix\_trace\_stop()* functions, respectively, shall start and stop the  
30662            trace stream identified by the argument *trid*.30663            The effect of calling the *posix\_trace\_start()* function shall be recorded in the trace stream as the  
30664            POSIX\_TRACE\_START system trace event and the status of the trace stream shall become  
30665            POSIX\_TRACE\_RUNNING. If the trace stream is in progress when this function is called, the  
30666            POSIX\_TRACE\_START system trace event shall not be recorded and the trace stream shall  
30667            continue to run. If the trace stream is full, the POSIX\_TRACE\_START system trace event shall  
30668            not be recorded and the status of the trace stream shall not be changed.30669            The effect of calling the *posix\_trace\_stop()* function shall be recorded in the trace stream as the  
30670            POSIX\_TRACE\_STOP system trace event and the status of the trace stream shall become  
30671            POSIX\_TRACE\_SUSPENDED. If the trace stream is suspended when this function is called, the  
30672            POSIX\_TRACE\_STOP system trace event shall not be recorded and the trace stream shall remain  
30673            suspended. If the trace stream is full, the POSIX\_TRACE\_STOP system trace event shall not be  
30674            recorded and the status of the trace stream shall not be changed.30675 **RETURN VALUE**30676            Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
30677            return the corresponding error number.30678 **ERRORS**

30679            These functions shall fail if:

30680            [EINVAL]            The value of the argument *trid* does not correspond to an active trace stream  
30681            and thus no trace stream was started or stopped.30682            [EINTR]            The operation was interrupted by a signal and thus the trace stream was not  
30683            necessarily started or stopped.30684 **EXAMPLES**

30685            None.

30686 **APPLICATION USAGE**

30687            None.

30688 **RATIONALE**

30689            None.

30690 **FUTURE DIRECTIONS**

30691            None.

30692 **SEE ALSO**30693            *posix\_trace\_create()*, the Base Definitions volume of IEEE Std 1003.1-2001, <trace.h>

30694 **CHANGE HISTORY**

30695           First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30696           IEEE PASC Interpretation 1003.1 #123 is applied.

30697 **NAME**30698        posix\_trace\_timedgetnext\_event, — retrieve a trace event (**TRACING**)30699 **SYNOPSIS**

30700 TRC TMO #include &lt;sys/types.h&gt;

30701        #include &lt;trace.h&gt;

```
30702 int posix_trace_timedgetnext_event(trace_id_t trid,
30703 struct posix_trace_event_info *restrict event,
30704 void *restrict data, size_t num_bytes,
30705 size_t *restrict data_len, int *restrict unavailable,
30706 const struct timespec *restrict abs_timeout);
30707
```

30708 **DESCRIPTION**30709        Refer to *posix\_trace\_getnext\_event()*.

30710 **NAME**

30711        posix\_trace\_trid\_eventid\_open — open a trace event type identifier (**TRACING**)

30712 **SYNOPSIS**

30713 TRC TEF #include <trace.h>

```
30714 int posix_trace_trid_eventid_open(trace_id_t trid,
30715 const char *restrict event_name,
30716 trace_event_id_t *restrict event);
```

30717

30718 **DESCRIPTION**

30719        Refer to *posix\_trace\_eventid\_equal()*.

30720 **NAME**30721        posix\_trace\_trygetnext\_event — retrieve a trace event (**TRACING**)30722 **SYNOPSIS**

30723 TRC        #include &lt;sys/types.h&gt;

30724        #include &lt;trace.h&gt;

```
30725 int posix_trace_trygetnext_event(trace_id_t trid,
30726 struct posix_trace_event_info *restrict event,
30727 void *restrict data, size_t num_bytes,
30728 size_t *restrict data_len, int *restrict unavailable);
```

30729

30730 **DESCRIPTION**30731        Refer to *posix\_trace\_getnext\_event()*.

## 30732 NAME

30733 posix\_typed\_mem\_get\_info — query typed memory information (**ADVANCED REALTIME**)

## 30734 SYNOPSIS

30735 TYM `#include <sys/mman.h>`

```
30736 int posix_typed_mem_get_info(int fildes,
30737 struct posix_typed_mem_info *info);
```

30738

## 30739 DESCRIPTION

30740 The *posix\_typed\_mem\_get\_info()* function shall return, in the *posix\_tmi\_length* field of the  
 30741 **posix\_typed\_mem\_info** structure pointed to by *info*, the maximum length which may be  
 30742 successfully allocated by the typed memory object designated by *fildes*. This maximum length  
 30743 shall take into account the flag `POSIX_TYPED_MEM_ALLOCATE` or  
 30744 `POSIX_TYPED_MEM_ALLOCATE_CONTIG` specified when the typed memory object  
 30745 represented by *fildes* was opened. The maximum length is dynamic; therefore, the value returned  
 30746 is valid only while the current mapping of the corresponding typed memory pool remains  
 30747 unchanged.

30748 If *fildes* represents a typed memory object opened with neither the  
 30749 `POSIX_TYPED_MEM_ALLOCATE` flag nor the `POSIX_TYPED_MEM_ALLOCATE_CONTIG`  
 30750 flag specified, the returned value of *info*->*posix\_tmi\_length* is unspecified.

30751 The *posix\_typed\_mem\_get\_info()* function may return additional implementation-defined  
 30752 information in other fields of the **posix\_typed\_mem\_info** structure pointed to by *info*.

30753 If the memory object specified by *fildes* is not a typed memory object, then the behavior of this  
 30754 function is undefined.

## 30755 RETURN VALUE

30756 Upon successful completion, the *posix\_typed\_mem\_get\_info()* function shall return zero;  
 30757 otherwise, the corresponding error status value shall be returned.

## 30758 ERRORS

30759 The *posix\_typed\_mem\_get\_info()* function shall fail if:

30760 [EBADF] The *fildes* argument is not a valid open file descriptor.

30761 [ENODEV] The *fildes* argument is not connected to a memory object supported by this  
 30762 function.

30763 This function shall not return an error code of [EINTR].

## 30764 EXAMPLES

30765 None.

## 30766 APPLICATION USAGE

30767 None.

## 30768 RATIONALE

30769 An application that needs to allocate a block of typed memory with length dependent upon the  
 30770 amount of memory currently available must either query the typed memory object to obtain the  
 30771 amount available, or repeatedly invoke *mmap()* attempting to guess an appropriate length.  
 30772 While the latter method is existing practice with *malloc()*, it is awkward and imprecise. The  
 30773 *posix\_typed\_mem\_get\_info()* function allows an application to immediately determine available  
 30774 memory. This is particularly important for typed memory objects that may in some cases be  
 30775 scarce resources. Note that when a typed memory pool is a shared resource, some form of  
 30776 mutual-exclusion or synchronization may be required while typed memory is being queried and

30777 allocated to prevent race conditions.

30778 The existing *fstat()* function is not suitable for this purpose. We realize that implementations  
30779 may wish to provide other attributes of typed memory objects (for example, alignment  
30780 requirements, page size, and so on). The *fstat()* function returns a structure which is not  
30781 extensible and, furthermore, contains substantial information that is inappropriate for typed  
30782 memory objects.

30783 **FUTURE DIRECTIONS**

30784 None.

30785 **SEE ALSO**

30786 *fstat()*, *mmap()*, *posix\_typed\_mem\_open()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
30787 <sys/mman.h>

30788 **CHANGE HISTORY**

30789 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

30790 **NAME**

30791 posix\_typed\_mem\_open — open a typed memory object (**ADVANCED REALTIME**)

30792 **SYNOPSIS**

30793 TYM `#include <sys/mman.h>`

30794 `int posix_typed_mem_open(const char *name, int oflag, int tflag);`

30795

30796 **DESCRIPTION**

30797 The *posix\_typed\_mem\_open()* function shall establish a connection between the typed memory  
 30798 object specified by the string pointed to by *name* and a file descriptor. It shall create an open file  
 30799 description that refers to the typed memory object and a file descriptor that refers to that open  
 30800 file description. The file descriptor is used by other functions to refer to that typed memory  
 30801 object. It is unspecified whether the name appears in the file system and is visible to other  
 30802 functions that take pathnames as arguments. The *name* argument shall conform to the  
 30803 construction rules for a pathname. If *name* begins with the slash character, then processes calling  
 30804 *posix\_typed\_mem\_open()* with the same value of *name* shall refer to the same typed memory  
 30805 object. If *name* does not begin with the slash character, the effect is implementation-defined. The  
 30806 interpretation of slash characters other than the leading slash character in *name* is  
 30807 implementation-defined.

30808 Each typed memory object supported in a system shall be identified by a name which specifies  
 30809 not only its associated typed memory pool, but also the path or port by which it is accessed. That  
 30810 is, the same typed memory pool accessed via several different ports shall have several different  
 30811 corresponding names. The binding between names and typed memory objects is established in  
 30812 an implementation-defined manner. Unlike shared memory objects, there is no way within  
 30813 IEEE Std 1003.1-2001 for a program to create a typed memory object.

30814 The value of *tflag* shall determine how the typed memory object behaves when subsequently  
 30815 mapped by calls to *mmap()*. At most, one of the following flags defined in `<sys/mman.h>` may  
 30816 be specified:

30817 POSIX\_TYPED\_MEM\_ALLOCATE

30818 Allocate on *mmap()*.

30819 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG

30820 Allocate contiguously on *mmap()*.

30821 POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE

30822 Map on *mmap()*, without affecting allocatability.

30823 If *tflag* has the flag POSIX\_TYPED\_MEM\_ALLOCATE specified, any subsequent call to *mmap()*  
 30824 using the returned file descriptor shall result in allocation and mapping of typed memory from  
 30825 the specified typed memory pool. The allocated memory may be a contiguous previously  
 30826 unallocated area of the typed memory pool or several non-contiguous previously unallocated  
 30827 areas (mapped to a contiguous portion of the process address space). If *tflag* has the flag  
 30828 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG specified, any subsequent call to *mmap()* using the  
 30829 returned file descriptor shall result in allocation and mapping of a single contiguous previously  
 30830 unallocated area of the typed memory pool (also mapped to a contiguous portion of the process  
 30831 address space). If *tflag* has none of the flags POSIX\_TYPED\_MEM\_ALLOCATE or  
 30832 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG specified, any subsequent call to *mmap()* using the  
 30833 returned file descriptor shall map an application-chosen area from the specified typed memory  
 30834 pool such that this mapped area becomes unavailable for allocation until unmapped by all  
 30835 processes. If *tflag* has the flag POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE specified, any  
 30836 subsequent call to *mmap()* using the returned file descriptor shall map an application-chosen  
 30837 area from the specified typed memory pool without an effect on the availability of that area for

30838 allocation; that is, mapping such an object leaves each byte of the mapped area unallocated if it  
 30839 was unallocated prior to the mapping or allocated if it was allocated prior to the mapping. The  
 30840 appropriate privilege to specify the POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE flag is  
 30841 implementation-defined.

30842 If successful, *posix\_typed\_mem\_open()* shall return a file descriptor for the typed memory object  
 30843 that is the lowest numbered file descriptor not currently open for that process. The open file  
 30844 description is new, and therefore the file descriptor shall not share it with any other processes. It  
 30845 is unspecified whether the file offset is set. The FD\_CLOEXEC file descriptor flag associated  
 30846 with the new file descriptor shall be cleared.

30847 The behavior of *msync()*, *ftruncate()*, and all file operations other than *mmap()*,  
 30848 *posix\_mem\_offset()*, *posix\_typed\_mem\_get\_info()*, *fstat()*, *dup()*, *dup2()*, and *close()*, is unspecified  
 30849 when passed a file descriptor connected to a typed memory object by this function.

30850 The file status flags of the open file description shall be set according to the value of *oflag*.  
 30851 Applications shall specify exactly one of the three access mode values described below and  
 30852 defined in the `<fcntl.h>` header, as the value of *oflag*.

30853 O\_RDONLY      Open for read access only.  
 30854 O\_WRONLY      Open for write access only.  
 30855 O\_RDWR        Open for read or write access.

#### 30856 RETURN VALUE

30857 Upon successful completion, the *posix\_typed\_mem\_open()* function shall return a non-negative  
 30858 integer representing the lowest numbered unused file descriptor. Otherwise, it shall return `-1`  
 30859 and set *errno* to indicate the error.

#### 30860 ERRORS

30861 The *posix\_typed\_mem\_open()* function shall fail if:

30862 [EACCES]      The typed memory object exists and the permissions specified by *oflag* are  
 30863 denied.

30864 [EINTR]        The *posix\_typed\_mem\_open()* operation was interrupted by a signal.

30865 [EINVAL]       The flags specified in *tflag* are invalid (more than one of  
 30866 POSIX\_TYPED\_MEM\_ALLOCATE,  
 30867 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG, or  
 30868 POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE is specified).

30869 [EMFILE]       Too many file descriptors are currently in use by this process.

30870 [ENAMETOOLONG]   The length of the *name* argument exceeds `{PATH_MAX}` or a pathname  
 30871 component is longer than `{NAME_MAX}`.  
 30872

30873 [ENFILE]       Too many file descriptors are currently open in the system.

30874 [ENOENT]       The named typed memory object does not exist.

30875 [EPERM]        The caller lacks the appropriate privilege to specify the flag  
 30876 POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE in argument *tflag*.

30877 **EXAMPLES**

30878           None.

30879 **APPLICATION USAGE**

30880           None.

30881 **RATIONALE**

30882           None.

30883 **FUTURE DIRECTIONS**

30884           None.

30885 **SEE ALSO**

30886           *close()*, *dup()*, *exec*, *fcntl()*, *fstat()*, *ftruncate()*, *mmap()*, *msync()*, *posix\_mem\_offset()*,  
30887           *posix\_typed\_mem\_get\_info()*, *umask()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
30888           <fcntl.h>, <sys/mman.h>

30889 **CHANGE HISTORY**

30890           First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

30891 **NAME**

30892 pow, powf, powl — power function

30893 **SYNOPSIS**

30894 #include &lt;math.h&gt;

30895 double pow(double x, double y);

30896 float powf(float x, float y);

30897 long double powl(long double x, long double y);

30898 **DESCRIPTION**

30899 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 30900 conflict between the requirements described here and the ISO C standard is unintentional. This  
 30901 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

30902 These functions shall compute the value of  $x$  raised to the power  $y$ ,  $x^y$ . If  $x$  is negative, the  
 30903 application shall ensure that  $y$  is an integer value.

30904 An application wishing to check for error situations should set *errno* to zero and call  
 30905 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 30906 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 30907 zero, an error has occurred.

30908 **RETURN VALUE**30909 Upon successful completion, these functions shall return the value of  $x$  raised to the power  $y$ .

30910 **MX** For finite values of  $x < 0$ , and finite non-integer values of  $y$ , a domain error shall occur and either  
 30911 a NaN (if representable), or an implementation-defined value shall be returned.

30912 If the correct value would cause overflow, a range error shall occur and *pow()*, *powf()*, and  
 30913 *powl()* shall return  $\pm$ HUGE\_VAL,  $\pm$ HUGE\_VALF, and  $\pm$ HUGE\_VALL, respectively, with the  
 30914 same sign as the correct value of the function.

30915 If the correct value would cause underflow, and is not representable, a range error may occur,  
 30916 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.

30917 **MX** If  $x$  or  $y$  is a NaN, a NaN shall be returned (unless specified elsewhere in this description).30918 For any value of  $y$  (including NaN), if  $x$  is +1, 1.0 shall be returned.30919 For any value of  $x$  (including NaN), if  $y$  is  $\pm 0$ , 1.0 shall be returned.30920 For any odd integer value of  $y > 0$ , if  $x$  is  $\pm 0$ ,  $\pm 0$  shall be returned.30921 For  $y > 0$  and not an odd integer, if  $x$  is  $\pm 0$ , +0 shall be returned.30922 If  $x$  is  $-1$ , and  $y$  is  $\pm$ Inf, 1.0 shall be returned.30923 For  $|x| < 1$ , if  $y$  is  $-$ Inf, +Inf shall be returned.30924 For  $|x| > 1$ , if  $y$  is  $-$ Inf, +0 shall be returned.30925 For  $|x| < 1$ , if  $y$  is +Inf, +0 shall be returned.30926 For  $|x| > 1$ , if  $y$  is +Inf, +Inf shall be returned.30927 For  $y$  an odd integer  $< 0$ , if  $x$  is  $-$ Inf,  $-0$  shall be returned.30928 For  $y < 0$  and not an odd integer, if  $x$  is  $-$ Inf, +0 shall be returned.30929 For  $y$  an odd integer  $> 0$ , if  $x$  is  $-$ Inf,  $-$ Inf shall be returned.30930 For  $y > 0$  and not an odd integer, if  $x$  is  $-$ Inf, +Inf shall be returned.

- 30931 For  $y < 0$ , if  $x$  is  $+\text{Inf}$ ,  $+0$  shall be returned.
- 30932 For  $y > 0$ , if  $x$  is  $+\text{Inf}$ ,  $+\text{Inf}$  shall be returned.
- 30933 For  $y$  an odd integer  $< 0$ , if  $x$  is  $\pm 0$ , a pole error shall occur and  $\pm\text{HUGE\_VAL}$ ,  $\pm\text{HUGE\_VALF}$ , and  
30934  $\pm\text{HUGE\_VALL}$  shall be returned for  $\text{pow}()$ ,  $\text{powf}()$ , and  $\text{powl}()$ , respectively.
- 30935 For  $y < 0$  and not an odd integer, if  $x$  is  $\pm 0$ , a pole error shall occur and  $\text{HUGE\_VAL}$ ,  
30936  $\text{HUGE\_VALF}$ , and  $\text{HUGE\_VALL}$  shall be returned for  $\text{pow}()$ ,  $\text{powf}()$ , and  $\text{powl}()$ , respectively.
- 30937 If the correct value would cause underflow, and is representable, a range error may occur and  
30938 the correct value shall be returned.

**30939 ERRORS**

- 30940 These functions shall fail if:
- 30941 Domain Error The value of  $x$  is negative and  $y$  is a finite non-integer.
- 30942 If the integer expression (`math_errhandling` & `MATH_ERRNO`) is non-zero,  
30943 then *errno* shall be set to `[EDOM]`. If the integer expression (`math_errhandling`  
30944 & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception  
30945 shall be raised.
- 30946 MX Pole Error The value of  $x$  is zero and  $y$  is negative.
- 30947 If the integer expression (`math_errhandling` & `MATH_ERRNO`) is non-zero,  
30948 then *errno* shall be set to `[ERANGE]`. If the integer expression  
30949 (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the divide-by-  
30950 zero floating-point exception shall be raised.
- 30951 Range Error The result overflows.
- 30952 If the integer expression (`math_errhandling` & `MATH_ERRNO`) is non-zero,  
30953 then *errno* shall be set to `[ERANGE]`. If the integer expression  
30954 (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the overflow  
30955 floating-point exception shall be raised.
- 30956 These functions may fail if:
- 30957 Range Error The result underflows.
- 30958 If the integer expression (`math_errhandling` & `MATH_ERRNO`) is non-zero,  
30959 then *errno* shall be set to `[ERANGE]`. If the integer expression  
30960 (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the underflow  
30961 floating-point exception shall be raised.

**30962 EXAMPLES**

30963 None.

**30964 APPLICATION USAGE**

30965 On error, the expressions (`math_errhandling` & `MATH_ERRNO`) and (`math_errhandling` &  
30966 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

**30967 RATIONALE**

30968 None.

**30969 FUTURE DIRECTIONS**

30970 None.

30971 **SEE ALSO**

30972 *exp()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
30973 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

30974 **CHANGE HISTORY**

30975 First released in Issue 1. Derived from Issue 1 of the SVID.

30976 **Issue 5**

30977 The DESCRIPTION is updated to indicate how an application should check for an error. This  
30978 text was previously published in the APPLICATION USAGE section.

30979 **Issue 6**

30980 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

30981 The *powf()* and *powl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

30982 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
30983 revised to align with the ISO/IEC 9899:1999 standard.

30984 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
30985 marked.

30986 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/42 is applied, correcting the third |  
30987 paragraph in the RETURN VALUE section. |

30988 **NAME**

30989       

```
pread — read from a file
```

30990 **SYNOPSIS**

30991 xSI       

```
#include <unistd.h>
```

30992       

```
ssize_t pread(int fd, void *buf, size_t nbyte, off_t offset);
```

30993

30994 **DESCRIPTION**

30995       Refer to *read()*.

30996 **NAME**

30997       printf — print formatted output

30998 **SYNOPSIS**

30999       #include &lt;stdio.h&gt;

31000       int printf(const char \*restrict *format*, ...);31001 **DESCRIPTION**31002       Refer to *fprintf()*.

## 31003 NAME

31004 pselect, select — synchronous I/O multiplexing

## 31005 SYNOPSIS

```

31006 #include <sys/select.h>

31007 int pselect(int nfds, fd_set *restrict readfds,
31008 fd_set *restrict writefds, fd_set *restrict errorfds,
31009 const struct timespec *restrict timeout,
31010 const sigset_t *restrict sigmask);
31011 int select(int nfds, fd_set *restrict readfds,
31012 fd_set *restrict writefds, fd_set *restrict errorfds,
31013 struct timeval *restrict timeout);
31014 void FD_CLR(int fd, fd_set *fdset);
31015 int FD_ISSET(int fd, fd_set *fdset);
31016 void FD_SET(int fd, fd_set *fdset);
31017 void FD_ZERO(fd_set *fdset);

```

## 31018 DESCRIPTION

31019 The *pselect()* function shall examine the file descriptor sets whose addresses are passed in the  
 31020 *readfds*, *writefds*, and *errorfds* parameters to see whether some of their descriptors are ready for  
 31021 reading, are ready for writing, or have an exceptional condition pending, respectively.

31022 The *select()* function shall be equivalent to the *pselect()* function, except as follows:

- 31023 • For the *select()* function, the timeout period is given in seconds and microseconds in an  
 31024 argument of type **struct timeval**, whereas for the *pselect()* function the timeout period is  
 31025 given in seconds and nanoseconds in an argument of type **struct timespec**.
- 31026 • The *select()* function has no *sigmask* argument; it shall behave as *pselect()* does when *sigmask*  
 31027 is a null pointer.
- 31028 • Upon successful completion, the *select()* function may modify the object pointed to by the  
 31029 *timeout* argument.

31030 The *pselect()* and *select()* functions shall support regular files, terminal and pseudo-terminal  
 31031 XSR devices, **STREAMS**-based files, FIFOs, pipes, and sockets. The behavior of *pselect()* and *select()*  
 31032 on file descriptors that refer to other types of file is unspecified.

31033 The *nfds* argument specifies the range of descriptors to be tested. The first *nfds* descriptors shall  
 31034 be checked in each set; that is, the descriptors from zero through *nfds*–1 in the descriptor sets  
 31035 shall be examined.

31036 If the *readfds* argument is not a null pointer, it points to an object of type **fd\_set** that on input  
 31037 specifies the file descriptors to be checked for being ready to read, and on output indicates  
 31038 which file descriptors are ready to read.

31039 If the *writefds* argument is not a null pointer, it points to an object of type **fd\_set** that on input  
 31040 specifies the file descriptors to be checked for being ready to write, and on output indicates  
 31041 which file descriptors are ready to write.

31042 If the *errorfds* argument is not a null pointer, it points to an object of type **fd\_set** that on input  
 31043 specifies the file descriptors to be checked for error conditions pending, and on output indicates  
 31044 which file descriptors have error conditions pending.

31045 Upon successful completion, the *pselect()* or *select()* function shall modify the objects pointed to  
 31046 by the *readfds*, *writefds*, and *errorfds* arguments to indicate which file descriptors are ready for  
 31047 reading, ready for writing, or have an error condition pending, respectively, and shall return the  
 31048 total number of ready descriptors in all the output sets. For each file descriptor less than *nfds*, the

31049 corresponding bit shall be set on successful completion if it was set on input and the associated  
31050 condition is true for that file descriptor.

31051 If none of the selected descriptors are ready for the requested operation, the *pselect()* or *select()*  
31052 function shall block until at least one of the requested operations becomes ready, until the  
31053 *timeout* occurs, or until interrupted by a signal. The *timeout* parameter controls how long the  
31054 *pselect()* or *select()* function shall take before timing out. If the *timeout* parameter is not a null  
31055 pointer, it specifies a maximum interval to wait for the selection to complete. If the specified  
31056 time interval expires without any requested operation becoming ready, the function shall return.  
31057 If the *timeout* parameter is a null pointer, then the call to *pselect()* or *select()* shall block  
31058 indefinitely until at least one descriptor meets the specified criteria. To effect a poll, the *timeout*  
31059 parameter should not be a null pointer, and should point to a zero-valued **timespec** structure.

31060 The use of a timeout does not affect any pending timers set up by *alarm()*, *ualarm()*, or  
31061 *setitimer()*.

31062 Implementations may place limitations on the maximum timeout interval supported. All  
31063 implementations shall support a maximum timeout interval of at least 31 days. If the *timeout*  
31064 argument specifies a timeout interval greater than the implementation-defined maximum value,  
31065 the maximum value shall be used as the actual timeout value. Implementations may also place  
31066 limitations on the granularity of timeout intervals. If the requested timeout interval requires a  
31067 finer granularity than the implementation supports, the actual timeout interval shall be rounded  
31068 up to the next supported value.

31069 If *sigmask* is not a null pointer, then the *pselect()* function shall replace the signal mask of the  
31070 process by the set of signals pointed to by *sigmask* before examining the descriptors, and shall  
31071 restore the signal mask of the process before returning.

31072 A descriptor shall be considered ready for reading when a call to an input function with  
31073 O\_NONBLOCK clear would not block, whether or not the function would transfer data  
31074 successfully. (The function might return data, an end-of-file indication, or an error other than  
31075 one indicating that it is blocked, and in each of these cases the descriptor shall be considered  
31076 ready for reading.)

31077 A descriptor shall be considered ready for writing when a call to an output function with  
31078 O\_NONBLOCK clear would not block, whether or not the function would transfer data  
31079 successfully.

31080 If a socket has a pending error, it shall be considered to have an exceptional condition pending.  
31081 Otherwise, what constitutes an exceptional condition is file type-specific. For a file descriptor for  
31082 use with a socket, it is protocol-specific except as noted below. For other file types it is  
31083 implementation-defined. If the operation is meaningless for a particular file type, *pselect()* or  
31084 *select()* shall indicate that the descriptor is ready for read or write operations, and shall indicate  
31085 that the descriptor has no exceptional condition pending.

31086 If a descriptor refers to a socket, the implied input function is the *recvmsg()* function with  
31087 parameters requesting normal and ancillary data, such that the presence of either type shall  
31088 cause the socket to be marked as readable. The presence of out-of-band data shall be checked if  
31089 the socket option SO\_OOBINLINE has been enabled, as out-of-band data is enqueued with  
31090 normal data. If the socket is currently listening, then it shall be marked as readable if an  
31091 incoming connection request has been received, and a call to the *accept()* function shall complete  
31092 without blocking.

31093 If a descriptor refers to a socket, the implied output function is the *sendmsg()* function supplying  
31094 an amount of normal data equal to the current value of the SO\_SNDLOWAT option for the  
31095 socket. If a non-blocking call to the *connect()* function has been made for a socket, and the  
31096 connection attempt has either succeeded or failed leaving a pending error, the socket shall be

- 31097 marked as writable.
- 31098 A socket shall be considered to have an exceptional condition pending if a receive operation  
31099 with `O_NONBLOCK` clear for the open file description and with the `MSG_OOB` flag set would  
31100 return out-of-band data without blocking. (It is protocol-specific whether the `MSG_OOB` flag  
31101 would be used to read out-of-band data.) A socket shall also be considered to have an  
31102 exceptional condition pending if an out-of-band data mark is present in the receive queue. Other  
31103 circumstances under which a socket may be considered to have an exceptional condition  
31104 pending are protocol-specific and implementation-defined.
- 31105 If the `readfds`, `writefds`, and `errorfds` arguments are all null pointers and the `timeout` argument is not  
31106 a null pointer, the `pselect()` or `select()` function shall block for the time specified, or until  
31107 interrupted by a signal. If the `readfds`, `writefds`, and `errorfds` arguments are all null pointers and the  
31108 `timeout` argument is a null pointer, the `pselect()` or `select()` function shall block until interrupted  
31109 by a signal.
- 31110 File descriptors associated with regular files shall always select true for ready to read, ready to  
31111 write, and error conditions.
- 31112 On failure, the objects pointed to by the `readfds`, `writefds`, and `errorfds` arguments shall not be  
31113 modified. If the timeout interval expires without the specified condition being true for any of the  
31114 specified file descriptors, the objects pointed to by the `readfds`, `writefds`, and `errorfds` arguments  
31115 shall have all bits set to 0.
- 31116 File descriptor masks of type `fd_set` can be initialized and tested with `FD_CLR()`, `FD_ISSET()`,  
31117 `FD_SET()`, and `FD_ZERO()`. It is unspecified whether each of these is a macro or a function. If a  
31118 macro definition is suppressed in order to access an actual function, or a program defines an  
31119 external identifier with any of these names, the behavior is undefined.
- 31120 `FD_CLR(fd, fdsetp)` shall remove the file descriptor `fd` from the set pointed to by `fdsetp`. If `fd` is not  
31121 a member of this set, there shall be no effect on the set, nor will an error be returned.
- 31122 `FD_ISSET(fd, fdsetp)` shall evaluate to non-zero if the file descriptor `fd` is a member of the set  
31123 pointed to by `fdsetp`, and shall evaluate to zero otherwise.
- 31124 `FD_SET(fd, fdsetp)` shall add the file descriptor `fd` to the set pointed to by `fdsetp`. If the file  
31125 descriptor `fd` is already in this set, there shall be no effect on the set, nor will an error be returned.
- 31126 `FD_ZERO(fdsetp)` shall initialize the descriptor set pointed to by `fdsetp` to the null set. No error is  
31127 returned if the set is not empty at the time `FD_ZERO()` is invoked.
- 31128 The behavior of these macros is undefined if the `fd` argument is less than 0 or greater than or  
31129 equal to `FD_SETSIZE`, or if `fd` is not a valid file descriptor, or if any of the arguments are  
31130 expressions with side effects.
- 31131 **RETURN VALUE**
- 31132 Upon successful completion, the `pselect()` and `select()` functions shall return the total number of  
31133 bits set in the bit masks. Otherwise, `-1` shall be returned, and `errno` shall be set to indicate the  
31134 error.
- 31135 `FD_CLR()`, `FD_SET()`, and `FD_ZERO()` do not return a value. `FD_ISSET()` shall return a non-  
31136 zero value if the bit for the file descriptor `fd` is set in the file descriptor set pointed to by `fdset`, and  
31137 0 otherwise.
- 31138 **ERRORS**
- 31139 Under the following conditions, `pselect()` and `select()` shall fail and set `errno` to:
- 31140 [EBADF] One or more of the file descriptor sets specified a file descriptor that is not a  
31141 valid open file descriptor.

- 31142 [EINTR] The function was interrupted before any of the selected events occurred and  
31143 before the timeout interval expired.
- 31144 XSI If SA\_RESTART has been set for the interrupting signal, it is implementation-  
31145 defined whether the function restarts or returns with [EINTR].
- 31146 [EINVAL] An invalid timeout interval was specified.
- 31147 [EINVAL] The *nfds* argument is less than 0 or greater than FD\_SETSIZE.
- 31148 XSR [EINVAL] One of the specified file descriptors refers to a STREAM or multiplexer that is  
31149 linked (directly or indirectly) downstream from a multiplexer.

**31150 EXAMPLES**

31151 None.

**31152 APPLICATION USAGE**

31153 None.

**31154 RATIONALE**

31155 In previous versions of the Single UNIX Specification, the *select()* function was defined in the  
31156 `<sys/time.h>` header. This is now changed to `<sys/select.h>`. The rationale for this change was  
31157 as follows: the introduction of the *pselect()* function included the `<sys/select.h>` header and the  
31158 `<sys/select.h>` header defines all the related definitions for the *pselect()* and *select()* functions.  
31159 Backwards-compatibility to existing XSI implementations is handled by allowing `<sys/time.h>`  
31160 to include `<sys/select.h>`.

**31161 FUTURE DIRECTIONS**

31162 None.

**31163 SEE ALSO**

31164 *accept()*, *alarm()*, *connect()*, *fcntl()*, *poll()*, *read()*, *recvmsg()*, *sendmsg()*, *setitimer()*, *ualarm()*,  
31165 *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<sys/select.h>`, `<sys/time.h>`

**31166 CHANGE HISTORY**

31167 First released in Issue 4, Version 2.

**31168 Issue 5**

31169 Moved from X/OPEN UNIX extension to BASE.

31170 In the ERRORS section, the text has been changed to indicate that [EINVAL] is returned when  
31171 *nfds* is less than 0 or greater than FD\_SETSIZE. It previously stated less than 0, or greater than or  
31172 equal to FD\_SETSIZE.

31173 Text about *timeout* is moved from the APPLICATION USAGE section to the DESCRIPTION.

**31174 Issue 6**

31175 The Open Group Corrigendum U026/6 is applied, changing the occurrences of *readfds* and *writfds*  
31176 in the *select()* DESCRIPTION to be *readfds* and *writfds*.

31177 Text referring to sockets is added to the DESCRIPTION.

31178 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are  
31179 marked as part of the XSI STREAMS Option Group.

31180 The following new requirements on POSIX implementations derive from alignment with the  
31181 Single UNIX Specification:

- 31182 • These functions are now mandatory.

31183 The *pselect()* function is added for alignment with IEEE Std 1003.1g-2000 and additional detail  
31184 related to sockets semantics is added to the DESCRIPTION.

31185 The *select()* function now requires inclusion of `<sys/select.h>`.  
31186 The **restrict** keyword is added to the *select()* prototype for alignment with the  
31187 ISO/IEC 9899:1999 standard.

31188 **NAME**

31189 pthread\_atfork — register fork handlers

31190 **SYNOPSIS**

31191 THR #include &lt;pthread.h&gt;

```
31192 int pthread_atfork(void (*prepare)(void), void (*parent)(void),
31193 void (*child)(void));
```

31194

31195 **DESCRIPTION**

31196 The *pthread\_atfork()* function shall declare fork handlers to be called before and after *fork()*, in  
 31197 the context of the thread that called *fork()*. The *prepare* fork handler shall be called before *fork()*  
 31198 processing commences. The *parent* fork handle shall be called after *fork()* processing completes  
 31199 in the parent process. The *child* fork handler shall be called after *fork()* processing completes in  
 31200 the child process. If no handling is desired at one or more of these three points, the  
 31201 corresponding fork handler address(es) may be set to NULL.

31202 The order of calls to *pthread\_atfork()* is significant. The *parent* and *child* fork handlers shall be  
 31203 called in the order in which they were established by calls to *pthread\_atfork()*. The *prepare* fork  
 31204 handlers shall be called in the opposite order.

31205 **RETURN VALUE**

31206 Upon successful completion, *pthread\_atfork()* shall return a value of zero; otherwise, an error  
 31207 number shall be returned to indicate the error.

31208 **ERRORS**31209 The *pthread\_atfork()* function shall fail if:

31210 [ENOMEM] Insufficient table space exists to record the fork handler addresses.

31211 The *pthread\_atfork()* function shall not return an error code of [EINTR].31212 **EXAMPLES**

31213 None.

31214 **APPLICATION USAGE**

31215 None.

31216 **RATIONALE**

31217 There are at least two serious problems with the semantics of *fork()* in a multi-threaded  
 31218 program. One problem has to do with state (for example, memory) covered by mutexes.  
 31219 Consider the case where one thread has a mutex locked and the state covered by that mutex is  
 31220 inconsistent while another thread calls *fork()*. In the child, the mutex is in the locked state  
 31221 (locked by a nonexistent thread and thus can never be unlocked). Having the child simply  
 31222 reinitialize the mutex is unsatisfactory since this approach does not resolve the question about  
 31223 how to correct or otherwise deal with the inconsistent state in the child.

31224 It is suggested that programs that use *fork()* call an *exec* function very soon afterwards in the  
 31225 child process, thus resetting all states. In the meantime, only a short list of async-signal-safe  
 31226 library routines are promised to be available.

31227 Unfortunately, this solution does not address the needs of multi-threaded libraries. Application  
 31228 programs may not be aware that a multi-threaded library is in use, and they feel free to call any  
 31229 number of library routines between the *fork()* and *exec* calls, just as they always have. Indeed,  
 31230 they may be extant single-threaded programs and cannot, therefore, be expected to obey new  
 31231 restrictions imposed by the threads library.

31232 On the other hand, the multi-threaded library needs a way to protect its internal state during  
31233 *fork()* in case it is re-entered later in the child process. The problem arises especially in multi-  
31234 threaded I/O libraries, which are almost sure to be invoked between the *fork()* and *exec* calls to  
31235 effect I/O redirection. The solution may require locking mutex variables during *fork()*, or it may  
31236 entail simply resetting the state in the child after the *fork()* processing completes.

31237 The *pthread\_atfork()* function provides multi-threaded libraries with a means to protect  
31238 themselves from innocent application programs that call *fork()*, and it provides multi-threaded  
31239 application programs with a standard mechanism for protecting themselves from *fork()* calls in  
31240 a library routine or the application itself.

31241 The expected usage is that the *prepare* handler acquires all mutex locks and the other two fork  
31242 handlers release them.

31243 For example, an application can supply a *prepare* routine that acquires the necessary mutexes the  
31244 library maintains and supply *child* and *parent* routines that release those mutexes, thus ensuring  
31245 that the child gets a consistent snapshot of the state of the library (and that no mutexes are left  
31246 stranded). Alternatively, some libraries might be able to supply just a *child* routine that  
31247 reinitializes the mutexes in the library and all associated states to some known value (for  
31248 example, what it was when the image was originally executed).

31249 When *fork()* is called, only the calling thread is duplicated in the child process. Synchronization  
31250 variables remain in the same state in the child as they were in the parent at the time *fork()* was  
31251 called. Thus, for example, mutex locks may be held by threads that no longer exist in the child  
31252 process, and any associated states may be inconsistent. The parent process may avoid this by  
31253 explicit code that acquires and releases locks critical to the child via *pthread\_atfork()*. In addition,  
31254 any critical threads need to be recreated and reinitialized to the proper state in the child (also via  
31255 *pthread\_atfork()*).

31256 A higher-level package may acquire locks on its own data structures before invoking lower-level  
31257 packages. Under this scenario, the order specified for fork handler calls allows a simple rule of  
31258 initialization for avoiding package deadlock: a package initializes all packages on which it  
31259 depends before it calls the *pthread\_atfork()* function for itself.

#### 31260 **FUTURE DIRECTIONS**

31261 None.

#### 31262 **SEE ALSO**

31263 *atexit()*, *fork()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>

#### 31264 **CHANGE HISTORY**

31265 First released in Issue 5. Derived from the POSIX Threads Extension.

31266 IEEE PASC Interpretation 1003.1c #4 is applied.

#### 31267 **Issue 6**

31268 The *pthread\_atfork()* function is marked as part of the Threads option.

31269 The <pthread.h> header is added to the SYNOPSIS.

31270 **NAME**

31271 pthread\_attr\_destroy, pthread\_attr\_init — destroy and initialize the thread attributes object

31272 **SYNOPSIS**

31273 THR #include <pthread.h>

31274 int pthread\_attr\_destroy(pthread\_attr\_t \*attr);

31275 int pthread\_attr\_init(pthread\_attr\_t \*attr);

31276

31277 **DESCRIPTION**

31278 The *pthread\_attr\_destroy()* function shall destroy a thread attributes object. An implementation  
 31279 may cause *pthread\_attr\_destroy()* to set *attr* to an implementation-defined invalid value. A  
 31280 destroyed *attr* attributes object can be reinitialized using *pthread\_attr\_init()*; the results of  
 31281 otherwise referencing the object after it has been destroyed are undefined.

31282 The *pthread\_attr\_init()* function shall initialize a thread attributes object *attr* with the default  
 31283 value for all of the individual attributes used by a given implementation.

31284 The resulting attributes object (possibly modified by setting individual attribute values) when  
 31285 used by *pthread\_create()* defines the attributes of the thread created. A single attributes object can  
 31286 be used in multiple simultaneous calls to *pthread\_create()*. Results are undefined if  
 31287 *pthread\_attr\_init()* is called specifying an already initialized *attr* attributes object.

31288 **RETURN VALUE**

31289 Upon successful completion, *pthread\_attr\_destroy()* and *pthread\_attr\_init()* shall return a value of  
 31290 0; otherwise, an error number shall be returned to indicate the error.

31291 **ERRORS**

31292 The *pthread\_attr\_init()* function shall fail if:

31293 [ENOMEM] Insufficient memory exists to initialize the thread attributes object.

31294 These functions shall not return an error code of [EINTR].

31295 **EXAMPLES**

31296 None.

31297 **APPLICATION USAGE**

31298 None.

31299 **RATIONALE**

31300 Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to  
 31301 support probable future standardization in these areas without requiring that the function itself  
 31302 be changed.

31303 Attributes objects provide clean isolation of the configurable aspects of threads. For example,  
 31304 “stack size” is an important attribute of a thread, but it cannot be expressed portably. When  
 31305 porting a threaded program, stack sizes often need to be adjusted. The use of attributes objects  
 31306 can help by allowing the changes to be isolated in a single place, rather than being spread across  
 31307 every instance of thread creation.

31308 Attributes objects can be used to set up “classes” of threads with similar attributes; for example,  
 31309 “threads with large stacks and high priority” or “threads with minimal stacks”. These classes  
 31310 can be defined in a single place and then referenced wherever threads need to be created.  
 31311 Changes to “class” decisions become straightforward, and detailed analysis of each  
 31312 *pthread\_create()* call is not required.

31313 The attributes objects are defined as opaque types as an aid to extensibility. If these objects had  
 31314 been specified as structures, adding new attributes would force recompilation of all multi-

31315 threaded programs when the attributes objects are extended; this might not be possible if  
31316 different program components were supplied by different vendors.

31317 Additionally, opaque attributes objects present opportunities for improving performance.  
31318 Argument validity can be checked once when attributes are set, rather than each time a thread is  
31319 created. Implementations often need to cache kernel objects that are expensive to create.  
31320 Opaque attributes objects provide an efficient mechanism to detect when cached objects become  
31321 invalid due to attribute changes.

31322 Since assignment is not necessarily defined on a given opaque type, implementation-defined  
31323 default values cannot be defined in a portable way. The solution to this problem is to allow  
31324 attributes objects to be initialized dynamically by attributes object initialization functions, so  
31325 that default values can be supplied automatically by the implementation.

31326 The following proposal was provided as a suggested alternative to the supplied attributes:

- 31327 1. Maintain the style of passing a parameter formed by the bitwise-inclusive OR of flags to  
31328 the initialization routines (*pthread\_create()*, *pthread\_mutex\_init()*, *pthread\_cond\_init()*). The  
31329 parameter containing the flags should be an opaque type for extensibility. If no flags are  
31330 set in the parameter, then the objects are created with default characteristics. An  
31331 implementation may specify implementation-defined flag values and associated behavior.
- 31332 2. If further specialization of mutexes and condition variables is necessary, implementations  
31333 may specify additional procedures that operate on the **pthread\_mutex\_t** and  
31334 **pthread\_cond\_t** objects (instead of on attributes objects).

31335 The difficulties with this solution are:

- 31336 1. A bitmask is not opaque if bits have to be set into bitvector attributes objects using  
31337 explicitly-coded bitwise-inclusive OR operations. If the set of options exceeds an **int**,  
31338 application programmers need to know the location of each bit. If bits are set or read by  
31339 encapsulation (that is, *get* and *set* functions), then the bitmask is merely an  
31340 implementation of attributes objects as currently defined and should not be exposed to the  
31341 programmer.
- 31342 2. Many attributes are not Boolean or very small integral values. For example, scheduling  
31343 policy may be placed in 3-bit or 4-bit, but priority requires 5-bit or more, thereby taking up  
31344 at least 8 bits out of a possible 16 bits on machines with 16-bit integers. Because of this, the  
31345 bitmask can only reasonably control whether particular attributes are set or not, and it  
31346 cannot serve as the repository of the value itself. The value needs to be specified as a  
31347 function parameter (which is non-extensible), or by setting a structure field (which is non-  
31348 opaque), or by *get* and *set* functions (making the bitmask a redundant addition to the  
31349 attributes objects).

31350 Stack size is defined as an optional attribute because the very notion of a stack is inherently  
31351 machine-dependent. Some implementations may not be able to change the size of the stack, for  
31352 example, and others may not need to because stack pages may be discontinuous and can be  
31353 allocated and released on demand.

31354 The attribute mechanism has been designed in large measure for extensibility. Future extensions  
31355 to the attribute mechanism or to any attributes object defined in this volume of  
31356 IEEE Std 1003.1-2001 has to be done with care so as not to affect binary-compatibility.

31357 Attributes objects, even if allocated by means of dynamic allocation functions such as *malloc()*,  
31358 may have their size fixed at compile time. This means, for example, a *pthread\_create()* in an  
31359 implementation with extensions to **pthread\_attr\_t** cannot look beyond the area that the binary  
31360 application assumes is valid. This suggests that implementations should maintain a size field in  
31361 the attributes object, as well as possibly version information, if extensions in different directions

31362 (possibly by different vendors) are to be accommodated.

31363 **FUTURE DIRECTIONS**

31364 None.

31365 **SEE ALSO**

31366 *pthread\_attr\_getstackaddr()*, *pthread\_attr\_getstacksize()*, *pthread\_attr\_getdetachstate()*,  
31367 *pthread\_create()*, the Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

31368 **CHANGE HISTORY**

31369 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31370 **Issue 6**

31371 The *pthread\_attr\_destroy()* and *pthread\_attr\_init()* functions are marked as part of the Threads  
31372 option.

31373 IEEE PASC Interpretation 1003.1 #107 is applied, noting that the effect of initializing an already  
31374 initialized thread attributes object is undefined.

31375 **NAME**

31376 pthread\_attr\_getdetachstate, pthread\_attr\_setdetachstate — get and set the detachstate attribute

31377 **SYNOPSIS**

31378 THR #include <pthread.h>

```
31379 int pthread_attr_getdetachstate(const pthread_attr_t *attr,
31380 int *detachstate);
```

```
31381 int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
31382
```

31383 **DESCRIPTION**

31384 The *detachstate* attribute controls whether the thread is created in a detached state. If the thread  
31385 is created detached, then use of the ID of the newly created thread by the *pthread\_detach()* or  
31386 *pthread\_join()* function is an error.

31387 The *pthread\_attr\_getdetachstate()* and *pthread\_attr\_setdetachstate()* functions, respectively, shall  
31388 get and set the *detachstate* attribute in the *attr* object.

31389 For *pthread\_attr\_getdetachstate()*, *detachstate* shall be set to either  
31390 PTHREAD\_CREATE\_DETACHED or PTHREAD\_CREATE\_JOINABLE.

31391 For *pthread\_attr\_setdetachstate()*, the application shall set *detachstate* to either  
31392 PTHREAD\_CREATE\_DETACHED or PTHREAD\_CREATE\_JOINABLE.

31393 A value of PTHREAD\_CREATE\_DETACHED shall cause all threads created with *attr* to be in  
31394 the detached state, whereas using a value of PTHREAD\_CREATE\_JOINABLE shall cause all  
31395 threads created with *attr* to be in the joinable state. The default value of the *detachstate* attribute  
31396 shall be PTHREAD\_CREATE\_JOINABLE.

31397 **RETURN VALUE**

31398 Upon successful completion, *pthread\_attr\_getdetachstate()* and *pthread\_attr\_setdetachstate()* shall  
31399 return a value of 0; otherwise, an error number shall be returned to indicate the error.

31400 The *pthread\_attr\_getdetachstate()* function stores the value of the *detachstate* attribute in *detachstate*  
31401 if successful.

31402 **ERRORS**

31403 The *pthread\_attr\_setdetachstate()* function shall fail if:

31404 [EINVAL] The value of *detachstate* was not valid

31405 These functions shall not return an error code of [EINTR].

31406 **EXAMPLES**

31407 None.

31408 **APPLICATION USAGE**

31409 None.

31410 **RATIONALE**

31411 None.

31412 **FUTURE DIRECTIONS**

31413 None.

31414 **SEE ALSO**

31415 *pthread\_attr\_destroy()*, *pthread\_attr\_getstackaddr()*, *pthread\_attr\_getstacksize()*, *pthread\_create()*, the  
31416 Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

31417 **CHANGE HISTORY**

31418 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31419 **Issue 6**

31420 The *pthread\_attr\_setdetachstate()* and *pthread\_attr\_getdetachstate()* functions are marked as part of the Threads option.

31422 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

## 31423 NAME

31424 pthread\_attr\_getguardsize, pthread\_attr\_setguardsize — get and set the thread guardsize  
31425 attribute

## 31426 SYNOPSIS

```
31427 xsi #include <pthread.h>
31428 int pthread_attr_getguardsize(const pthread_attr_t *restrict attr,
31429 size_t *restrict guardsize);
31430 int pthread_attr_setguardsize(pthread_attr_t *attr,
31431 size_t guardsize);
31432
```

## 31433 DESCRIPTION

31434 The *pthread\_attr\_getguardsize()* function shall get the *guardsize* attribute in the *attr* object. This  
31435 attribute shall be returned in the *guardsize* parameter.

31436 The *pthread\_attr\_setguardsize()* function shall set the *guardsize* attribute in the *attr* object. The new  
31437 value of this attribute shall be obtained from the *guardsize* parameter. If *guardsize* is zero, a guard  
31438 area shall not be provided for threads created with *attr*. If *guardsize* is greater than zero, a guard  
31439 area of at least size *guardsize* bytes shall be provided for each thread created with *attr*.

31440 The *guardsize* attribute controls the size of the guard area for the created thread's stack. The  
31441 *guardsize* attribute provides protection against overflow of the stack pointer. If a thread's stack is  
31442 created with guard protection, the implementation allocates extra memory at the overflow end  
31443 of the stack as a buffer against stack overflow of the stack pointer. If an application overflows  
31444 into this buffer an error shall result (possibly in a SIGSEGV signal being delivered to the thread).

31445 A conforming implementation may round up the value contained in *guardsize* to a multiple of  
31446 the configurable system variable {PAGESIZE} (see <sys/mman.h>). If an implementation  
31447 rounds up the value of *guardsize* to a multiple of {PAGESIZE}, a call to *pthread\_attr\_getguardsize()*  
31448 specifying *attr* shall store in the *guardsize* parameter the guard size specified by the previous  
31449 *pthread\_attr\_setguardsize()* function call.

31450 The default value of the *guardsize* attribute is {PAGESIZE} bytes. The actual value of {PAGESIZE}  
31451 is implementation-defined.

31452 If the *stackaddr* or *stack* attribute has been set (that is, the caller is allocating and managing its  
31453 own thread stacks), the *guardsize* attribute shall be ignored and no protection shall be provided  
31454 by the implementation. It is the responsibility of the application to manage stack overflow along  
31455 with stack allocation and management in this case.

## 31456 RETURN VALUE

31457 If successful, the *pthread\_attr\_getguardsize()* and *pthread\_attr\_setguardsize()* functions shall return  
31458 zero; otherwise, an error number shall be returned to indicate the error.

## 31459 ERRORS

31460 The *pthread\_attr\_getguardsize()* and *pthread\_attr\_setguardsize()* functions shall fail if:

31461 [EINVAL] The attribute *attr* is invalid.

31462 [EINVAL] The parameter *guardsize* is invalid.

31463 These functions shall not return an error code of [EINTR].

31464 **EXAMPLES**

31465 None.

31466 **APPLICATION USAGE**

31467 None.

31468 **RATIONALE**31469 The *guardsize* attribute is provided to the application for two reasons:

- 31470 1. Overflow protection can potentially result in wasted system resources. An application  
31471 that creates a large number of threads, and which knows its threads never overflow their  
31472 stack, can save system resources by turning off guard areas.
- 31473 2. When threads allocate large data structures on the stack, large guard areas may be needed  
31474 to detect stack overflow.

31475 **FUTURE DIRECTIONS**

31476 None.

31477 **SEE ALSO**31478 The Base Definitions volume of IEEE Std 1003.1-2001, `<pthread.h>`, `<sys/mman.h>`31479 **CHANGE HISTORY**

31480 First released in Issue 5.

31481 **Issue 6**31482 In the **ERRORS** section, a third [EINVAL] error condition is removed as it is covered by the  
31483 second error condition.31484 The **restrict** keyword is added to the *pthread\_attr\_getguardsize()* prototype for alignment with the  
31485 ISO/IEC 9899:1999 standard.

## 31486 NAME

31487 pthread\_attr\_getinheritsched, pthread\_attr\_setinheritsched — get and set the inheritsched  
 31488 attribute (**REALTIME THREADS**)

## 31489 SYNOPSIS

31490 THR TPS #include <pthread.h>

```
31491 int pthread_attr_getinheritsched(const pthread_attr_t *restrict attr,
31492 int *restrict inheritsched);
31493 int pthread_attr_setinheritsched(pthread_attr_t *attr,
31494 int inheritsched);
31495
```

## 31496 DESCRIPTION

31497 The *pthread\_attr\_getinheritsched()*, and *pthread\_attr\_setinheritsched()* functions, respectively, shall  
 31498 get and set the *inheritsched* attribute in the *attr* argument.

31499 When the attributes objects are used by *pthread\_create()*, the *inheritsched* attribute determines  
 31500 how the other scheduling attributes of the created thread shall be set.

## 31501 PTHREAD\_INHERIT\_SCHED

31502 Specifies that the thread scheduling attributes shall be inherited from the creating thread,  
 31503 and the scheduling attributes in this *attr* argument shall be ignored.

## 31504 PTHREAD\_EXPLICIT\_SCHED

31505 Specifies that the thread scheduling attributes shall be set to the corresponding values from  
 31506 this attributes object.

31507 The symbols PTHREAD\_INHERIT\_SCHED and PTHREAD\_EXPLICIT\_SCHED are defined in  
 31508 the <pthread.h> header.

31509 The following thread scheduling attributes defined by IEEE Std 1003.1-2001 are affected by the  
 31510 *inheritsched* attribute: scheduling policy (*schedpolicy*), scheduling parameters (*schedparam*), and  
 31511 scheduling contention scope (*contentionscope*).

## 31512 RETURN VALUE

31513 If successful, the *pthread\_attr\_getinheritsched()* and *pthread\_attr\_setinheritsched()* functions shall  
 31514 return zero; otherwise, an error number shall be returned to indicate the error.

## 31515 ERRORS

31516 The *pthread\_attr\_setinheritsched()* function may fail if:

31517 [EINVAL] The value of *inheritsched* is not valid.

31518 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

31519 These functions shall not return an error code of [EINTR].

## 31520 EXAMPLES

31521 None.

## 31522 APPLICATION USAGE

31523 After these attributes have been set, a thread can be created with the specified attributes using  
 31524 *pthread\_create()*. Using these routines does not affect the current running thread.

## 31525 RATIONALE

31526 None.

31527 **FUTURE DIRECTIONS**

31528 None.

31529 **SEE ALSO**

31530 *pthread\_attr\_destroy()*, *pthread\_attr\_getscope()*, *pthread\_attr\_getschedpolicy()*,  
31531 *pthread\_attr\_getschedparam()*, *pthread\_create()*, the Base Definitions volume of  
31532 IEEE Std 1003.1-2001, <pthread.h>, <sched.h>

31533 **CHANGE HISTORY**

31534 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31535 Marked as part of the Realtime Threads Feature Group.

31536 **Issue 6**31537 The *pthread\_attr\_getinheritsched()* and *pthread\_attr\_setinheritsched()* functions are marked as part  
31538 of the Threads and Thread Execution Scheduling options.31539 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
31540 implementation does not support the Thread Execution Scheduling option.31541 The **restrict** keyword is added to the *pthread\_attr\_getinheritsched()* prototype for alignment with  
31542 the ISO/IEC 9899:1999 standard.

## 31543 NAME

31544 pthread\_attr\_getschedparam, pthread\_attr\_setschedparam — get and set the schedparam  
31545 attribute

## 31546 SYNOPSIS

31547 THR #include <pthread.h>

```
31548 int pthread_attr_getschedparam(const pthread_attr_t *restrict attr,
31549 struct sched_param *restrict param);
```

```
31550 int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
31551 const struct sched_param *restrict param);
```

31552

## 31553 DESCRIPTION

31554 The *pthread\_attr\_getschedparam()*, and *pthread\_attr\_setschedparam()* functions, respectively, shall  
31555 get and set the scheduling parameter attributes in the *attr* argument. The contents of the *param*  
31556 structure are defined in the <sched.h> header. For the SCHED\_FIFO and SCHED\_RR policies,  
31557 the only required member of *param* is *sched\_priority*.

31558 TSP For the SCHED\_SPORADIC policy, the required members of the *param* structure are  
31559 *sched\_priority*, *sched\_ss\_low\_priority*, *sched\_ss\_repl\_period*, *sched\_ss\_init\_budget*, and  
31560 *sched\_ss\_max\_repl*. The specified *sched\_ss\_repl\_period* must be greater than or equal to the  
31561 specified *sched\_ss\_init\_budget* for the function to succeed; if it is not, then the function shall fail.  
31562 The value of *sched\_ss\_max\_repl* shall be within the inclusive range [1,{SS\_REPL\_MAX}] for the  
31563 function to succeed; if not, the function shall fail.

## 31564 RETURN VALUE

31565 If successful, the *pthread\_attr\_getschedparam()* and *pthread\_attr\_setschedparam()* functions shall  
31566 return zero; otherwise, an error number shall be returned to indicate the error.

## 31567 ERRORS

31568 The *pthread\_attr\_setschedparam()* function may fail if:

31569 [EINVAL] The value of *param* is not valid.

31570 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

31571 These functions shall not return an error code of [EINTR].

## 31572 EXAMPLES

31573 None.

## 31574 APPLICATION USAGE

31575 After these attributes have been set, a thread can be created with the specified attributes using  
31576 *pthread\_create()*. Using these routines does not affect the current running thread.

## 31577 RATIONALE

31578 None.

## 31579 FUTURE DIRECTIONS

31580 None.

## 31581 SEE ALSO

31582 *pthread\_attr\_destroy()*, *pthread\_attr\_getscope()*, *pthread\_attr\_getinheritsched()*,  
31583 *pthread\_attr\_getschedpolicy()*, *pthread\_create()*, the Base Definitions volume of  
31584 IEEE Std 1003.1-2001, <pthread.h>, <sched.h>

31585 **CHANGE HISTORY**

31586 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31587 **Issue 6**

31588 The *pthread\_attr\_getschedparam()* and *pthread\_attr\_setschedparam()* functions are marked as part  
31589 of the Threads option.

31590 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

31591 The **restrict** keyword is added to the *pthread\_attr\_getschedparam()* and  
31592 *pthread\_attr\_setschedparam()* prototypes for alignment with the ISO/IEC 9899:1999 standard.

## 31593 NAME

31594 pthread\_attr\_getschedpolicy, pthread\_attr\_setschedpolicy — get and set the schedpolicy  
31595 attribute (**REALTIME THREADS**)

## 31596 SYNOPSIS

```
31597 THR TPS #include <pthread.h>
31598
31598 int pthread_attr_getschedpolicy(const pthread_attr_t *restrict attr,
31599 int *restrict policy);
31600 int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
31601
```

## 31602 DESCRIPTION

31603 The *pthread\_attr\_getschedpolicy()* and *pthread\_attr\_setschedpolicy()* functions, respectively, shall  
31604 get and set the *schedpolicy* attribute in the *attr* argument.

31605 The supported values of *policy* shall include SCHED\_FIFO, SCHED\_RR, and SCHED\_OTHER,  
31606 which are defined in the <**sched.h**> header. When threads executing with the scheduling policy  
31607 TSP SCHED\_FIFO, SCHED\_RR, or SCHED\_SPORADIC are waiting on a mutex, they shall acquire  
31608 the mutex in priority order when the mutex is unlocked.

## 31609 RETURN VALUE

31610 If successful, the *pthread\_attr\_getschedpolicy()* and *pthread\_attr\_setschedpolicy()* functions shall  
31611 return zero; otherwise, an error number shall be returned to indicate the error.

## 31612 ERRORS

31613 The *pthread\_attr\_setschedpolicy()* function may fail if:

31614 [EINVAL] The value of *policy* is not valid.

31615 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

31616 These functions shall not return an error code of [EINTR].

## 31617 EXAMPLES

31618 None.

## 31619 APPLICATION USAGE

31620 After these attributes have been set, a thread can be created with the specified attributes using  
31621 *pthread\_create()*. Using these routines does not affect the current running thread.

## 31622 RATIONALE

31623 None.

## 31624 FUTURE DIRECTIONS

31625 None.

## 31626 SEE ALSO

31627 *pthread\_attr\_destroy()*, *pthread\_attr\_getscope()*, *pthread\_attr\_getinheritsched()*,  
31628 *pthread\_attr\_getschedparam()*, *pthread\_create()*, the Base Definitions volume of  
31629 IEEE Std 1003.1-2001, <**pthread.h**>, <**sched.h**>

## 31630 CHANGE HISTORY

31631 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31632 Marked as part of the Realtime Threads Feature Group.

31633 **Issue 6**

31634 The *pthread\_attr\_getschedpolicy()* and *pthread\_attr\_setschedpolicy()* functions are marked as part of  
31635 the Threads and Thread Execution Scheduling options.

31636 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
31637 implementation does not support the Thread Execution Scheduling option.

31638 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

31639 The **restrict** keyword is added to the *pthread\_attr\_getschedpolicy()* prototype for alignment with  
31640 the ISO/IEC 9899:1999 standard.

31641 **NAME**

31642 pthread\_attr\_getscope, pthread\_attr\_setscope — get and set the contentionscope attribute  
 31643 (**REALTIME THREADS**)

31644 **SYNOPSIS**

```
31645 THR TPS #include <pthread.h>
```

```
31646 int pthread_attr_getscope(const pthread_attr_t *restrict attr,
31647 int *restrict contentionscope);
31648 int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);
31649
```

31650 **DESCRIPTION**

31651 The *pthread\_attr\_getscope()* and *pthread\_attr\_setscope()* functions, respectively, shall get and set  
 31652 the *contentionscope* attribute in the *attr* object.

31653 The *contentionscope* attribute may have the values PTHREAD\_SCOPE\_SYSTEM, signifying  
 31654 system scheduling contention scope, or PTHREAD\_SCOPE\_PROCESS, signifying process  
 31655 scheduling contention scope. The symbols PTHREAD\_SCOPE\_SYSTEM and  
 31656 PTHREAD\_SCOPE\_PROCESS are defined in the **<pthread.h>** header.

31657 **RETURN VALUE**

31658 If successful, the *pthread\_attr\_getscope()* and *pthread\_attr\_setscope()* functions shall return zero;  
 31659 otherwise, an error number shall be returned to indicate the error.

31660 **ERRORS**

31661 The *pthread\_attr\_setscope()* function may fail if:

31662 [EINVAL] The value of *contentionscope* is not valid.

31663 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

31664 These functions shall not return an error code of [EINTR].

31665 **EXAMPLES**

31666 None.

31667 **APPLICATION USAGE**

31668 After these attributes have been set, a thread can be created with the specified attributes using  
 31669 *pthread\_create()*. Using these routines does not affect the current running thread.

31670 **RATIONALE**

31671 None.

31672 **FUTURE DIRECTIONS**

31673 None.

31674 **SEE ALSO**

31675 *pthread\_attr\_destroy()*, *pthread\_attr\_getinheritsched()*, *pthread\_attr\_getschedpolicy()*,  
 31676 *pthread\_attr\_getschedparam()*, *pthread\_create()*, the Base Definitions volume of  
 31677 IEEE Std 1003.1-2001, **<pthread.h>**, **<sched.h>**

31678 **CHANGE HISTORY**

31679 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31680 Marked as part of the Realtime Threads Feature Group.

31681 **Issue 6**

31682 The *pthread\_attr\_getscope()* and *pthread\_attr\_setscope()* functions are marked as part of the  
31683 Threads and Thread Execution Scheduling options.

31684 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
31685 implementation does not support the Thread Execution Scheduling option.

31686 The **restrict** keyword is added to the *pthread\_attr\_getscope()* prototype for alignment with the  
31687 ISO/IEC 9899:1999 standard.

## 31688 NAME

31689 pthread\_attr\_getstack, pthread\_attr\_setstack — get and set stack attributes

## 31690 SYNOPSIS

31691 THR #include &lt;pthread.h&gt;

```

31692 TSA TSS int pthread_attr_getstack(const pthread_attr_t *restrict attr,
31693 void **restrict stackaddr, size_t *restrict stacksize);
31694 int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
31695 size_t stacksize);

```

31696

## 31697 DESCRIPTION

31698 The *pthread\_attr\_getstack()* and *pthread\_attr\_setstack()* functions, respectively, shall get and set  
 31699 the thread creation stack attributes *stackaddr* and *stacksize* in the *attr* object.

31700 The stack attributes specify the area of storage to be used for the created thread's stack. The base  
 31701 (lowest addressable byte) of the storage shall be *stackaddr*, and the size of the storage shall be  
 31702 *stacksize* bytes. The *stacksize* shall be at least {PTHREAD\_STACK\_MIN}. The *stackaddr* shall be  
 31703 aligned appropriately to be used as a stack; for example, *pthread\_attr\_setstack()* may fail with  
 31704 [EINVAL] if (*stackaddr* & 0x7) is not 0. All pages within the stack described by *stackaddr* and  
 31705 *stacksize* shall be both readable and writable by the thread.

## 31706 RETURN VALUE

31707 Upon successful completion, these functions shall return a value of 0; otherwise, an error  
 31708 number shall be returned to indicate the error.

31709 The *pthread\_attr\_getstack()* function shall store the stack attribute values in *stackaddr* and *stacksize*  
 31710 if successful.

## 31711 ERRORS

31712 The *pthread\_attr\_setstack()* function shall fail if:

31713 [EINVAL] The value of *stacksize* is less than {PTHREAD\_STACK\_MIN} or exceeds an  
 31714 implementation-defined limit.

31715 The *pthread\_attr\_setstack()* function may fail if:

31716 [EINVAL] The value of *stackaddr* does not have proper alignment to be used as a stack, or  
 31717 if (*stackaddr* + *stacksize*) lacks proper alignment.

31718 [EACCES] The stack page(s) described by *stackaddr* and *stacksize* are not both readable  
 31719 and writable by the thread.

31720 These functions shall not return an error code of [EINTR].

## 31721 EXAMPLES

31722 None.

## 31723 APPLICATION USAGE

31724 These functions are appropriate for use by applications in an environment where the stack for a  
 31725 thread must be placed in some particular region of memory.

31726 While it might seem that an application could detect stack overflow by providing a protected  
 31727 page outside the specified stack region, this cannot be done portably. Implementations are free  
 31728 to place the thread's initial stack pointer anywhere within the specified region to accommodate  
 31729 the machine's stack pointer behavior and allocation requirements. Furthermore, on some  
 31730 architectures, such as the IA-64, "overflow" might mean that two separate stack pointers  
 31731 allocated within the region will overlap somewhere in the middle of the region.

31732 **RATIONALE**

31733           None.

31734 **FUTURE DIRECTIONS**

31735           None.

31736 **SEE ALSO**31737           *pthread\_attr\_init()*, *pthread\_attr\_setdetachstate()*, *pthread\_attr\_setstacksize()*, *pthread\_create()*, the  
31738           Base Definitions volume of IEEE Std 1003.1-2001, <limits.h>, <pthread.h>31739 **CHANGE HISTORY**

31740           First released in Issue 6. Developed as an XSI extension and brought into the BASE by IEEE

31741           PASC Interpretation 1003.1 #101.

31742 **NAME**

31743 pthread\_attr\_getstackaddr, pthread\_attr\_setstackaddr — get and set the stackaddr attribute

31744 **SYNOPSIS**

31745 THR TSA #include &lt;pthread.h&gt;

```
31746 OB int pthread_attr_getstackaddr(const pthread_attr_t *restrict attr,
31747 void **restrict stackaddr);
31748 int pthread_attr_setstackaddr(pthread_attr_t *attr, void *stackaddr);
31749
```

31750 **DESCRIPTION**31751 The *pthread\_attr\_getstackaddr()* and *pthread\_attr\_setstackaddr()* functions, respectively, shall get  
31752 and set the thread creation *stackaddr* attribute in the *attr* object.31753 The *stackaddr* attribute specifies the location of storage to be used for the created thread's stack.  
31754 The size of the storage shall be at least {PTHREAD\_STACK\_MIN}.31755 **RETURN VALUE**31756 Upon successful completion, *pthread\_attr\_getstackaddr()* and *pthread\_attr\_setstackaddr()* shall  
31757 return a value of 0; otherwise, an error number shall be returned to indicate the error.31758 The *pthread\_attr\_getstackaddr()* function stores the *stackaddr* attribute value in *stackaddr* if  
31759 successful.31760 **ERRORS**

31761 No errors are defined.

31762 These functions shall not return an error code of [EINTR].

31763 **EXAMPLES**

31764 None.

31765 **APPLICATION USAGE**

31766 The specification of the *stackaddr* attribute presents several ambiguities that make portable use of  
31767 these interfaces impossible. The description of the single address parameter as a “stack” does  
31768 not specify a particular relationship between the address and the “stack” implied by that  
31769 address. For example, the address may be taken as the low memory address of a buffer intended  
31770 for use as a stack, or it may be taken as the address to be used as the initial stack pointer register  
31771 value for the new thread. These two are not the same except for a machine on which the stack  
31772 grows “up” from low memory to high, and on which a “push” operation first stores the value in  
31773 memory and then increments the stack pointer register. Further, on a machine where the stack  
31774 grows “down” from high memory to low, interpretation of the address as the “low memory”  
31775 address requires a determination of the intended size of the stack. IEEE Std 1003.1-2001 has  
31776 introduced the new interfaces *pthread\_attr\_setstack()* and *pthread\_attr\_getstack()* to resolve these  
31777 ambiguities.

31778 **RATIONALE**

31779 None.

31780 **FUTURE DIRECTIONS**

31781 None.

31782 **SEE ALSO**

31783 *pthread\_attr\_destroy()*, *pthread\_attr\_getdetachstate()*, *pthread\_attr\_getstack()*,  
31784 *pthread\_attr\_getstacksize()*, *pthread\_attr\_setstack()*, *pthread\_create()*, the Base Definitions volume  
31785 of IEEE Std 1003.1-2001, <limits.h>, <pthread.h>

31786 **CHANGE HISTORY**

31787 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31788 **Issue 6**

31789 The *pthread\_attr\_getstackaddr()* and *pthread\_attr\_setstackaddr()* functions are marked as part of  
31790 the Threads and Thread Stack Address Attribute options.

31791 The **restrict** keyword is added to the *pthread\_attr\_getstackaddr()* prototype for alignment with the  
31792 ISO/IEC 9899:1999 standard.

31793 These functions are marked obsolescent.

31794 **NAME**

31795 pthread\_attr\_getstacksize, pthread\_attr\_setstacksize — get and set the stacksize attribute

31796 **SYNOPSIS**

31797 THR TSS #include <pthread.h>

```
31798 int pthread_attr_getstacksize(const pthread_attr_t *restrict attr,
31799 size_t *restrict stacksize);
31800 int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
31801
```

31802 **DESCRIPTION**

31803 The *pthread\_attr\_getstacksize()* and *pthread\_attr\_setstacksize()* functions, respectively, shall get  
31804 and set the thread creation *stacksize* attribute in the *attr* object.

31805 The *stacksize* attribute shall define the minimum stack size (in bytes) allocated for the created  
31806 threads stack.

31807 **RETURN VALUE**

31808 Upon successful completion, *pthread\_attr\_getstacksize()* and *pthread\_attr\_setstacksize()* shall  
31809 return a value of 0; otherwise, an error number shall be returned to indicate the error.

31810 The *pthread\_attr\_getstacksize()* function stores the *stacksize* attribute value in *stacksize* if  
31811 successful.

31812 **ERRORS**

31813 The *pthread\_attr\_setstacksize()* function shall fail if:

31814 [EINVAL] The value of *stacksize* is less than {PTHREAD\_STACK\_MIN} or exceeds a  
31815 system-imposed limit.

31816 These functions shall not return an error code of [EINTR].

31817 **EXAMPLES**

31818 None.

31819 **APPLICATION USAGE**

31820 None.

31821 **RATIONALE**

31822 None.

31823 **FUTURE DIRECTIONS**

31824 None.

31825 **SEE ALSO**

31826 *pthread\_attr\_destroy()*, *pthread\_attr\_getstackaddr()*, *pthread\_attr\_getdetachstate()*, *pthread\_create()*,  
31827 the Base Definitions volume of IEEE Std 1003.1-2001, <limits.h>, <pthread.h>

31828 **CHANGE HISTORY**

31829 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31830 **Issue 6**

31831 The *pthread\_attr\_getstacksize()* and *pthread\_attr\_setstacksize()* functions are marked as part of the  
31832 Threads and Thread Stack Size Attribute options.

31833 The **restrict** keyword is added to the *pthread\_attr\_getstacksize()* prototype for alignment with the  
31834 ISO/IEC 9899:1999 standard.

31835 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/43 is applied, correcting the margin code |  
31836 in the SYNOPSIS from TSA to TSS and updating the CHANGE HISTORY from “Thread Stack |  
31837 Address Attribute” option to “Thread Stack Size Attribute” option. |

31838 **NAME**

31839 pthread\_attr\_init — initialize the thread attributes object

31840 **SYNOPSIS**

31841 THR #include <pthread.h>

31842 int pthread\_attr\_init(pthread\_attr\_t \*attr);

31843

31844 **DESCRIPTION**

31845 Refer to *pthread\_attr\_destroy()*.

31846 **NAME**

31847 pthread\_attr\_setdetachstate — set the detachstate attribute

31848 **SYNOPSIS**

31849 THR #include &lt;pthread.h&gt;

31850 int pthread\_attr\_setdetachstate(pthread\_attr\_t \*attr, int detachstate);

31851

31852 **DESCRIPTION**31853 Refer to *pthread\_attr\_getdetachstate()*.

31854 **NAME**

31855 pthread\_attr\_setguardsize — set the thread guardsize attribute

31856 **SYNOPSIS**

31857 XSI #include <pthread.h>

```
31858 int pthread_attr_setguardsize(pthread_attr_t *attr,
31859 size_t guardsize);
```

31860

31861 **DESCRIPTION**

31862 Refer to *pthread\_attr\_getguardsize()*.

31863 **NAME**

31864 pthread\_attr\_setinheritsched — set the inheritsched attribute (**REALTIME THREADS**)

31865 **SYNOPSIS**

31866 THR TPS #include <pthread.h>

```
31867 int pthread_attr_setinheritsched(pthread_attr_t *attr,
31868 int inheritsched);
```

31869

31870 **DESCRIPTION**

31871 Refer to *pthread\_attr\_getinheritsched()*.

31872 **NAME**

31873 pthread\_attr\_setschedparam — set the schedparam attribute

31874 **SYNOPSIS**

31875 THR #include <pthread.h>

```
31876 int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
31877 const struct sched_param *restrict param);
```

31878

31879 **DESCRIPTION**

31880 Refer to *pthread\_attr\_getschedparam()*.

31881 **NAME**31882 pthread\_attr\_setschedpolicy — set the schedpolicy attribute (**REALTIME THREADS**)31883 **SYNOPSIS**

31884 THR TPS #include &lt;pthread.h&gt;

31885 int pthread\_attr\_setschedpolicy(pthread\_attr\_t \*attr, int policy);

31886

31887 **DESCRIPTION**31888 Refer to *pthread\_attr\_getschedpolicy()*.

31889 **NAME**

31890 pthread\_attr\_setscope — set the contention scope attribute (**REALTIME THREADS**)

31891 **SYNOPSIS**

31892 THR TPS #include <pthread.h>

31893 int pthread\_attr\_setscope(pthread\_attr\_t \*attr, int contention\_scope);

31894

31895 **DESCRIPTION**

31896 Refer to *pthread\_attr\_getscope()*.

31897 **NAME**

31898 pthread\_attr\_setstack — set the stack attribute

31899 **SYNOPSIS**

31900 xSI #include &lt;pthread.h&gt;

31901 int pthread\_attr\_setstack(pthread\_attr\_t \*attr, void \*stackaddr,  
31902 size\_t stacksize);

31903

31904 **DESCRIPTION**31905 Refer to *pthread\_attr\_getstack()*.

31906 **NAME**

31907 pthread\_attr\_setstackaddr — set the stackaddr attribute

31908 **SYNOPSIS**

31909 THR TSA #include <pthread.h>

31910 OB int pthread\_attr\_setstackaddr(pthread\_attr\_t \*attr, void \*stackaddr);

31911

31912 **DESCRIPTION**

31913 Refer to *pthread\_attr\_getstackaddr()*.

31914 **NAME**

31915 pthread\_attr\_setstacksize — set the stacksize attribute

31916 **SYNOPSIS**

31917 THR TSS #include &lt;pthread.h&gt;

31918 int pthread\_attr\_setstacksize(pthread\_attr\_t \*attr, size\_t stacksize);

31919

31920 **DESCRIPTION**31921 Refer to *pthread\_attr\_getstacksize()*.

## 31922 NAME

31923 pthread\_barrier\_destroy, pthread\_barrier\_init — destroy and initialize a barrier object  
 31924 (ADVANCED REALTIME THREADS)

## 31925 SYNOPSIS

```
31926 THR BAR #include <pthread.h>
31927
31927 int pthread_barrier_destroy(pthread_barrier_t *barrier);
31928 int pthread_barrier_init(pthread_barrier_t *restrict barrier,
31929 const pthread_barrierattr_t *restrict attr, unsigned count);
31930
```

## 31931 DESCRIPTION

31932 The *pthread\_barrier\_destroy()* function shall destroy the barrier referenced by *barrier* and release  
 31933 any resources used by the barrier. The effect of subsequent use of the barrier is undefined until  
 31934 the barrier is reinitialized by another call to *pthread\_barrier\_init()*. An implementation may use  
 31935 this function to set *barrier* to an invalid value. The results are undefined if  
 31936 *pthread\_barrier\_destroy()* is called when any thread is blocked on the barrier, or if this function is  
 31937 called with an uninitialized barrier.

31938 The *pthread\_barrier\_init()* function shall allocate any resources required to use the barrier  
 31939 referenced by *barrier* and shall initialize the barrier with attributes referenced by *attr*. If *attr* is  
 31940 NULL, the default barrier attributes shall be used; the effect is the same as passing the address of  
 31941 a default barrier attributes object. The results are undefined if *pthread\_barrier\_init()* is called  
 31942 when any thread is blocked on the barrier (that is, has not returned from the  
 31943 *pthread\_barrier\_wait()* call). The results are undefined if a barrier is used without first being  
 31944 initialized. The results are undefined if *pthread\_barrier\_init()* is called specifying an already  
 31945 initialized barrier.

31946 The *count* argument specifies the number of threads that must call *pthread\_barrier\_wait()* before  
 31947 any of them successfully return from the call. The value specified by *count* must be greater than  
 31948 zero.

31949 If the *pthread\_barrier\_init()* function fails, the barrier shall not be initialized and the contents of  
 31950 *barrier* are undefined.

31951 Only the object referenced by *barrier* may be used for performing synchronization. The result of  
 31952 referring to copies of that object in calls to *pthread\_barrier\_destroy()* or *pthread\_barrier\_wait()* is  
 31953 undefined.

## 31954 RETURN VALUE

31955 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
 31956 be returned to indicate the error.

## 31957 ERRORS

31958 The *pthread\_barrier\_destroy()* function may fail if:

31959 [EBUSY] The implementation has detected an attempt to destroy a barrier while it is in  
 31960 use (for example, while being used in a *pthread\_barrier\_wait()* call) by another  
 31961 thread.

31962 [EINVAL] The value specified by *barrier* is invalid.

31963 The *pthread\_barrier\_init()* function shall fail if:

31964 [EAGAIN] The system lacks the necessary resources to initialize another barrier.

31965 [EINVAL] The value specified by *count* is equal to zero.

- 31966 [ENOMEM] Insufficient memory exists to initialize the barrier.
- 31967 The *pthread\_barrier\_init()* function may fail if:
- 31968 [EBUSY] The implementation has detected an attempt to reinitialize a barrier while it is  
31969 in use (for example, while being used in a *pthread\_barrier\_wait()* call) by  
31970 another thread.
- 31971 [EINVAL] The value specified by *attr* is invalid.
- 31972 These functions shall not return an error code of [EINTR].
- 31973 **EXAMPLES**
- 31974 None.
- 31975 **APPLICATION USAGE**
- 31976 The *pthread\_barrier\_destroy()* and *pthread\_barrier\_init()* functions are part of the Barriers option  
31977 and need not be provided on all implementations.
- 31978 **RATIONALE**
- 31979 None.
- 31980 **FUTURE DIRECTIONS**
- 31981 None.
- 31982 **SEE ALSO**
- 31983 *pthread\_barrier\_wait()*, the Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>
- 31984 **CHANGE HISTORY**
- 31985 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

## 31986 NAME

31987 pthread\_barrier\_wait — synchronize at a barrier (ADVANCED REALTIME THREADS)

## 31988 SYNOPSIS

```
31989 THR BAR #include <pthread.h>
```

```
31990 int pthread_barrier_wait(pthread_barrier_t *barrier);
```

31991

## 31992 DESCRIPTION

31993 The *pthread\_barrier\_wait()* function shall synchronize participating threads at the barrier  
31994 referenced by *barrier*. The calling thread shall block until the required number of threads have  
31995 called *pthread\_barrier\_wait()* specifying the barrier.

31996 When the required number of threads have called *pthread\_barrier\_wait()* specifying the barrier,  
31997 the constant PTHREAD\_BARRIER\_SERIAL\_THREAD shall be returned to one unspecified  
31998 thread and zero shall be returned to each of the remaining threads. At this point, the barrier shall  
31999 be reset to the state it had as a result of the most recent *pthread\_barrier\_init()* function that  
32000 referenced it.

32001 The constant PTHREAD\_BARRIER\_SERIAL\_THREAD is defined in <pthread.h> and its value  
32002 shall be distinct from any other value returned by *pthread\_barrier\_wait()*.

32003 The results are undefined if this function is called with an uninitialized barrier.

32004 If a signal is delivered to a thread blocked on a barrier, upon return from the signal handler the  
32005 thread shall resume waiting at the barrier if the barrier wait has not completed (that is, if the  
32006 required number of threads have not arrived at the barrier during the execution of the signal  
32007 handler); otherwise, the thread shall continue as normal from the completed barrier wait. Until  
32008 the thread in the signal handler returns from it, it is unspecified whether other threads may  
32009 proceed past the barrier once they have all reached it.

32010 A thread that has blocked on a barrier shall not prevent any unblocked thread that is eligible to  
32011 use the same processing resources from eventually making forward progress in its execution.  
32012 Eligibility for processing resources shall be determined by the scheduling policy.

## 32013 RETURN VALUE

32014 Upon successful completion, the *pthread\_barrier\_wait()* function shall return  
32015 PTHREAD\_BARRIER\_SERIAL\_THREAD for a single (arbitrary) thread synchronized at the  
32016 barrier and zero for each of the other threads. Otherwise, an error number shall be returned to  
32017 indicate the error.

## 32018 ERRORS

32019 The *pthread\_barrier\_wait()* function may fail if:

32020 [EINVAL] The value specified by *barrier* does not refer to an initialized barrier object.

32021 This function shall not return an error code of [EINTR].

## 32022 EXAMPLES

32023 None.

## 32024 APPLICATION USAGE

32025 Applications using this function may be subject to priority inversion, as discussed in the Base  
32026 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

32027 The *pthread\_barrier\_wait()* function is part of the Barriers option and need not be provided on all  
32028 implementations.

32029 **RATIONALE**

32030           None.

32031 **FUTURE DIRECTIONS**

32032           None.

32033 **SEE ALSO**32034           *pthread\_barrier\_destroy()*, the Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>32035 **CHANGE HISTORY**

32036           First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

32037           In the SYNOPSIS, the inclusion of &lt;sys/types.h&gt; is no longer required.

## 32038 NAME

32039 pthread\_barrierattr\_destroy, pthread\_barrierattr\_init — destroy and initialize the barrier  
32040 attributes object (**ADVANCED REALTIME THREADS**)

## 32041 SYNOPSIS

```
32042 THR BAR #include <pthread.h>
```

```
32043 int pthread_barrierattr_destroy(pthread_barrierattr_t *attr);
```

```
32044 int pthread_barrierattr_init(pthread_barrierattr_t *attr);
```

32045

## 32046 DESCRIPTION

32047 The *pthread\_barrierattr\_destroy()* function shall destroy a barrier attributes object. A destroyed  
32048 *attr* attributes object can be reinitialized using *pthread\_barrierattr\_init()*; the results of otherwise  
32049 referencing the object after it has been destroyed are undefined. An implementation may cause  
32050 *pthread\_barrierattr\_destroy()* to set the object referenced by *attr* to an invalid value.

32051 The *pthread\_barrierattr\_init()* function shall initialize a barrier attributes object *attr* with the  
32052 default value for all of the attributes defined by the implementation.

32053 Results are undefined if *pthread\_barrierattr\_init()* is called specifying an already initialized *attr*  
32054 attributes object.

32055 After a barrier attributes object has been used to initialize one or more barriers, any function  
32056 affecting the attributes object (including destruction) shall not affect any previously initialized  
32057 barrier.

## 32058 RETURN VALUE

32059 If successful, the *pthread\_barrierattr\_destroy()* and *pthread\_barrierattr\_init()* functions shall return  
32060 zero; otherwise, an error number shall be returned to indicate the error.

## 32061 ERRORS

32062 The *pthread\_barrierattr\_destroy()* function may fail if:

32063 [EINVAL] The value specified by *attr* is invalid.

32064 The *pthread\_barrierattr\_init()* function shall fail if:

32065 [ENOMEM] Insufficient memory exists to initialize the barrier attributes object.

32066 These functions shall not return an error code of [EINTR].

## 32067 EXAMPLES

32068 None.

## 32069 APPLICATION USAGE

32070 The *pthread\_barrierattr\_destroy()* and *pthread\_barrierattr\_init()* functions are part of the Barriers  
32071 option and need not be provided on all implementations.

## 32072 RATIONALE

32073 None.

## 32074 FUTURE DIRECTIONS

32075 None.

## 32076 SEE ALSO

32077 *pthread\_barrierattr\_getpshared()*, *pthread\_barrierattr\_setpshared()*, the Base Definitions volume of  
32078 IEEE Std 1003.1-2001, <pthread.h>.

32079 **CHANGE HISTORY**

32080 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

32081 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

## 32082 NAME

32083 pthread\_barrierattr\_getpshared, pthread\_barrierattr\_setpshared — get and set the process-  
 32084 shared attribute of the barrier attributes object (**ADVANCED REALTIME THREADS**)

## 32085 SYNOPSIS

```
32086 THR #include <pthread.h>
32087 BAR TSH int pthread_barrierattr_getpshared(const pthread_barrierattr_t *
32088 restrict attr, int *restrict pshared);
32089 int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
32090 int pshared);
32091
```

## 32092 DESCRIPTION

32093 The *pthread\_barrierattr\_getpshared()* function shall obtain the value of the *process-shared* attribute  
 32094 from the attributes object referenced by *attr*. The *pthread\_barrierattr\_setpshared()* function shall  
 32095 set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

32096 The *process-shared* attribute is set to PTHREAD\_PROCESS\_SHARED to permit a barrier to be  
 32097 operated upon by any thread that has access to the memory where the barrier is allocated. If the  
 32098 *process-shared* attribute is PTHREAD\_PROCESS\_PRIVATE, the barrier shall only be operated  
 32099 upon by threads created within the same process as the thread that initialized the barrier; if  
 32100 threads of different processes attempt to operate on such a barrier, the behavior is undefined.  
 32101 The default value of the attribute shall be PTHREAD\_PROCESS\_PRIVATE. Both constants  
 32102 PTHREAD\_PROCESS\_SHARED and PTHREAD\_PROCESS\_PRIVATE are defined in  
 32103 **<pthread.h>**.

32104 Additional attributes, their default values, and the names of the associated functions to get and  
 32105 set those attribute values are implementation-defined.

## 32106 RETURN VALUE

32107 If successful, the *pthread\_barrierattr\_getpshared()* function shall return zero and store the value of  
 32108 the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise,  
 32109 an error number shall be returned to indicate the error.

32110 If successful, the *pthread\_barrierattr\_setpshared()* function shall return zero; otherwise, an error  
 32111 number shall be returned to indicate the error.

## 32112 ERRORS

32113 These functions may fail if:

32114 [EINVAL] The value specified by *attr* is invalid.

32115 The *pthread\_barrierattr\_setpshared()* function may fail if:

32116 [EINVAL] The new value specified for the *process-shared* attribute is not one of the legal  
 32117 values PTHREAD\_PROCESS\_SHARED or PTHREAD\_PROCESS\_PRIVATE.

32118 These functions shall not return an error code of [EINTR].

32119 **EXAMPLES**

32120 None.

32121 **APPLICATION USAGE**

32122 The *pthread\_barrierattr\_getpshared()* and *pthread\_barrierattr\_setpshared()* functions are part of the  
32123 Barriers option and need not be provided on all implementations.

32124 **RATIONALE**

32125 None.

32126 **FUTURE DIRECTIONS**

32127 None.

32128 **SEE ALSO**

32129 *pthread\_barrier\_destroy()*, *pthread\_barrierattr\_destroy()*, *pthread\_barrierattr\_init()*, the Base  
32130 Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

32131 **CHANGE HISTORY**

32132 First released in Issue 6. Derived from IEEE Std 1003.1j-2000

32133 **NAME**

32134 pthread\_barrierattr\_init — initialize the barrier attributes object (**ADVANCED REALTIME**  
32135 **THREADS**)

32136 **SYNOPSIS**

32137 THR BAR #include <pthread.h>

32138 int pthread\_barrierattr\_init(pthread\_barrierattr\_t \*attr);

32139

32140 **DESCRIPTION**

32141 Refer to *pthread\_barrierattr\_destroy()*.

32142 **NAME**

32143 pthread\_barrierattr\_setpshared — set the process-shared attribute of the barrier attributes object  
32144 (**ADVANCED REALTIME THREADS**)

32145 **SYNOPSIS**

```
32146 THR #include <pthread.h>
```

```
32147 BAR TSH int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
32148 int pshared);
```

32149

32150 **DESCRIPTION**

32151 Refer to *pthread\_barrierattr\_getpshared()*.

32152 **NAME**

32153 pthread\_cancel — cancel execution of a thread

32154 **SYNOPSIS**

32155 THR #include &lt;pthread.h&gt;

32156 int pthread\_cancel(pthread\_t thread);

32157

32158 **DESCRIPTION**

32159 The *pthread\_cancel()* function shall request that *thread* be canceled. The target thread's  
32160 cancelability state and type determines when the cancellation takes effect. When the cancellation  
32161 is acted on, the cancellation cleanup handlers for *thread* shall be called. When the last  
32162 cancellation cleanup handler returns, the thread-specific data destructor functions shall be called  
32163 for *thread*. When the last destructor function returns, *thread* shall be terminated.

32164 The cancellation processing in the target thread shall run asynchronously with respect to the  
32165 calling thread returning from *pthread\_cancel()*.

32166 **RETURN VALUE**

32167 If successful, the *pthread\_cancel()* function shall return zero; otherwise, an error number shall be  
32168 returned to indicate the error.

32169 **ERRORS**32170 The *pthread\_cancel()* function may fail if:

32171 [ESRCH] No thread could be found corresponding to that specified by the given thread  
32172 ID.

32173 The *pthread\_cancel()* function shall not return an error code of [EINTR].32174 **EXAMPLES**

32175 None.

32176 **APPLICATION USAGE**

32177 None.

32178 **RATIONALE**

32179 Two alternative functions were considered for sending the cancellation notification to a thread.  
32180 One would be to define a new SIGCANCEL signal that had the cancellation semantics when  
32181 delivered; the other was to define the new *pthread\_cancel()* function, which would trigger the  
32182 cancellation semantics.

32183 The advantage of a new signal was that so much of the delivery criteria were identical to that  
32184 used when trying to deliver a signal that making cancellation notification a signal was seen as  
32185 consistent. Indeed, many implementations implement cancellation using a special signal. On the  
32186 other hand, there would be no signal functions that could be used with this signal except  
32187 *pthread\_kill()*, and the behavior of the delivered cancellation signal would be unlike any  
32188 previously existing defined signal.

32189 The benefits of a special function include the recognition that this signal would be defined  
32190 because of the similar delivery criteria and that this is the only common behavior between a  
32191 cancellation request and a signal. In addition, the cancellation delivery mechanism does not  
32192 have to be implemented as a signal. There are also strong, if not stronger, parallels with  
32193 language exception mechanisms than with signals that are potentially obscured if the delivery  
32194 mechanism is visibly closer to signals.

32195 In the end, it was considered that as there were so many exceptions to the use of the new signal  
32196 with existing signals functions it would be misleading. A special function has resolved this

32197 problem. This function was carefully defined so that an implementation wishing to provide the  
32198 cancellation functions on top of signals could do so. The special function also means that  
32199 implementations are not obliged to implement cancellation with signals.

32200 **FUTURE DIRECTIONS**

32201 None.

32202 **SEE ALSO**

32203 *pthread\_exit()*, *pthread\_cond\_timedwait()*, *pthread\_join()*, *pthread\_setcancelstate()*, the Base  
32204 Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

32205 **CHANGE HISTORY**

32206 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32207 **Issue 6**

32208 The *pthread\_cancel()* function is marked as part of the Threads option.

## 32209 NAME

32210 pthread\_cleanup\_pop, pthread\_cleanup\_push — establish cancellation handlers

## 32211 SYNOPSIS

32212 THR #include &lt;pthread.h&gt;

32213 void pthread\_cleanup\_pop(int execute);

32214 void pthread\_cleanup\_push(void (\*routine)(void\*), void \*arg);

32215

## 32216 DESCRIPTION

32217 The *pthread\_cleanup\_pop()* function shall remove the routine at the top of the calling thread's  
32218 cancellation cleanup stack and optionally invoke it (if *execute* is non-zero).32219 The *pthread\_cleanup\_push()* function shall push the specified cancellation cleanup handler *routine*  
32220 onto the calling thread's cancellation cleanup stack. The cancellation cleanup handler shall be  
32221 popped from the cancellation cleanup stack and invoked with the argument *arg* when:

- 32222 • The thread exits (that is, calls *pthread\_exit()*).
- 32223 • The thread acts upon a cancellation request.
- 32224 • The thread calls *pthread\_cleanup\_pop()* with a non-zero *execute* argument.

32225 These functions may be implemented as macros. The application shall ensure that they appear  
32226 as statements, and in pairs within the same lexical scope (that is, the *pthread\_cleanup\_push()*  
32227 macro may be thought to expand to a token list whose first token is '{' with  
32228 *pthread\_cleanup\_pop()* expanding to a token list whose last token is the corresponding '}').32229 The effect of calling *longjmp()* or *siglongjmp()* is undefined if there have been any calls to  
32230 *pthread\_cleanup\_push()* or *pthread\_cleanup\_pop()* made without the matching call since the jump  
32231 buffer was filled. The effect of calling *longjmp()* or *siglongjmp()* from inside a cancellation  
32232 cleanup handler is also undefined unless the jump buffer was also filled in the cancellation  
32233 cleanup handler.

## 32234 RETURN VALUE

32235 The *pthread\_cleanup\_push()* and *pthread\_cleanup\_pop()* functions shall not return a value.

## 32236 ERRORS

32237 No errors are defined.

32238 These functions shall not return an error code of [EINTR].

## 32239 EXAMPLES

32240 The following is an example using thread primitives to implement a cancelable, writers-priority  
32241 read-write lock:

```

32242 typedef struct {
32243 pthread_mutex_t lock;
32244 pthread_cond_t rcond,
32245 wcond;
32246 int lock_count; /* < 0 .. Held by writer. */
32247 /* > 0 .. Held by lock_count readers. */
32248 /* = 0 .. Held by nobody. */
32249 int waiting_writers; /* Count of waiting writers. */
32250 } rwlock;
32251
32252 void
32253 waiting_reader_cleanup(void *arg)
32254 {

```

```

32254 rwlock *l;
32255 l = (rwlock *) arg;
32256 pthread_mutex_unlock(&l->lock);
32257 }
32258 void
32259 lock_for_read(rwlock *l)
32260 {
32261 pthread_mutex_lock(&l->lock);
32262 pthread_cleanup_push(waiting_reader_cleanup, l);
32263 while ((l->lock_count < 0) && (l->waiting_writers != 0))
32264 pthread_cond_wait(&l->rcond, &l->lock);
32265 l->lock_count++;
32266 /*
32267 * Note the pthread_cleanup_pop executes
32268 * waiting_reader_cleanup.
32269 */
32270 pthread_cleanup_pop(1);
32271 }
32272 void
32273 release_read_lock(rwlock *l)
32274 {
32275 pthread_mutex_lock(&l->lock);
32276 if (--l->lock_count == 0)
32277 pthread_cond_signal(&l->wcond);
32278 pthread_mutex_unlock(l);
32279 }
32280 void
32281 waiting_writer_cleanup(void *arg)
32282 {
32283 rwlock *l;
32284 l = (rwlock *) arg;
32285 if ((--l->waiting_writers == 0) && (l->lock_count >= 0)) {
32286 /*
32287 * This only happens if we have been canceled.
32288 */
32289 pthread_cond_broadcast(&l->wcond);
32290 }
32291 pthread_mutex_unlock(&l->lock);
32292 }
32293 void
32294 lock_for_write(rwlock *l)
32295 {
32296 pthread_mutex_lock(&l->lock);
32297 l->waiting_writers++;
32298 pthread_cleanup_push(waiting_writer_cleanup, l);
32299 while (l->lock_count != 0)
32300 pthread_cond_wait(&l->wcond, &l->lock);
32301 l->lock_count = -1;
32302 /*

```

```
32303 * Note the pthread_cleanup_pop executes
32304 * waiting_writer_cleanup.
32305 */
32306 pthread_cleanup_pop(1);
32307 }

32308 void
32309 release_write_lock(rwlock *l)
32310 {
32311 pthread_mutex_lock(&l->lock);
32312 l->lock_count = 0;
32313 if (l->waiting_writers == 0)
32314 pthread_cond_broadcast(&l->rcond)
32315 else
32316 pthread_cond_signal(&l->wcond);
32317 pthread_mutex_unlock(&l->lock);
32318 }

32319 /*
32320 * This function is called to initialize the read/write lock.
32321 */
32322 void
32323 initialize_rwlock(rwlock *l)
32324 {
32325 pthread_mutex_init(&l->lock, pthread_mutexattr_default);
32326 pthread_cond_init(&l->wcond, pthread_condattr_default);
32327 pthread_cond_init(&l->rcond, pthread_condattr_default);
32328 l->lock_count = 0;
32329 l->waiting_writers = 0;
32330 }

32331 reader_thread()
32332 {
32333 lock_for_read(&lock);
32334 pthread_cleanup_push(release_read_lock, &lock);
32335 /*
32336 * Thread has read lock.
32337 */
32338 pthread_cleanup_pop(1);
32339 }

32340 writer_thread()
32341 {
32342 lock_for_write(&lock);
32343 pthread_cleanup_push(release_write_lock, &lock);
32344 /*
32345 * Thread has write lock.
32346 */
32347 pthread_cleanup_pop(1);
32348 }
```

32349 **APPLICATION USAGE**

32350 The two routines that push and pop cancellation cleanup handlers, *pthread\_cleanup\_push()* and  
 32351 *pthread\_cleanup\_pop()*, can be thought of as left and right parentheses. They always need to be  
 32352 matched.

32353 **RATIONALE**

32354 The restriction that the two routines that push and pop cancellation cleanup handlers,  
 32355 *pthread\_cleanup\_push()* and *pthread\_cleanup\_pop()*, have to appear in the same lexical scope  
 32356 allows for efficient macro or compiler implementations and efficient storage management. A  
 32357 sample implementation of these routines as macros might look like this:

```
32358 #define pthread_cleanup_push(rtn,arg) { \
32359 struct _pthread_handler_rec __cleanup_handler, **__head; \
32360 __cleanup_handler.rtn = rtn; \
32361 __cleanup_handler.arg = arg; \
32362 (void) pthread_getspecific(_pthread_handler_key, &__head); \
32363 __cleanup_handler.next = *__head; \
32364 *__head = &__cleanup_handler;
32365
32366 #define pthread_cleanup_pop(ex) \
32367 *__head = __cleanup_handler.next; \
32368 if (ex) (*__cleanup_handler.rtn)(__cleanup_handler.arg); \
32369 }
```

32369 A more ambitious implementation of these routines might do even better by allowing the  
 32370 compiler to note that the cancellation cleanup handler is a constant and can be expanded inline.

32371 This volume of IEEE Std 1003.1-2001 currently leaves unspecified the effect of calling *longjmp()*  
 32372 from a signal handler executing in a POSIX System Interfaces function. If an implementation  
 32373 wants to allow this and give the programmer reasonable behavior, the *longjmp()* function has to  
 32374 call all cancellation cleanup handlers that have been pushed but not popped since the time  
 32375 *setjmp()* was called.

32376 Consider a multi-threaded function called by a thread that uses signals. If a signal were  
 32377 delivered to a signal handler during the operation of *qsort()* and that handler were to call  
 32378 *longjmp()* (which, in turn, did *not* call the cancellation cleanup handlers) the helper threads  
 32379 created by the *qsort()* function would not be canceled. Instead, they would continue to execute  
 32380 and write into the argument array even though the array might have been popped off the stack.

32381 Note that the specified cleanup handling mechanism is especially tied to the C language and,  
 32382 while the requirement for a uniform mechanism for expressing cleanup is language-  
 32383 independent, the mechanism used in other languages may be quite different. In addition, this  
 32384 mechanism is really only necessary due to the lack of a real exception mechanism in the C  
 32385 language, which would be the ideal solution.

32386 There is no notion of a cancellation cleanup-safe function. If an application has no cancellation  
 32387 points in its signal handlers, blocks any signal whose handler may have cancellation points  
 32388 while calling async-unsafe functions, or disables cancellation while calling async-unsafe  
 32389 functions, all functions may be safely called from cancellation cleanup routines.

32390 **FUTURE DIRECTIONS**

32391 None.

32392 **SEE ALSO**

32393 *pthread\_cancel()*, *pthread\_setcancelstate()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
 32394 <pthread.h>

## 32395 CHANGE HISTORY

32396 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

## 32397 Issue 6

32398 The *pthread\_cleanup\_pop()* and *pthread\_cleanup\_push()* functions are marked as part of the  
32399 Threads option.

32400 The APPLICATION USAGE section is added.

32401 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

32402 **NAME**

32403 pthread\_cond\_broadcast, pthread\_cond\_signal — broadcast or signal a condition

32404 **SYNOPSIS**

32405 THR #include &lt;pthread.h&gt;

32406 int pthread\_cond\_broadcast(pthread\_cond\_t \*cond);

32407 int pthread\_cond\_signal(pthread\_cond\_t \*cond);

32408

32409 **DESCRIPTION**

32410 These functions shall unblock threads blocked on a condition variable.

32411 The *pthread\_cond\_broadcast()* function shall unblock all threads currently blocked on the  
32412 specified condition variable *cond*.32413 The *pthread\_cond\_signal()* function shall unblock at least one of the threads that are blocked on  
32414 the specified condition variable *cond* (if any threads are blocked on *cond*).32415 If more than one thread is blocked on a condition variable, the scheduling policy shall determine  
32416 the order in which threads are unblocked. When each thread unblocked as a result of a  
32417 *pthread\_cond\_broadcast()* or *pthread\_cond\_signal()* returns from its call to *pthread\_cond\_wait()* or  
32418 *pthread\_cond\_timedwait()*, the thread shall own the mutex with which it called  
32419 *pthread\_cond\_wait()* or *pthread\_cond\_timedwait()*. The thread(s) that are unblocked shall contend  
32420 for the mutex according to the scheduling policy (if applicable), and as if each had called  
32421 *pthread\_mutex\_lock()*.32422 The *pthread\_cond\_broadcast()* or *pthread\_cond\_signal()* functions may be called by a thread  
32423 whether or not it currently owns the mutex that threads calling *pthread\_cond\_wait()* or  
32424 *pthread\_cond\_timedwait()* have associated with the condition variable during their waits;  
32425 however, if predictable scheduling behavior is required, then that mutex shall be locked by the  
32426 thread calling *pthread\_cond\_broadcast()* or *pthread\_cond\_signal()*.32427 The *pthread\_cond\_broadcast()* and *pthread\_cond\_signal()* functions shall have no effect if there are  
32428 no threads currently blocked on *cond*.32429 **RETURN VALUE**32430 If successful, the *pthread\_cond\_broadcast()* and *pthread\_cond\_signal()* functions shall return zero;  
32431 otherwise, an error number shall be returned to indicate the error.32432 **ERRORS**32433 The *pthread\_cond\_broadcast()* and *pthread\_cond\_signal()* function may fail if:32434 [EINVAL] The value *cond* does not refer to an initialized condition variable.

32435 These functions shall not return an error code of [EINTR].

32436 **EXAMPLES**

32437 None.

32438 **APPLICATION USAGE**32439 The *pthread\_cond\_broadcast()* function is used whenever the shared-variable state has been  
32440 changed in a way that more than one thread can proceed with its task. Consider a single  
32441 producer/multiple consumer problem, where the producer can insert multiple items on a list  
32442 that is accessed one item at a time by the consumers. By calling the *pthread\_cond\_broadcast()*  
32443 function, the producer would notify all consumers that might be waiting, and thereby the  
32444 application would receive more throughput on a multi-processor. In addition,  
32445 *pthread\_cond\_broadcast()* makes it easier to implement a read-write lock. The  
32446 *pthread\_cond\_broadcast()* function is needed in order to wake up all waiting readers when a

32447 writer releases its lock. Finally, the two-phase commit algorithm can use this broadcast function  
32448 to notify all clients of an impending transaction commit.

32449 It is not safe to use the *pthread\_cond\_signal()* function in a signal handler that is invoked  
32450 asynchronously. Even if it were safe, there would still be a race between the test of the Boolean  
32451 *pthread\_cond\_wait()* that could not be efficiently eliminated.

32452 Mutexes and condition variables are thus not suitable for releasing a waiting thread by signaling  
32453 from code running in a signal handler.

#### 32454 RATIONALE

##### 32455 Multiple Awakenings by Condition Signal

32456 On a multi-processor, it may be impossible for an implementation of *pthread\_cond\_signal()* to  
32457 avoid the unblocking of more than one thread blocked on a condition variable. For example,  
32458 consider the following partial implementation of *pthread\_cond\_wait()* and *pthread\_cond\_signal()*,  
32459 executed by two threads in the order given. One thread is trying to wait on the condition  
32460 variable, another is concurrently executing *pthread\_cond\_signal()*, while a third thread is already  
32461 waiting.

```
32462 pthread_cond_wait(mutex, cond):
32463 value = cond->value; /* 1 */
32464 pthread_mutex_unlock(mutex); /* 2 */
32465 pthread_mutex_lock(cond->mutex); /* 10 */
32466 if (value == cond->value) { /* 11 */
32467 me->next_cond = cond->waiter;
32468 cond->waiter = me;
32469 pthread_mutex_unlock(cond->mutex);
32470 unable_to_run(me);
32471 } else
32472 pthread_mutex_unlock(cond->mutex); /* 12 */
32473 pthread_mutex_lock(mutex); /* 13 */

32474 pthread_cond_signal(cond):
32475 pthread_mutex_lock(cond->mutex); /* 3 */
32476 cond->value++; /* 4 */
32477 if (cond->waiter) { /* 5 */
32478 sleeper = cond->waiter; /* 6 */
32479 cond->waiter = sleeper->next_cond; /* 7 */
32480 able_to_run(sleeper); /* 8 */
32481 }
32482 pthread_mutex_unlock(cond->mutex); /* 9 */
```

32483 The effect is that more than one thread can return from its call to *pthread\_cond\_wait()* or  
32484 *pthread\_cond\_timedwait()* as a result of one call to *pthread\_cond\_signal()*. This effect is called  
32485 “spurious wakeup”. Note that the situation is self-correcting in that the number of threads that  
32486 are so awakened is finite; for example, the next thread to call *pthread\_cond\_wait()* after the  
32487 sequence of events above blocks.

32488 While this problem could be resolved, the loss of efficiency for a fringe condition that occurs  
32489 only rarely is unacceptable, especially given that one has to check the predicate associated with a  
32490 condition variable anyway. Correcting this problem would unnecessarily reduce the degree of  
32491 concurrency in this basic building block for all higher-level synchronization operations.

32492 An added benefit of allowing spurious wakeups is that applications are forced to code a  
32493 predicate-testing-loop around the condition wait. This also makes the application tolerate

32494 superfluous condition broadcasts or signals on the same condition variable that may be coded in  
32495 some other part of the application. The resulting applications are thus more robust. Therefore,  
32496 IEEE Std 1003.1-2001 explicitly documents that spurious wakeups may occur.

32497 **FUTURE DIRECTIONS**

32498 None.

32499 **SEE ALSO**

32500 *pthread\_cond\_destroy()*, *pthread\_cond\_timedwait()*, the Base Definitions volume of  
32501 IEEE Std 1003.1-2001, <pthread.h>

32502 **CHANGE HISTORY**

32503 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32504 **Issue 6**

32505 The *pthread\_cond\_broadcast()* and *pthread\_cond\_signal()* functions are marked as part of the  
32506 Threads option.

32507 The APPLICATION USAGE section is added.

## 32508 NAME

32509 pthread\_cond\_destroy, pthread\_cond\_init — destroy and initialize condition variables

## 32510 SYNOPSIS

32511 THR #include &lt;pthread.h&gt;

```

32512 int pthread_cond_destroy(pthread_cond_t *cond);
32513 int pthread_cond_init(pthread_cond_t *restrict cond,
32514 const pthread_condattr_t *restrict attr);
32515 pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
32516

```

## 32517 DESCRIPTION

32518 The *pthread\_cond\_destroy()* function shall destroy the given condition variable specified by *cond*;  
 32519 the object becomes, in effect, uninitialized. An implementation may cause *pthread\_cond\_destroy()*  
 32520 to set the object referenced by *cond* to an invalid value. A destroyed condition variable object can  
 32521 be reinitialized using *pthread\_cond\_init()*; the results of otherwise referencing the object after it  
 32522 has been destroyed are undefined.

32523 It shall be safe to destroy an initialized condition variable upon which no threads are currently  
 32524 blocked. Attempting to destroy a condition variable upon which other threads are currently  
 32525 blocked results in undefined behavior.

32526 The *pthread\_cond\_init()* function shall initialize the condition variable referenced by *cond* with  
 32527 attributes referenced by *attr*. If *attr* is NULL, the default condition variable attributes shall be  
 32528 used; the effect is the same as passing the address of a default condition variable attributes  
 32529 object. Upon successful initialization, the state of the condition variable shall become initialized.

32530 Only *cond* itself may be used for performing synchronization. The result of referring to copies of  
 32531 *cond* in calls to *pthread\_cond\_wait()*, *pthread\_cond\_timedwait()*, *pthread\_cond\_signal()*,  
 32532 *pthread\_cond\_broadcast()*, and *pthread\_cond\_destroy()* is undefined.

32533 Attempting to initialize an already initialized condition variable results in undefined behavior.

32534 In cases where default condition variable attributes are appropriate, the macro  
 32535 PTHREAD\_COND\_INITIALIZER can be used to initialize condition variables that are statically  
 32536 allocated. The effect shall be equivalent to dynamic initialization by a call to *pthread\_cond\_init()*  
 32537 with parameter *attr* specified as NULL, except that no error checks are performed.

## 32538 RETURN VALUE

32539 If successful, the *pthread\_cond\_destroy()* and *pthread\_cond\_init()* functions shall return zero;  
 32540 otherwise, an error number shall be returned to indicate the error.

32541 The [EBUSY] and [EINVAL] error checks, if implemented, shall act as if they were performed  
 32542 immediately at the beginning of processing for the function and caused an error return prior to  
 32543 modifying the state of the condition variable specified by *cond*.

## 32544 ERRORS

32545 The *pthread\_cond\_destroy()* function may fail if:

32546 [EBUSY] The implementation has detected an attempt to destroy the object referenced  
 32547 by *cond* while it is referenced (for example, while being used in a  
 32548 *pthread\_cond\_wait()* or *pthread\_cond\_timedwait()*) by another thread.

32549 [EINVAL] The value specified by *cond* is invalid.

32550 The *pthread\_cond\_init()* function shall fail if:

32551 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize  
 32552 another condition variable.

32553 [ENOMEM] Insufficient memory exists to initialize the condition variable.

32554 The `pthread_cond_init()` function may fail if:

32555 [EBUSY] The implementation has detected an attempt to reinitialize the object  
32556 referenced by `cond`, a previously initialized, but not yet destroyed, condition  
32557 variable.

32558 [EINVAL] The value specified by `attr` is invalid.

32559 These functions shall not return an error code of [EINTR].

#### 32560 EXAMPLES

32561 A condition variable can be destroyed immediately after all the threads that are blocked on it are  
32562 awakened. For example, consider the following code:

```

32563 struct list {
32564 pthread_mutex_t lm;
32565 ...
32566 }
32567 struct elt {
32568 key k;
32569 int busy;
32570 pthread_cond_t notbusy;
32571 ...
32572 }
32573 /* Find a list element and reserve it. */
32574 struct elt *
32575 list_find(struct list *lp, key k)
32576 {
32577 struct elt *ep;
32578 pthread_mutex_lock(&lp->lm);
32579 while ((ep = find_elt(l, k) != NULL) && ep->busy)
32580 pthread_cond_wait(&ep->notbusy, &lp->lm);
32581 if (ep != NULL)
32582 ep->busy = 1;
32583 pthread_mutex_unlock(&lp->lm);
32584 return(ep);
32585 }
32586 delete_elt(struct list *lp, struct elt *ep)
32587 {
32588 pthread_mutex_lock(&lp->lm);
32589 assert(ep->busy);
32590 ... remove ep from list ...
32591 ep->busy = 0; /* Paranoid. */
32592 (A) pthread_cond_broadcast(&ep->notbusy);
32593 pthread_mutex_unlock(&lp->lm);
32594 (B) pthread_cond_destroy(&rp->notbusy);
32595 free(ep);
32596 }

```

32597 In this example, the condition variable and its list element may be freed (line B) immediately  
32598 after all threads waiting for it are awakened (line A), since the mutex and the code ensure that no  
32599 other thread can touch the element to be deleted.

## 32600 APPLICATION USAGE

32601 None.

## 32602 RATIONALE

32603 See *pthread\_mutex\_init()*; a similar rationale applies to condition variables.

## 32604 FUTURE DIRECTIONS

32605 None.

## 32606 SEE ALSO

32607 *pthread\_cond\_broadcast()*, *pthread\_cond\_signal()*, *pthread\_cond\_timedwait()*, the Base Definitions  
32608 volume of IEEE Std 1003.1-2001, <pthread.h>

## 32609 CHANGE HISTORY

32610 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

### 32611 Issue 6

32612 The *pthread\_cond\_destroy()* and *pthread\_cond\_init()* functions are marked as part of the Threads  
32613 option.

32614 IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

32615 The **restrict** keyword is added to the *pthread\_cond\_init()* prototype for alignment with the  
32616 ISO/IEC 9899:1999 standard.

32617 **NAME**

32618 pthread\_cond\_signal — signal a condition

32619 **SYNOPSIS**

32620 THR #include &lt;pthread.h&gt;

32621 int pthread\_cond\_signal(pthread\_cond\_t \*cond);

32622

32623 **DESCRIPTION**32624 Refer to *pthread\_cond\_broadcast()*.

## 32625 NAME

32626 pthread\_cond\_timedwait, pthread\_cond\_wait — wait on a condition

## 32627 SYNOPSIS

32628 THR #include &lt;pthread.h&gt;

```

32629 int pthread_cond_timedwait(pthread_cond_t *restrict cond,
32630 pthread_mutex_t *restrict mutex,
32631 const struct timespec *restrict abstime);
32632 int pthread_cond_wait(pthread_cond_t *restrict cond,
32633 pthread_mutex_t *restrict mutex);
32634

```

## 32635 DESCRIPTION

32636 The *pthread\_cond\_timedwait()* and *pthread\_cond\_wait()* functions shall block on a condition  
 32637 variable. They shall be called with *mutex* locked by the calling thread or undefined behavior  
 32638 results.

32639 These functions atomically release *mutex* and cause the calling thread to block on the condition  
 32640 variable *cond*; atomically here means “atomically with respect to access by another thread to the  
 32641 mutex and then the condition variable”. That is, if another thread is able to acquire the mutex  
 32642 after the about-to-block thread has released it, then a subsequent call to *pthread\_cond\_broadcast()*  
 32643 or *pthread\_cond\_signal()* in that thread shall behave as if it were issued after the about-to-block  
 32644 thread has blocked.

32645 Upon successful return, the mutex shall have been locked and shall be owned by the calling  
 32646 thread.

32647 When using condition variables there is always a Boolean predicate involving shared variables  
 32648 associated with each condition wait that is true if the thread should proceed. Spurious wakeups  
 32649 from the *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()* functions may occur. Since the return  
 32650 from *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()* does not imply anything about the value  
 32651 of this predicate, the predicate should be re-evaluated upon such return.

32652 The effect of using more than one mutex for concurrent *pthread\_cond\_timedwait()* or  
 32653 *pthread\_cond\_wait()* operations on the same condition variable is undefined; that is, a condition  
 32654 variable becomes bound to a unique mutex when a thread waits on the condition variable, and  
 32655 this (dynamic) binding shall end when the wait returns.

32656 A condition wait (whether timed or not) is a cancellation point. When the cancelability enable  
 32657 state of a thread is set to PTHREAD\_CANCEL\_DEFERRED, a side effect of acting upon a  
 32658 cancellation request while in a condition wait is that the mutex is (in effect) re-acquired before  
 32659 calling the first cancellation cleanup handler. The effect is as if the thread were unblocked,  
 32660 allowed to execute up to the point of returning from the call to *pthread\_cond\_timedwait()* or  
 32661 *pthread\_cond\_wait()*, but at that point notices the cancellation request and instead of returning to  
 32662 the caller of *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()*, starts the thread cancellation  
 32663 activities, which includes calling cancellation cleanup handlers.

32664 A thread that has been unblocked because it has been canceled while blocked in a call to  
 32665 *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()* shall not consume any condition signal that  
 32666 may be directed concurrently at the condition variable if there are other threads blocked on the  
 32667 condition variable.

32668 The *pthread\_cond\_timedwait()* function shall be equivalent to *pthread\_cond\_wait()*, except that an  
 32669 error is returned if the absolute time specified by *abstime* passes (that is, system time equals or  
 32670 exceeds *abstime*) before the condition *cond* is signaled or broadcasted, or if the absolute time  
 32671 specified by *abstime* has already been passed at the time of the call.

32672 cs If the Clock Selection option is supported, the condition variable shall have a clock attribute  
 32673 which specifies the clock that shall be used to measure the time specified by the *abstime*  
 32674 argument. When such timeouts occur, *pthread\_cond\_timedwait()* shall nonetheless release and  
 32675 re-acquire the mutex referenced by *mutex*. The *pthread\_cond\_timedwait()* function is also a  
 32676 cancellation point.

32677 If a signal is delivered to a thread waiting for a condition variable, upon return from the signal  
 32678 handler the thread resumes waiting for the condition variable as if it was not interrupted, or it  
 32679 shall return zero due to spurious wakeup.

#### 32680 RETURN VALUE

32681 Except in the case of [ETIMEDOUT], all these error checks shall act as if they were performed  
 32682 immediately at the beginning of processing for the function and shall cause an error return, in  
 32683 effect, prior to modifying the state of the mutex specified by *mutex* or the condition variable  
 32684 specified by *cond*.

32685 Upon successful completion, a value of zero shall be returned; otherwise, an error number shall  
 32686 be returned to indicate the error.

#### 32687 ERRORS

32688 The *pthread\_cond\_timedwait()* function shall fail if:

32689 [ETIMEDOUT] The time specified by *abstime* to *pthread\_cond\_timedwait()* has passed.

32690 The *pthread\_cond\_timedwait()* and *pthread\_cond\_wait()* functions may fail if:

32691 [EINVAL] The value specified by *cond*, *mutex*, or *abstime* is invalid.

32692 [EINVAL] Different mutexes were supplied for concurrent *pthread\_cond\_timedwait()* or  
 32693 *pthread\_cond\_wait()* operations on the same condition variable.

32694 [EPERM] The mutex was not owned by the current thread at the time of the call.

32695 These functions shall not return an error code of [EINTR].

#### 32696 EXAMPLES

32697 None.

#### 32698 APPLICATION USAGE

32699 None.

#### 32700 RATIONALE

##### 32701 Condition Wait Semantics

32702 It is important to note that when *pthread\_cond\_wait()* and *pthread\_cond\_timedwait()* return  
 32703 without error, the associated predicate may still be false. Similarly, when  
 32704 *pthread\_cond\_timedwait()* returns with the timeout error, the associated predicate may be true  
 32705 due to an unavoidable race between the expiration of the timeout and the predicate state change.

32706 Some implementations, particularly on a multi-processor, may sometimes cause multiple  
 32707 threads to wake up when the condition variable is signaled simultaneously on different  
 32708 processors.

32709 In general, whenever a condition wait returns, the thread has to re-evaluate the predicate  
 32710 associated with the condition wait to determine whether it can safely proceed, should wait  
 32711 again, or should declare a timeout. A return from the wait does not imply that the associated  
 32712 predicate is either true or false.

32713 It is thus recommended that a condition wait be enclosed in the equivalent of a “while loop”  
 32714 that checks the predicate.

32715 **Timed Wait Semantics**

32716 An absolute time measure was chosen for specifying the timeout parameter for two reasons.  
 32717 First, a relative time measure can be easily implemented on top of a function that specifies  
 32718 absolute time, but there is a race condition associated with specifying an absolute timeout on top  
 32719 of a function that specifies relative timeouts. For example, assume that `clock_gettime()` returns  
 32720 the current time and `cond_relative_timed_wait()` uses relative timeouts:

```
32721 clock_gettime(CLOCK_REALTIME, &now)
32722 reltime = sleep_til_this_absolute_time - now;
32723 cond_relative_timed_wait(c, m, &reltime);
```

32724 If the thread is preempted between the first statement and the last statement, the thread blocks  
 32725 for too long. Blocking, however, is irrelevant if an absolute timeout is used. An absolute timeout  
 32726 also need not be recomputed if it is used multiple times in a loop, such as that enclosing a  
 32727 condition wait.

32728 For cases when the system clock is advanced discontinuously by an operator, it is expected that  
 32729 implementations process any timed wait expiring at an intervening time as if that time had  
 32730 actually occurred.

32731 **Cancellation and Condition Wait**

32732 A condition wait, whether timed or not, is a cancellation point. That is, the functions  
 32733 `pthread_cond_wait()` or `pthread_cond_timedwait()` are points where a pending (or concurrent)  
 32734 cancellation request is noticed. The reason for this is that an indefinite wait is possible at these  
 32735 points—whatever event is being waited for, even if the program is totally correct, might never  
 32736 occur; for example, some input data being awaited might never be sent. By making condition  
 32737 wait a cancellation point, the thread can be canceled and perform its cancellation cleanup  
 32738 handler even though it may be stuck in some indefinite wait.

32739 A side effect of acting on a cancellation request while a thread is blocked on a condition variable  
 32740 is to re-acquire the mutex before calling any of the cancellation cleanup handlers. This is done in  
 32741 order to ensure that the cancellation cleanup handler is executed in the same state as the critical  
 32742 code that lies both before and after the call to the condition wait function. This rule is also  
 32743 required when interfacing to POSIX threads from languages, such as Ada or C++, which may  
 32744 choose to map cancellation onto a language exception; this rule ensures that each exception  
 32745 handler guarding a critical section can always safely depend upon the fact that the associated  
 32746 mutex has already been locked regardless of exactly where within the critical section the  
 32747 exception was raised. Without this rule, there would not be a uniform rule that exception  
 32748 handlers could follow regarding the lock, and so coding would become very cumbersome.

32749 Therefore, since *some* statement has to be made regarding the state of the lock when a  
 32750 cancellation is delivered during a wait, a definition has been chosen that makes application  
 32751 coding most convenient and error free.

32752 When acting on a cancellation request while a thread is blocked on a condition variable, the  
 32753 implementation is required to ensure that the thread does not consume any condition signals  
 32754 directed at that condition variable if there are any other threads waiting on that condition  
 32755 variable. This rule is specified in order to avoid deadlock conditions that could occur if these two  
 32756 independent requests (one acting on a thread and the other acting on the condition variable)  
 32757 were not processed independently.

32758 **Performance of Mutexes and Condition Variables**

32759 Mutexes are expected to be locked only for a few instructions. This practice is almost  
 32760 automatically enforced by the desire of programmers to avoid long serial regions of execution  
 32761 (which would reduce total effective parallelism).

32762 When using mutexes and condition variables, one tries to ensure that the usual case is to lock the  
 32763 mutex, access shared data, and unlock the mutex. Waiting on a condition variable should be a  
 32764 relatively rare situation. For example, when implementing a read-write lock, code that acquires a  
 32765 read-lock typically needs only to increment the count of readers (under mutual-exclusion) and  
 32766 return. The calling thread would actually wait on the condition variable only when there is  
 32767 already an active writer. So the efficiency of a synchronization operation is bounded by the cost  
 32768 of mutex lock/unlock and not by condition wait. Note that in the usual case there is no context  
 32769 switch.

32770 This is not to say that the efficiency of condition waiting is unimportant. Since there needs to be  
 32771 at least one context switch per Ada rendezvous, the efficiency of waiting on a condition variable  
 32772 is important. The cost of waiting on a condition variable should be little more than the minimal  
 32773 cost for a context switch plus the time to unlock and lock the mutex.

32774 **Features of Mutexes and Condition Variables**

32775 It had been suggested that the mutex acquisition and release be decoupled from condition wait.  
 32776 This was rejected because it is the combined nature of the operation that, in fact, facilitates  
 32777 realtime implementations. Those implementations can atomically move a high-priority thread  
 32778 between the condition variable and the mutex in a manner that is transparent to the caller. This  
 32779 can prevent extra context switches and provide more deterministic acquisition of a mutex when  
 32780 the waiting thread is signaled. Thus, fairness and priority issues can be dealt with directly by the  
 32781 scheduling discipline. Furthermore, the current condition wait operation matches existing  
 32782 practice.

32783 **Scheduling Behavior of Mutexes and Condition Variables**

32784 Synchronization primitives that attempt to interfere with scheduling policy by specifying an  
 32785 ordering rule are considered undesirable. Threads waiting on mutexes and condition variables  
 32786 are selected to proceed in an order dependent upon the scheduling policy rather than in some  
 32787 fixed order (for example, FIFO or priority). Thus, the scheduling policy determines which  
 32788 thread(s) are awakened and allowed to proceed.

32789 **Timed Condition Wait**

32790 The *pthread\_cond\_timedwait()* function allows an application to give up waiting for a particular  
 32791 condition after a given amount of time. An example of its use follows:

```
32792 (void) pthread_mutex_lock(&t.mn);
32793 t.waiters++;
32794 clock_gettime(CLOCK_REALTIME, &ts);
32795 ts.tv_sec += 5;
32796 rc = 0;
32797 while (! mypredicate(&t) && rc == 0)
32798 rc = pthread_cond_timedwait(&t.cond, &t.mn, &ts);
32799 t.waiters--;
32800 if (rc == 0) setmystate(&t);
32801 (void) pthread_mutex_unlock(&t.mn);
```

32802 By making the timeout parameter absolute, it does not need to be recomputed each time the  
32803 program checks its blocking predicate. If the timeout was relative, it would have to be  
32804 recomputed before each call. This would be especially difficult since such code would need to  
32805 take into account the possibility of extra wakeups that result from extra broadcasts or signals on  
32806 the condition variable that occur before either the predicate is true or the timeout is due.

## 32807 FUTURE DIRECTIONS

32808 None.

## 32809 SEE ALSO

32810 *pthread\_cond\_signal()*, *pthread\_cond\_broadcast()*, the Base Definitions volume of  
32811 IEEE Std 1003.1-2001, <pthread.h>

## 32812 CHANGE HISTORY

32813 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

## 32814 Issue 6

32815 The *pthread\_cond\_timedwait()* and *pthread\_cond\_wait()* functions are marked as part of the  
32816 Threads option.

32817 The Open Group Corrigendum U021/9 is applied, correcting the prototype for the  
32818 *pthread\_cond\_wait()* function.

32819 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for  
32820 the Clock Selection option.

32821 The ERRORS section has an additional case for [EPERM] in response to IEEE PASC  
32822 Interpretation 1003.1c #28.

32823 The **restrict** keyword is added to the *pthread\_cond\_timedwait()* and *pthread\_cond\_wait()*  
32824 prototypes for alignment with the ISO/IEC 9899:1999 standard.

32825 **NAME**

32826 pthread\_condattr\_destroy, pthread\_condattr\_init — destroy and initialize the condition variable  
 32827 attributes object

32828 **SYNOPSIS**

32829 THR #include <pthread.h>

32830 int pthread\_condattr\_destroy(pthread\_condattr\_t \*attr);

32831 int pthread\_condattr\_init(pthread\_condattr\_t \*attr);

32832

32833 **DESCRIPTION**

32834 The *pthread\_condattr\_destroy()* function shall destroy a condition variable attributes object; the  
 32835 object becomes, in effect, uninitialized. An implementation may cause *pthread\_condattr\_destroy()*  
 32836 to set the object referenced by *attr* to an invalid value. A destroyed *attr* attributes object can be  
 32837 reinitialized using *pthread\_condattr\_init()*; the results of otherwise referencing the object after it  
 32838 has been destroyed are undefined.

32839 The *pthread\_condattr\_init()* function shall initialize a condition variable attributes object *attr* with  
 32840 the default value for all of the attributes defined by the implementation.

32841 Results are undefined if *pthread\_condattr\_init()* is called specifying an already initialized *attr*  
 32842 attributes object.

32843 After a condition variable attributes object has been used to initialize one or more condition  
 32844 variables, any function affecting the attributes object (including destruction) shall not affect any  
 32845 previously initialized condition variables.

32846 This volume of IEEE Std 1003.1-2001 requires two attributes, the *clock* attribute and the *process-*  
 32847 *shared* attribute.

32848 Additional attributes, their default values, and the names of the associated functions to get and  
 32849 set those attribute values are implementation-defined.

32850 **RETURN VALUE**

32851 If successful, the *pthread\_condattr\_destroy()* and *pthread\_condattr\_init()* functions shall return  
 32852 zero; otherwise, an error number shall be returned to indicate the error.

32853 **ERRORS**

32854 The *pthread\_condattr\_destroy()* function may fail if:

32855 [EINVAL] The value specified by *attr* is invalid.

32856 The *pthread\_condattr\_init()* function shall fail if:

32857 [ENOMEM] Insufficient memory exists to initialize the condition variable attributes object.

32858 These functions shall not return an error code of [EINTR].

32859 **EXAMPLES**

32860 None.

32861 **APPLICATION USAGE**

32862 None.

32863 **RATIONALE**

32864 See *pthread\_attr\_init()* and *pthread\_mutex\_init()*.

32865 A *process-shared* attribute has been defined for condition variables for the same reason it has been  
 32866 defined for mutexes.

32867 **FUTURE DIRECTIONS**

32868       None.

32869 **SEE ALSO**

32870       *pthread\_attr\_destroy()*, *pthread\_cond\_destroy()*, *pthread\_condattr\_getpshared()*, *pthread\_create()*,  
32871       *pthread\_mutex\_destroy()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>

32872 **CHANGE HISTORY**

32873       First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32874 **Issue 6**

32875       The *pthread\_condattr\_destroy()* and *pthread\_condattr\_init()* functions are marked as part of the  
32876       Threads option.

32877 **NAME**

32878 pthread\_condattr\_getclock, pthread\_condattr\_setclock — get and set the clock selection  
 32879 condition variable attribute (**ADVANCED REALTIME**)

32880 **SYNOPSIS**

32881 THR CS #include <pthread.h>

```
32882 int pthread_condattr_getclock(const pthread_condattr_t *restrict attr,
32883 clockid_t *restrict clock_id);
32884 int pthread_condattr_setclock(pthread_condattr_t *attr,
32885 clockid_t clock_id);
32886
```

32887 **DESCRIPTION**

32888 The *pthread\_condattr\_getclock()* function shall obtain the value of the *clock* attribute from the  
 32889 attributes object referenced by *attr*. The *pthread\_condattr\_setclock()* function shall set the *clock*  
 32890 attribute in an initialized attributes object referenced by *attr*. If *pthread\_condattr\_setclock()* is  
 32891 called with a *clock\_id* argument that refers to a CPU-time clock, the call shall fail.

32892 The *clock* attribute is the clock ID of the clock that shall be used to measure the timeout service of  
 32893 *pthread\_cond\_timedwait()*. The default value of the *clock* attribute shall refer to the system clock.

32894 **RETURN VALUE**

32895 If successful, the *pthread\_condattr\_getclock()* function shall return zero and store the value of the  
 32896 clock attribute of *attr* into the object referenced by the *clock\_id* argument. Otherwise, an error  
 32897 number shall be returned to indicate the error.

32898 If successful, the *pthread\_condattr\_setclock()* function shall return zero; otherwise, an error  
 32899 number shall be returned to indicate the error.

32900 **ERRORS**

32901 These functions may fail if:

32902 [EINVAL] The value specified by *attr* is invalid.

32903 The *pthread\_condattr\_setclock()* function may fail if:

32904 [EINVAL] The value specified by *clock\_id* does not refer to a known clock, or is a CPU-  
 32905 time clock.

32906 These functions shall not return an error code of [EINTR].

32907 **EXAMPLES**

32908 None.

32909 **APPLICATION USAGE**

32910 None.

32911 **RATIONALE**

32912 None.

32913 **FUTURE DIRECTIONS**

32914 None.

32915 **SEE ALSO**

32916 *pthread\_cond\_destroy()*, *pthread\_cond\_timedwait()*, *pthread\_condattr\_destroy()*,  
 32917 *pthread\_condattr\_getshared()* (on page 1045), *pthread\_condattr\_init()*,  
 32918 *pthread\_condattr\_setshared()* (on page 1049), *pthread\_create()*, *pthread\_mutex\_init()*, the Base  
 32919 Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

32920 **CHANGE HISTORY**

32921 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

32922 **NAME**

32923 pthread\_condattr\_getpshared, pthread\_condattr\_setpshared — get and set the process-shared  
 32924 condition variable attributes

32925 **SYNOPSIS**

```
32926 THR TSH #include <pthread.h>
32927
32927 int pthread_condattr_getpshared(const pthread_condattr_t *restrict attr,
32928 int *restrict pshared);
32929 int pthread_condattr_setpshared(pthread_condattr_t *attr,
32930 int pshared);
32931
```

32932 **DESCRIPTION**

32933 The *pthread\_condattr\_getpshared()* function shall obtain the value of the *process-shared* attribute  
 32934 from the attributes object referenced by *attr*. The *pthread\_condattr\_setpshared()* function shall set  
 32935 the *process-shared* attribute in an initialized attributes object referenced by *attr*.

32936 The *process-shared* attribute is set to PTHREAD\_PROCESS\_SHARED to permit a condition  
 32937 variable to be operated upon by any thread that has access to the memory where the condition  
 32938 variable is allocated, even if the condition variable is allocated in memory that is shared by  
 32939 multiple processes. If the *process-shared* attribute is PTHREAD\_PROCESS\_PRIVATE, the  
 32940 condition variable shall only be operated upon by threads created within the same process as the  
 32941 thread that initialized the condition variable; if threads of differing processes attempt to operate  
 32942 on such a condition variable, the behavior is undefined. The default value of the attribute is  
 32943 PTHREAD\_PROCESS\_PRIVATE.

32944 **RETURN VALUE**

32945 If successful, the *pthread\_condattr\_setpshared()* function shall return zero; otherwise, an error  
 32946 number shall be returned to indicate the error.

32947 If successful, the *pthread\_condattr\_getpshared()* function shall return zero and store the value of  
 32948 the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise,  
 32949 an error number shall be returned to indicate the error.

32950 **ERRORS**

32951 The *pthread\_condattr\_getpshared()* and *pthread\_condattr\_setpshared()* functions may fail if:

32952 [EINVAL] The value specified by *attr* is invalid.

32953 The *pthread\_condattr\_setpshared()* function may fail if:

32954 [EINVAL] The new value specified for the attribute is outside the range of legal values  
 32955 for that attribute.

32956 These functions shall not return an error code of [EINTR].

32957 **EXAMPLES**

32958 None.

32959 **APPLICATION USAGE**

32960 None.

32961 **RATIONALE**

32962 None.

## 32963 FUTURE DIRECTIONS

32964 None.

## 32965 SEE ALSO

32966 *pthread\_create()*, *pthread\_cond\_destroy()*, *pthread\_condattr\_destroy()*, *pthread\_mutex\_destroy()*, the  
32967 Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

## 32968 CHANGE HISTORY

32969 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

## 32970 Issue 6

32971 The *pthread\_condattr\_getpshared()* and *pthread\_condattr\_setpshared()* functions are marked as part  
32972 of the Threads and Thread Process-Shared Synchronization options.

32973 The **restrict** keyword is added to the *pthread\_condattr\_getpshared()* prototype for alignment with  
32974 the ISO/IEC 9899:1999 standard.

32975 **NAME**

32976 pthread\_condattr\_init — initialize the condition variable attributes object

32977 **SYNOPSIS**

32978 THR #include <pthread.h>

32979 int pthread\_condattr\_init(pthread\_condattr\_t \*attr);

32980

32981 **DESCRIPTION**

32982 Refer to *pthread\_condattr\_destroy()*.

32983 **NAME**

32984 pthread\_condattr\_setclock — set the clock selection condition variable attribute

32985 **SYNOPSIS**

32986 THR CS #include <pthread.h>

```
32987 int pthread_condattr_setclock(pthread_condattr_t *attr,
32988 clockid_t clock_id);
```

32989

32990 **DESCRIPTION**

32991 Refer to *pthread\_condattr\_getclock()*.

32992 **NAME**

32993 pthread\_condattr\_setpshared — set the process-shared condition variable attribute

32994 **SYNOPSIS**

32995 THR TSH #include &lt;pthread.h&gt;

32996 int pthread\_condattr\_setpshared(pthread\_condattr\_t \*attr,  
32997 int pshared);

32998

32999 **DESCRIPTION**33000 Refer to *pthread\_condattr\_getpshared()*.

## 33001 NAME

33002 pthread\_create — thread creation

## 33003 SYNOPSIS

33004 THR #include &lt;pthread.h&gt;

```
33005 int pthread_create(pthread_t *restrict thread,
33006 const pthread_attr_t *restrict attr,
33007 void *(*start_routine)(void*), void *restrict arg);
33008
```

## 33009 DESCRIPTION

33010 The *pthread\_create()* function shall create a new thread, with attributes specified by *attr*, within a  
 33011 process. If *attr* is NULL, the default attributes shall be used. If the attributes specified by *attr* are  
 33012 modified later, the thread's attributes shall not be affected. Upon successful completion,  
 33013 *pthread\_create()* shall store the ID of the created thread in the location referenced by *thread*.

33014 The thread is created executing *start\_routine* with *arg* as its sole argument. If the *start\_routine*  
 33015 returns, the effect shall be as if there was an implicit call to *pthread\_exit()* using the return value  
 33016 of *start\_routine* as the exit status. Note that the thread in which *main()* was originally invoked  
 33017 differs from this. When it returns from *main()*, the effect shall be as if there was an implicit call  
 33018 to *exit()* using the return value of *main()* as the exit status.

33019 The signal state of the new thread shall be initialized as follows:

- 33020 • The signal mask shall be inherited from the creating thread.
- 33021 • The set of signals pending for the new thread shall be empty.

33022 XSI The alternate stack shall not be inherited. |

33023 The floating-point environment shall be inherited from the creating thread. |

33024 If *pthread\_create()* fails, no new thread is created and the contents of the location referenced by  
 33025 *thread* are undefined.

33026 TCT If `_POSIX_THREAD_CPUTIME` is defined, the new thread shall have a CPU-time clock  
 33027 accessible, and the initial value of this clock shall be set to zero.

## 33028 RETURN VALUE

33029 If successful, the *pthread\_create()* function shall return zero; otherwise, an error number shall be  
 33030 returned to indicate the error.

## 33031 ERRORS

33032 The *pthread\_create()* function shall fail if:

33033 [EAGAIN] The system lacked the necessary resources to create another thread, or the  
 33034 system-imposed limit on the total number of threads in a process  
 33035 {PTHREAD\_THREADS\_MAX} would be exceeded.

33036 [EINVAL] The value specified by *attr* is invalid.

33037 [EPERM] The caller does not have appropriate permission to set the required  
 33038 scheduling parameters or scheduling policy.

33039 The *pthread\_create()* function shall not return an error code of [EINTR].

33040 **EXAMPLES**

33041 None.

33042 **APPLICATION USAGE**

33043 None.

33044 **RATIONALE**

33045 A suggested alternative to *pthread\_create()* would be to define two separate operations: create  
 33046 and start. Some applications would find such behavior more natural. Ada, in particular,  
 33047 separates the “creation” of a task from its “activation”.

33048 Splitting the operation was rejected by the standard developers for many reasons:

- 33049 • The number of calls required to start a thread would increase from one to two and thus place  
 33050 an additional burden on applications that do not require the additional synchronization. The  
 33051 second call, however, could be avoided by the additional complication of a start-up state  
 33052 attribute.
- 33053 • An extra state would be introduced: “created but not started”. This would require the  
 33054 standard to specify the behavior of the thread operations when the target has not yet started  
 33055 executing.
- 33056 • For those applications that require such behavior, it is possible to simulate the two separate  
 33057 steps with the facilities that are currently provided. The *start\_routine()* can synchronize by  
 33058 waiting on a condition variable that is signaled by the start operation.

33059 An Ada implementor can choose to create the thread at either of two points in the Ada program:  
 33060 when the task object is created, or when the task is activated (generally at a “begin”). If the first  
 33061 approach is adopted, the *start\_routine()* needs to wait on a condition variable to receive the  
 33062 order to begin “activation”. The second approach requires no such condition variable or extra  
 33063 synchronization. In either approach, a separate Ada task control block would need to be created  
 33064 when the task object is created to hold rendezvous queues, and so on.

33065 An extension of the preceding model would be to allow the state of the thread to be modified  
 33066 between the create and start. This would allow the thread attributes object to be eliminated. This  
 33067 has been rejected because:

- 33068 • All state in the thread attributes object has to be able to be set for the thread. This would  
 33069 require the definition of functions to modify thread attributes. There would be no reduction  
 33070 in the number of function calls required to set up the thread. In fact, for an application that  
 33071 creates all threads using identical attributes, the number of function calls required to set up  
 33072 the threads would be dramatically increased. Use of a thread attributes object permits the  
 33073 application to make one set of attribute setting function calls. Otherwise, the set of attribute  
 33074 setting function calls needs to be made for each thread creation.
- 33075 • Depending on the implementation architecture, functions to set thread state would require  
 33076 kernel calls, or for other implementation reasons would not be able to be implemented as  
 33077 macros, thereby increasing the cost of thread creation.
- 33078 • The ability for applications to segregate threads by class would be lost.

33079 Another suggested alternative uses a model similar to that for process creation, such as “thread  
 33080 fork”. The fork semantics would provide more flexibility and the “create” function can be  
 33081 implemented simply by doing a thread fork followed immediately by a call to the desired “start  
 33082 routine” for the thread. This alternative has these problems:

- 33083 • For many implementations, the entire stack of the calling thread would need to be  
 33084 duplicated, since in many architectures there is no way to determine the size of the calling  
 33085 frame.

33086           • Efficiency is reduced since at least some part of the stack has to be copied, even though in  
33087           most cases the thread never needs the copied context, since it merely calls the desired start  
33088           routine.

33089 **FUTURE DIRECTIONS**

33090           None.

33091 **SEE ALSO**

33092           *fork()*, *pthread\_exit()*, *pthread\_join()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
33093           <pthread.h>

33094 **CHANGE HISTORY**

33095           First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33096 **Issue 6**

33097           The *pthread\_create()* function is marked as part of the Threads option.

33098           The following new requirements on POSIX implementations derive from alignment with the  
33099           Single UNIX Specification:

33100           • The [EPERM] mandatory error condition is added.

33101           The thread CPU-time clock semantics are added for alignment with IEEE Std 1003.1d-1999.

33102           The **restrict** keyword is added to the *pthread\_create()* prototype for alignment with the  
33103           ISO/IEC 9899:1999 standard.

33104           The DESCRIPTION is updated to make it explicit that the floating-point environment is  
33105           inherited from the creating thread.

33106           IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/44 is applied, adding text that the |  
33107           alternate stack is not inherited. |

33108 **NAME**

33109 pthread\_detach — detach a thread

33110 **SYNOPSIS**

33111 THR #include &lt;pthread.h&gt;

33112 int pthread\_detach(pthread\_t thread);

33113

33114 **DESCRIPTION**

33115 The *pthread\_detach()* function shall indicate to the implementation that storage for the thread  
 33116 *thread* can be reclaimed when that thread terminates. If *thread* has not terminated,  
 33117 *pthread\_detach()* shall not cause it to terminate. The effect of multiple *pthread\_detach()* calls on  
 33118 the same target thread is unspecified.

33119 **RETURN VALUE**

33120 If the call succeeds, *pthread\_detach()* shall return 0; otherwise, an error number shall be returned  
 33121 to indicate the error.

33122 **ERRORS**33123 The *pthread\_detach()* function shall fail if:

33124 [EINVAL] The implementation has detected that the value specified by *thread* does not  
 33125 refer to a joinable thread.

33126 [ESRCH] No thread could be found corresponding to that specified by the given thread  
 33127 ID.

33128 The *pthread\_detach()* function shall not return an error code of [EINTR].33129 **EXAMPLES**

33130 None.

33131 **APPLICATION USAGE**

33132 None.

33133 **RATIONALE**

33134 The *pthread\_join()* or *pthread\_detach()* functions should eventually be called for every thread that  
 33135 is created so that storage associated with the thread may be reclaimed.

33136 It has been suggested that a “detach” function is not necessary; the *detachstate* thread creation  
 33137 attribute is sufficient, since a thread need never be dynamically detached. However, need arises  
 33138 in at least two cases:

33139 1. In a cancellation handler for a *pthread\_join()* it is nearly essential to have a *pthread\_detach()*  
 33140 function in order to detach the thread on which *pthread\_join()* was waiting. Without it, it  
 33141 would be necessary to have the handler do another *pthread\_join()* to attempt to detach the  
 33142 thread, which would both delay the cancellation processing for an unbounded period and  
 33143 introduce a new call to *pthread\_join()*, which might itself need a cancellation handler. A  
 33144 dynamic detach is nearly essential in this case.

33145 2. In order to detach the “initial thread” (as may be desirable in processes that set up server  
 33146 threads).

33147 **FUTURE DIRECTIONS**

33148 None.

33149 **SEE ALSO**

33150 *pthread\_join()*, the Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

33151 **CHANGE HISTORY**

33152 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33153 **Issue 6**

33154 The *pthread\_detach()* function is marked as part of the Threads option.

33155 **NAME**

33156 pthread\_equal — compare thread IDs

33157 **SYNOPSIS**

33158 THR #include &lt;pthread.h&gt;

33159 int pthread\_equal(pthread\_t t1, pthread\_t t2);

33160

33161 **DESCRIPTION**33162 This function shall compare the thread IDs *t1* and *t2*.33163 **RETURN VALUE**33164 The *pthread\_equal()* function shall return a non-zero value if *t1* and *t2* are equal; otherwise, zero shall be returned.33166 If either *t1* or *t2* are not valid thread IDs, the behavior is undefined.33167 **ERRORS**

33168 No errors are defined.

33169 The *pthread\_equal()* function shall not return an error code of [EINTR].33170 **EXAMPLES**

33171 None.

33172 **APPLICATION USAGE**

33173 None.

33174 **RATIONALE**33175 Implementations may choose to define a thread ID as a structure. This allows additional flexibility and robustness over using an **int**. For example, a thread ID could include a sequence number that allows detection of “dangling IDs” (copies of a thread ID that has been detached).  
33176 Since the C language does not support comparison on structure types, the *pthread\_equal()*  
33177 function is provided to compare thread IDs.  
33178  
3317933180 **FUTURE DIRECTIONS**

33181 None.

33182 **SEE ALSO**33183 *pthread\_create()*, *pthread\_self()*, the Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>33184 **CHANGE HISTORY**

33185 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33186 **Issue 6**33187 The *pthread\_equal()* function is marked as part of the Threads option.

33188 **NAME**

33189 pthread\_exit — thread termination

33190 **SYNOPSIS**

33191 THR #include &lt;pthread.h&gt;

33192 void pthread\_exit(void \*value\_ptr);

33193

33194 **DESCRIPTION**

33195 The *pthread\_exit()* function shall terminate the calling thread and make the value *value\_ptr*  
33196 available to any successful join with the terminating thread. Any cancellation cleanup handlers  
33197 that have been pushed and not yet popped shall be popped in the reverse order that they were  
33198 pushed and then executed. After all cancellation cleanup handlers have been executed, if the  
33199 thread has any thread-specific data, appropriate destructor functions shall be called in an  
33200 unspecified order. Thread termination does not release any application visible process resources,  
33201 including, but not limited to, mutexes and file descriptors, nor does it perform any process-level  
33202 cleanup actions, including, but not limited to, calling any *atexit()* routines that may exist.

33203 An implicit call to *pthread\_exit()* is made when a thread other than the thread in which *main()*  
33204 was first invoked returns from the start routine that was used to create it. The function's return  
33205 value shall serve as the thread's exit status.

33206 The behavior of *pthread\_exit()* is undefined if called from a cancellation cleanup handler or  
33207 destructor function that was invoked as a result of either an implicit or explicit call to  
33208 *pthread\_exit()*.

33209 After a thread has terminated, the result of access to local (auto) variables of the thread is  
33210 undefined. Thus, references to local variables of the exiting thread should not be used for the  
33211 *pthread\_exit()* *value\_ptr* parameter value.

33212 The process shall exit with an exit status of 0 after the last thread has been terminated. The  
33213 behavior shall be as if the implementation called *exit()* with a zero argument at thread  
33214 termination time.

33215 **RETURN VALUE**33216 The *pthread\_exit()* function cannot return to its caller.33217 **ERRORS**

33218 No errors are defined.

33219 **EXAMPLES**

33220 None.

33221 **APPLICATION USAGE**

33222 None.

33223 **RATIONALE**

33224 The normal mechanism by which a thread terminates is to return from the routine that was  
33225 specified in the *pthread\_create()* call that started it. The *pthread\_exit()* function provides the  
33226 capability for a thread to terminate without requiring a return from the start routine of that  
33227 thread, thereby providing a function analogous to *exit()*.

33228 Regardless of the method of thread termination, any cancellation cleanup handlers that have  
33229 been pushed and not yet popped are executed, and the destructors for any existing thread-  
33230 specific data are executed. This volume of IEEE Std 1003.1-2001 requires that cancellation  
33231 cleanup handlers be popped and called in order. After all cancellation cleanup handlers have  
33232 been executed, thread-specific data destructors are called, in an unspecified order, for each item  
33233 of thread-specific data that exists in the thread. This ordering is necessary because cancellation

- 33234 cleanup handlers may rely on thread-specific data.
- 33235 As the meaning of the status is determined by the application (except when the thread has been  
33236 canceled, in which case it is PTHREAD\_CANCELED), the implementation has no idea what an  
33237 illegal status value is, which is why no address error checking is done.
- 33238 **FUTURE DIRECTIONS**
- 33239 None.
- 33240 **SEE ALSO**
- 33241 *exit()*, *pthread\_create()*, *pthread\_join()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
33242 <pthread.h>
- 33243 **CHANGE HISTORY**
- 33244 First released in Issue 5. Included for alignment with the POSIX Threads Extension.
- 33245 **Issue 6**
- 33246 The *pthread\_exit()* function is marked as part of the Threads option.

33247 **NAME**

33248 pthread\_getconcurrency, pthread\_setconcurrency — get and set the level of concurrency

33249 **SYNOPSIS**

33250 XSI #include &lt;pthread.h&gt;

33251 int pthread\_getconcurrency(void);

33252 int pthread\_setconcurrency(int new\_level);

33253

33254 **DESCRIPTION**

33255 Unbound threads in a process may or may not be required to be simultaneously active. By  
 33256 default, the threads implementation ensures that a sufficient number of threads are active so that  
 33257 the process can continue to make progress. While this conserves system resources, it may not  
 33258 produce the most effective level of concurrency.

33259 The *pthread\_setconcurrency()* function allows an application to inform the threads  
 33260 implementation of its desired concurrency level, *new\_level*. The actual level of concurrency  
 33261 provided by the implementation as a result of this function call is unspecified.

33262 If *new\_level* is zero, it causes the implementation to maintain the concurrency level at its  
 33263 discretion as if *pthread\_setconcurrency()* had never been called.

33264 The *pthread\_getconcurrency()* function shall return the value set by a previous call to the  
 33265 *pthread\_setconcurrency()* function. If the *pthread\_setconcurrency()* function was not previously  
 33266 called, this function shall return zero to indicate that the implementation is maintaining the  
 33267 concurrency level.

33268 A call to *pthread\_setconcurrency()* shall inform the implementation of its desired concurrency  
 33269 level. The implementation shall use this as a hint, not a requirement.

33270 If an implementation does not support multiplexing of user threads on top of several kernel-  
 33271 scheduled entities, the *pthread\_setconcurrency()* and *pthread\_getconcurrency()* functions are  
 33272 provided for source code compatibility but they shall have no effect when called. To maintain  
 33273 the function semantics, the *new\_level* parameter is saved when *pthread\_setconcurrency()* is called  
 33274 so that a subsequent call to *pthread\_getconcurrency()* shall return the same value.

33275 **RETURN VALUE**

33276 If successful, the *pthread\_setconcurrency()* function shall return zero; otherwise, an error number  
 33277 shall be returned to indicate the error.

33278 The *pthread\_getconcurrency()* function shall always return the concurrency level set by a previous  
 33279 call to *pthread\_setconcurrency()*. If the *pthread\_setconcurrency()* function has never been called,  
 33280 *pthread\_getconcurrency()* shall return zero.

33281 **ERRORS**33282 The *pthread\_setconcurrency()* function shall fail if:33283 [EINVAL] The value specified by *new\_level* is negative.33284 [EAGAIN] The value specific by *new\_level* would cause a system resource to be exceeded.

33285 These functions shall not return an error code of [EINTR].

33286 **EXAMPLES**

33287 None.

33288 **APPLICATION USAGE**

33289 Use of these functions changes the state of the underlying concurrency upon which the  
33290 application depends. Library developers are advised to not use the *pthread\_getconcurrency()* and  
33291 *pthread\_setconcurrency()* functions since their use may conflict with an applications use of these  
33292 functions.

33293 **RATIONALE**

33294 None.

33295 **FUTURE DIRECTIONS**

33296 None.

33297 **SEE ALSO**

33298 The Base Definitions volume of IEEE Std 1003.1-2001, &lt;pthread.h&gt;

33299 **CHANGE HISTORY**

33300 First released in Issue 5.

## 33301 NAME

33302 pthread\_getcpuclockid — access a thread CPU-time clock (ADVANCED REALTIME  
33303 THREADS)

## 33304 SYNOPSIS

```
33305 THR TCT #include <pthread.h>
```

```
33306 #include <time.h>
```

```
33307 int pthread_getcpuclockid(pthread_t thread_id, clockid_t *clock_id);
```

33308

## 33309 DESCRIPTION

33310 The *pthread\_getcpuclockid()* function shall return in *clock\_id* the clock ID of the CPU-time clock of  
33311 the thread specified by *thread\_id*, if the thread specified by *thread\_id* exists.

## 33312 RETURN VALUE

33313 Upon successful completion, *pthread\_getcpuclockid()* shall return zero; otherwise, an error  
33314 number shall be returned to indicate the error.

## 33315 ERRORS

33316 The *pthread\_getcpuclockid()* function may fail if:

33317 [ESRCH] The value specified by *thread\_id* does not refer to an existing thread.

## 33318 EXAMPLES

33319 None.

## 33320 APPLICATION USAGE

33321 The *pthread\_getcpuclockid()* function is part of the Thread CPU-Time Clocks option and need not  
33322 be provided on all implementations.

## 33323 RATIONALE

33324 None.

## 33325 FUTURE DIRECTIONS

33326 None.

## 33327 SEE ALSO

33328 *clock\_getcpuclockid()*, *clock\_getres()*, *timer\_create()*, the Base Definitions volume of  
33329 IEEE Std 1003.1-2001, <pthread.h>, <time.h>

## 33330 CHANGE HISTORY

33331 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

33332 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

## 33333 NAME

33334 pthread\_getschedparam, pthread\_setschedparam — dynamic thread scheduling parameters  
 33335 access (**REALTIME THREADS**)

## 33336 SYNOPSIS

33337 THR TPS #include <pthread.h>

```
33338 int pthread_getschedparam(pthread_t thread, int *restrict policy,
33339 struct sched_param *restrict param);
```

```
33340 int pthread_setschedparam(pthread_t thread, int policy,
33341 const struct sched_param *param);
```

33342

## 33343 DESCRIPTION

33344 The *pthread\_getschedparam()* and *pthread\_setschedparam()* functions shall, respectively, get and set  
 33345 the scheduling policy and parameters of individual threads within a multi-threaded process to  
 33346 be retrieved and set. For SCHED\_FIFO and SCHED\_RR, the only required member of the  
 33347 **sched\_param** structure is the priority *sched\_priority*. For SCHED\_OTHER, the affected  
 33348 scheduling parameters are implementation-defined.

33349 The *pthread\_getschedparam()* function shall retrieve the scheduling policy and scheduling  
 33350 parameters for the thread whose thread ID is given by *thread* and shall store those values in  
 33351 *policy* and *param*, respectively. The priority value returned from *pthread\_getschedparam()* shall be  
 33352 the value specified by the most recent *pthread\_setschedparam()*, *pthread\_setschedprio()*, or  
 33353 *pthread\_create()* call affecting the target thread. It shall not reflect any temporary adjustments to  
 33354 its priority as a result of any priority inheritance or ceiling functions. The *pthread\_setschedparam()*  
 33355 function shall set the scheduling policy and associated scheduling parameters for the thread  
 33356 whose thread ID is given by *thread* to the policy and associated parameters provided in *policy*  
 33357 and *param*, respectively.

33358 The *policy* parameter may have the value SCHED\_OTHER, SCHED\_FIFO, or SCHED\_RR. The  
 33359 scheduling parameters for the SCHED\_OTHER policy are implementation-defined. The  
 33360 SCHED\_FIFO and SCHED\_RR policies shall have a single scheduling parameter, *priority*.

33361 TSP If **\_POSIX\_THREAD\_SPORADIC\_SERVER** is defined, then the *policy* argument may have the  
 33362 value SCHED\_SPORADIC, with the exception for the *pthread\_setschedparam()* function that if the  
 33363 scheduling policy was not SCHED\_SPORADIC at the time of the call, it is implementation-  
 33364 defined whether the function is supported; in other words, the implementation need not allow  
 33365 the application to dynamically change the scheduling policy to SCHED\_SPORADIC. The  
 33366 sporadic server scheduling policy has the associated parameters *sched\_ss\_low\_priority*,  
 33367 *sched\_ss\_repl\_period*, *sched\_ss\_init\_budget*, *sched\_priority*, and *sched\_ss\_max\_repl*. The specified  
 33368 *sched\_ss\_repl\_period* shall be greater than or equal to the specified *sched\_ss\_init\_budget* for the  
 33369 function to succeed; if it is not, then the function shall fail. The value of *sched\_ss\_max\_repl* shall  
 33370 be within the inclusive range [1,{SS\_REPL\_MAX}] for the function to succeed; if not, the function  
 33371 shall fail.

33372 If the *pthread\_setschedparam()* function fails, the scheduling parameters shall not be changed for  
 33373 the target thread.

## 33374 RETURN VALUE

33375 If successful, the *pthread\_getschedparam()* and *pthread\_setschedparam()* functions shall return zero;  
 33376 otherwise, an error number shall be returned to indicate the error.

33377 **ERRORS**

33378 The *pthread\_getschedparam()* function may fail if:

33379 [ESRCH] The value specified by *thread* does not refer to an existing thread.

33380 The *pthread\_setschedparam()* function may fail if:

33381 [EINVAL] The value specified by *policy* or one of the scheduling parameters associated  
33382 with the scheduling policy *policy* is invalid.

33383 [ENOTSUP] An attempt was made to set the policy or scheduling parameters to an  
33384 unsupported value.

33385 TSP [ENOTSUP] An attempt was made to dynamically change the scheduling policy to  
33386 SCHED\_SPORADIC, and the implementation does not support this change.

33387 [EPERM] The caller does not have the appropriate permission to set either the  
33388 scheduling parameters or the scheduling policy of the specified thread.

33389 [EPERM] The implementation does not allow the application to modify one of the  
33390 parameters to the value specified.

33391 [ESRCH] The value specified by *thread* does not refer to a existing thread.

33392 These functions shall not return an error code of [EINTR].

33393 **EXAMPLES**

33394 None.

33395 **APPLICATION USAGE**

33396 None.

33397 **RATIONALE**

33398 None.

33399 **FUTURE DIRECTIONS**

33400 None.

33401 **SEE ALSO**

33402 *pthread\_setschedprio()*, *sched\_getparam()*, *sched\_getscheduler()*, the Base Definitions volume of  
33403 IEEE Std 1003.1-2001, <pthread.h>, <sched.h>

33404 **CHANGE HISTORY**

33405 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33406 **Issue 6**

33407 The *pthread\_getschedparam()* and *pthread\_setschedparam()* functions are marked as part of the  
33408 Threads and Thread Execution Scheduling options.

33409 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
33410 implementation does not support the Thread Execution Scheduling option.

33411 The Open Group Corrigendum U026/2 is applied, correcting the prototype for the  
33412 *pthread\_setschedparam()* function so that its second argument is of type **int**.

33413 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

33414 The **restrict** keyword is added to the *pthread\_getschedparam()* prototype for alignment with the  
33415 ISO/IEC 9899:1999 standard.

33416 The Open Group Corrigendum U047/1 is applied.

33417  
33418

IEEE PASC Interpretation 1003.1 #96 is applied, noting that priority values can also be set by a call to the *pthread\_setschedprio()* function.

33419 **NAME**

33420 pthread\_getspecific, pthread\_setspecific — thread-specific data management

33421 **SYNOPSIS**

33422 THR #include &lt;pthread.h&gt;

33423 void \*pthread\_getspecific(pthread\_key\_t key);

33424 int pthread\_setspecific(pthread\_key\_t key, const void \*value);

33425

33426 **DESCRIPTION**33427 The *pthread\_getspecific()* function shall return the value currently bound to the specified *key* on  
33428 behalf of the calling thread.33429 The *pthread\_setspecific()* function shall associate a thread-specific *value* with a *key* obtained via a  
33430 previous call to *pthread\_key\_create()*. Different threads may bind different values to the same  
33431 key. These values are typically pointers to blocks of dynamically allocated memory that have  
33432 been reserved for use by the calling thread.33433 The effect of calling *pthread\_getspecific()* or *pthread\_setspecific()* with a *key* value not obtained  
33434 from *pthread\_key\_create()* or after *key* has been deleted with *pthread\_key\_delete()* is undefined.33435 Both *pthread\_getspecific()* and *pthread\_setspecific()* may be called from a thread-specific data  
33436 destructor function. A call to *pthread\_getspecific()* for the thread-specific data key being  
33437 destroyed shall return the value NULL, unless the value is changed (after the destructor starts)  
33438 by a call to *pthread\_setspecific()*. Calling *pthread\_setspecific()* from a thread-specific data  
33439 destructor routine may result either in lost storage (after at least  
33440 PTHREAD\_DESTRUCTOR\_ITERATIONS attempts at destruction) or in an infinite loop.

33441 Both functions may be implemented as macros.

33442 **RETURN VALUE**33443 The *pthread\_getspecific()* function shall return the thread-specific data value associated with the  
33444 given *key*. If no thread-specific data value is associated with *key*, then the value NULL shall be  
33445 returned.33446 If successful, the *pthread\_setspecific()* function shall return zero; otherwise, an error number shall  
33447 be returned to indicate the error.33448 **ERRORS**33449 No errors are returned from *pthread\_getspecific()*.33450 The *pthread\_setspecific()* function shall fail if:

33451 [ENOMEM] Insufficient memory exists to associate the value with the key.

33452 The *pthread\_setspecific()* function may fail if:

33453 [EINVAL] The key value is invalid.

33454 These functions shall not return an error code of [EINTR].

33455 **EXAMPLES**

33456 None.

33457 **APPLICATION USAGE**

33458 None.

33459 **RATIONALE**

33460 Performance and ease-of-use of *pthread\_getspecific()* are critical for functions that rely on  
33461 maintaining state in thread-specific data. Since no errors are required to be detected by it, and  
33462 since the only error that could be detected is the use of an invalid key, the function to  
33463 *pthread\_getspecific()* has been designed to favor speed and simplicity over error reporting.

33464 **FUTURE DIRECTIONS**

33465 None.

33466 **SEE ALSO**33467 *pthread\_key\_create()*, the Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>33468 **CHANGE HISTORY**

33469 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33470 **Issue 6**

33471 The *pthread\_getspecific()* and *pthread\_setspecific()* functions are marked as part of the Threads  
33472 option.

33473 IEEE PASC Interpretation 1003.1c #3 (Part 6) is applied, updating the DESCRIPTION.

33474 **NAME**

33475 pthread\_join — wait for thread termination

33476 **SYNOPSIS**

33477 THR #include &lt;pthread.h&gt;

33478 int pthread\_join(pthread\_t thread, void \*\*value\_ptr);

33479

33480 **DESCRIPTION**

33481 The *pthread\_join()* function shall suspend execution of the calling thread until the target *thread*  
 33482 terminates, unless the target *thread* has already terminated. On return from a successful  
 33483 *pthread\_join()* call with a non-NULL *value\_ptr* argument, the value passed to *pthread\_exit()* by  
 33484 the terminating thread shall be made available in the location referenced by *value\_ptr*. When a  
 33485 *pthread\_join()* returns successfully, the target thread has been terminated. The results of multiple  
 33486 simultaneous calls to *pthread\_join()* specifying the same target thread are undefined. If the  
 33487 thread calling *pthread\_join()* is canceled, then the target thread shall not be detached.

33488 It is unspecified whether a thread that has exited but remains unjoined counts against  
 33489 {PTHREAD\_THREADS\_MAX}.

33490 **RETURN VALUE**

33491 If successful, the *pthread\_join()* function shall return zero; otherwise, an error number shall be  
 33492 returned to indicate the error.

33493 **ERRORS**33494 The *pthread\_join()* function shall fail if:

33495 [EINVAL] The implementation has detected that the value specified by *thread* does not  
 33496 refer to a joinable thread.

33497 [ESRCH] No thread could be found corresponding to that specified by the given thread  
 33498 ID.

33499 The *pthread\_join()* function may fail if:

33500 [EDEADLK] A deadlock was detected or the value of *thread* specifies the calling thread.

33501 The *pthread\_join()* function shall not return an error code of [EINTR].33502 **EXAMPLES**

33503 An example of thread creation and deletion follows:

```

33504 typedef struct {
33505 int *ar;
33506 long n;
33507 } subarray;
33508
33509 void *
33510 incer(void *arg)
33511 {
33512 long i;
33513 for (i = 0; i < ((subarray *)arg)->n; i++)
33514 ((subarray *)arg)->ar[i]++;
33515 }
33516
33517 int main(void)
33518 {
33519 int ar[1000000];

```

```

33518 pthread_t th1, th2;
33519 subarray sb1, sb2;

33520 sb1.ar = &ar[0];
33521 sb1.n = 500000;
33522 (void) pthread_create(&th1, NULL, incer, &sb1);

33523 sb2.ar = &ar[500000];
33524 sb2.n = 500000;
33525 (void) pthread_create(&th2, NULL, incer, &sb2);

33526 (void) pthread_join(th1, NULL);
33527 (void) pthread_join(th2, NULL);
33528 return 0;
33529 }

```

### 33530 APPLICATION USAGE

33531 None.

### 33532 RATIONALE

33533 The *pthread\_join()* function is a convenience that has proven useful in multi-threaded  
33534 applications. It is true that a programmer could simulate this function if it were not provided by  
33535 passing extra state as part of the argument to the *start\_routine()*. The terminating thread would  
33536 set a flag to indicate termination and broadcast a condition that is part of that state; a joining  
33537 thread would wait on that condition variable. While such a technique would allow a thread to  
33538 wait on more complex conditions (for example, waiting for multiple threads to terminate),  
33539 waiting on individual thread termination is considered widely useful. Also, including the  
33540 *pthread\_join()* function in no way precludes a programmer from coding such complex waits.  
33541 Thus, while not a primitive, including *pthread\_join()* in this volume of IEEE Std 1003.1-2001 was  
33542 considered valuable.

33543 The *pthread\_join()* function provides a simple mechanism allowing an application to wait for a  
33544 thread to terminate. After the thread terminates, the application may then choose to clean up  
33545 resources that were used by the thread. For instance, after *pthread\_join()* returns, any  
33546 application-provided stack storage could be reclaimed.

33547 The *pthread\_join()* or *pthread\_detach()* function should eventually be called for every thread that  
33548 is created with the *detachstate* attribute set to *PTHREAD\_CREATE\_JOINABLE* so that storage  
33549 associated with the thread may be reclaimed.

33550 The interaction between *pthread\_join()* and cancellation is well-defined for the following  
33551 reasons:

- 33552 • The *pthread\_join()* function, like all other non-async-cancel-safe functions, can only be called  
33553 with deferred cancelability type.
- 33554 • Cancellation cannot occur in the disabled cancelability state.

33555 Thus, only the default cancelability state need be considered. As specified, either the  
33556 *pthread\_join()* call is canceled, or it succeeds, but not both. The difference is obvious to the  
33557 application, since either a cancellation handler is run or *pthread\_join()* returns. There are no race  
33558 conditions since *pthread\_join()* was called in the deferred cancelability state.

### 33559 FUTURE DIRECTIONS

33560 None.

33561 **SEE ALSO**

33562 *pthread\_create()*, *wait()*, the Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

33563 **CHANGE HISTORY**

33564 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33565 **Issue 6**

33566 The *pthread\_join()* function is marked as part of the Threads option.

33567 **NAME**

33568 pthread\_key\_create — thread-specific data key creation

33569 **SYNOPSIS**

33570 THR #include &lt;pthread.h&gt;

33571 int pthread\_key\_create(pthread\_key\_t \*key, void (\*destructor)(void\*));

33572

33573 **DESCRIPTION**

33574 The *pthread\_key\_create()* function shall create a thread-specific data key visible to all threads in  
33575 the process. Key values provided by *pthread\_key\_create()* are opaque objects used to locate  
33576 thread-specific data. Although the same key value may be used by different threads, the values  
33577 bound to the key by *pthread\_setspecific()* are maintained on a per-thread basis and persist for the  
33578 life of the calling thread.

33579 Upon key creation, the value NULL shall be associated with the new key in all active threads.  
33580 Upon thread creation, the value NULL shall be associated with all defined keys in the new  
33581 thread.

33582 An optional destructor function may be associated with each key value. At thread exit, if a key  
33583 value has a non-NULL destructor pointer, and the thread has a non-NULL value associated with  
33584 that key, the value of the key is set to NULL, and then the function pointed to is called with the  
33585 previously associated value as its sole argument. The order of destructor calls is unspecified if  
33586 more than one destructor exists for a thread when it exits.

33587 If, after all the destructors have been called for all non-NULL values with associated destructors,  
33588 there are still some non-NULL values with associated destructors, then the process is repeated.  
33589 If, after at least {PTHREAD\_DESTRUCTOR\_ITERATIONS} iterations of destructor calls for  
33590 outstanding non-NULL values, there are still some non-NULL values with associated  
33591 destructors, implementations may stop calling destructors, or they may continue calling  
33592 destructors until no non-NULL values with associated destructors exist, even though this might  
33593 result in an infinite loop.

33594 **RETURN VALUE**

33595 If successful, the *pthread\_key\_create()* function shall store the newly created key value at *\*key* and  
33596 shall return zero. Otherwise, an error number shall be returned to indicate the error.

33597 **ERRORS**33598 The *pthread\_key\_create()* function shall fail if:

33599 [EAGAIN] The system lacked the necessary resources to create another thread-specific  
33600 data key, or the system-imposed limit on the total number of keys per process  
33601 {PTHREAD\_KEYS\_MAX} has been exceeded.

33602 [ENOMEM] Insufficient memory exists to create the key.

33603 The *pthread\_key\_create()* function shall not return an error code of [EINTR].

33604 **EXAMPLES**

33605 The following example demonstrates a function that initializes a thread-specific data key when  
 33606 it is first called, and associates a thread-specific object with each calling thread, initializing this  
 33607 object when necessary.

```

33608 static pthread_key_t key;
33609 static pthread_once_t key_once = PTHREAD_ONCE_INIT;

33610 static void
33611 make_key()
33612 {
33613 (void) pthread_key_create(&key, NULL);
33614 }

33615 func()
33616 {
33617 void *ptr;

33618 (void) pthread_once(&key_once, make_key);
33619 if ((ptr = pthread_getspecific(key)) == NULL) {
33620 ptr = malloc(OBJECT_SIZE);
33621 ...
33622 (void) pthread_setspecific(key, ptr);
33623 }
33624 ...
33625 }

```

33626 Note that the key has to be initialized before *pthread\_getspecific()* or *pthread\_setspecific()* can be  
 33627 used. The *pthread\_key\_create()* call could either be explicitly made in a module initialization  
 33628 routine, or it can be done implicitly by the first call to a module as in this example. Any attempt  
 33629 to use the key before it is initialized is a programming error, making the code below incorrect.

```

33630 static pthread_key_t key;

33631 func()
33632 {
33633 void *ptr;

33634 /* KEY NOT INITIALIZED!!! THIS WON'T WORK!!! */
33635 if ((ptr = pthread_getspecific(key)) == NULL &&
33636 pthread_setspecific(key, NULL) != 0) {
33637 pthread_key_create(&key, NULL);
33638 ...
33639 }
33640 }

```

33641 **APPLICATION USAGE**

33642 None.

## 33643 RATIONALE

33644 **Destructor Functions**

33645 Normally, the value bound to a key on behalf of a particular thread is a pointer to storage  
 33646 allocated dynamically on behalf of the calling thread. The destructor functions specified with  
 33647 *pthread\_key\_create()* are intended to be used to free this storage when the thread exits. Thread  
 33648 cancellation cleanup handlers cannot be used for this purpose because thread-specific data may  
 33649 persist outside the lexical scope in which the cancellation cleanup handlers operate.

33650 If the value associated with a key needs to be updated during the lifetime of the thread, it may  
 33651 be necessary to release the storage associated with the old value before the new value is bound.  
 33652 Although the *pthread\_setspecific()* function could do this automatically, this feature is not needed  
 33653 often enough to justify the added complexity. Instead, the programmer is responsible for freeing  
 33654 the stale storage:

```
33655 pthread_getspecific(key, &old);
33656 new = allocate();
33657 destructor(old);
33658 pthread_setspecific(key, new);
```

33659 **Note:** The above example could leak storage if run with asynchronous cancellation enabled. No such  
 33660 problems occur in the default cancellation state if no cancellation points occur between the get  
 33661 and set.

33662 There is no notion of a destructor-safe function. If an application does not call *pthread\_exit()*  
 33663 from a signal handler, or if it blocks any signal whose handler may call *pthread\_exit()* while  
 33664 calling async-unsafe functions, all functions may be safely called from destructors.

33665 **Non-Idempotent Data Key Creation**

33666 There were requests to make *pthread\_key\_create()* idempotent with respect to a given *key* address  
 33667 parameter. This would allow applications to call *pthread\_key\_create()* multiple times for a given  
 33668 *key* address and be guaranteed that only one key would be created. Doing so would require the  
 33669 key value to be previously initialized (possibly at compile time) to a known null value and  
 33670 would require that implicit mutual-exclusion be performed based on the address and contents of  
 33671 the *key* parameter in order to guarantee that exactly one key would be created.

33672 Unfortunately, the implicit mutual-exclusion would not be limited to only *pthread\_key\_create()*.  
 33673 On many implementations, implicit mutual-exclusion would also have to be performed by  
 33674 *pthread\_getspecific()* and *pthread\_setspecific()* in order to guard against using incompletely stored  
 33675 or not-yet-visible key values. This could significantly increase the cost of important operations,  
 33676 particularly *pthread\_getspecific()*.

33677 Thus, this proposal was rejected. The *pthread\_key\_create()* function performs no implicit  
 33678 synchronization. It is the responsibility of the programmer to ensure that it is called exactly once  
 33679 per key before use of the key. Several straightforward mechanisms can already be used to  
 33680 accomplish this, including calling explicit module initialization functions, using mutexes, and  
 33681 using *pthread\_once()*. This places no significant burden on the programmer, introduces no  
 33682 possibly confusing *ad hoc* implicit synchronization mechanism, and potentially allows  
 33683 commonly used thread-specific data operations to be more efficient.

33684 **FUTURE DIRECTIONS**

33685 None.

33686 **SEE ALSO**

33687        *pthread\_getspecific()*, *pthread\_key\_delete()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
33688        <pthread.h>

33689 **CHANGE HISTORY**

33690        First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33691 **Issue 6**

33692        The *pthread\_key\_create()* function is marked as part of the Threads option.

33693        IEEE PASC Interpretation 1003.1c #8 is applied, updating the DESCRIPTION.

33694 **NAME**

33695 pthread\_key\_delete — thread-specific data key deletion

33696 **SYNOPSIS**

33697 THR #include &lt;pthread.h&gt;

33698 int pthread\_key\_delete(pthread\_key\_t key);

33699

33700 **DESCRIPTION**

33701 The *pthread\_key\_delete()* function shall delete a thread-specific data key previously returned by  
33702 *pthread\_key\_create()*. The thread-specific data values associated with *key* need not be NULL at  
33703 the time *pthread\_key\_delete()* is called. It is the responsibility of the application to free any  
33704 application storage or perform any cleanup actions for data structures related to the deleted key  
33705 or associated thread-specific data in any threads; this cleanup can be done either before or after  
33706 *pthread\_key\_delete()* is called. Any attempt to use *key* following the call to *pthread\_key\_delete()*  
33707 results in undefined behavior.

33708 The *pthread\_key\_delete()* function shall be callable from within destructor functions. No  
33709 destructor functions shall be invoked by *pthread\_key\_delete()*. Any destructor function that may  
33710 have been associated with *key* shall no longer be called upon thread exit.

33711 **RETURN VALUE**

33712 If successful, the *pthread\_key\_delete()* function shall return zero; otherwise, an error number shall  
33713 be returned to indicate the error.

33714 **ERRORS**33715 The *pthread\_key\_delete()* function may fail if:33716 [EINVAL] The *key* value is invalid.33717 The *pthread\_key\_delete()* function shall not return an error code of [EINTR].33718 **EXAMPLES**

33719 None.

33720 **APPLICATION USAGE**

33721 None.

33722 **RATIONALE**

33723 A thread-specific data key deletion function has been included in order to allow the resources  
33724 associated with an unused thread-specific data key to be freed. Unused thread-specific data keys  
33725 can arise, among other scenarios, when a dynamically loaded module that allocated a key is  
33726 unloaded.

33727 Conforming applications are responsible for performing any cleanup actions needed for data  
33728 structures associated with the key to be deleted, including data referenced by thread-specific  
33729 data values. No such cleanup is done by *pthread\_key\_delete()*. In particular, destructor functions  
33730 are not called. There are several reasons for this division of responsibility:

- 33731 1. The associated destructor functions used to free thread-specific data at thread exit time are  
33732 only guaranteed to work correctly when called in the thread that allocated the thread-  
33733 specific data. (Destructors themselves may utilize thread-specific data.) Thus, they cannot  
33734 be used to free thread-specific data in other threads at key deletion time. Attempting to  
33735 have them called by other threads at key deletion time would require other threads to be  
33736 asynchronously interrupted. But since interrupted threads could be in an arbitrary state,  
33737 including holding locks necessary for the destructor to run, this approach would fail. In  
33738 general, there is no safe mechanism whereby an implementation could free thread-specific  
33739 data at key deletion time.

33740           2. Even if there were a means of safely freeing thread-specific data associated with keys to be  
33741 deleted, doing so would require that implementations be able to enumerate the threads  
33742 with non-NULL data and potentially keep them from creating more thread-specific data  
33743 while the key deletion is occurring. This special case could cause extra synchronization in  
33744 the normal case, which would otherwise be unnecessary.

33745           For an application to know that it is safe to delete a key, it has to know that all the threads that  
33746 might potentially ever use the key do not attempt to use it again. For example, it could know this  
33747 if all the client threads have called a cleanup procedure declaring that they are through with the  
33748 module that is being shut down, perhaps by setting a reference count to zero.

#### 33749 **FUTURE DIRECTIONS**

33750           None.

#### 33751 **SEE ALSO**

33752           *pthread\_key\_create()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>

#### 33753 **CHANGE HISTORY**

33754           First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 33755 **Issue 6**

33756           The *pthread\_key\_delete()* function is marked as part of the Threads option.

33757 **NAME**

33758 pthread\_kill — send a signal to a thread

33759 **SYNOPSIS**

33760 THR #include &lt;signal.h&gt;

33761 int pthread\_kill(pthread\_t thread, int sig);

33762

33763 **DESCRIPTION**33764 The *pthread\_kill()* function shall request that a signal be delivered to the specified thread.33765 As in *kill()*, if *sig* is zero, error checking shall be performed but no signal shall actually be sent.33766 **RETURN VALUE**

33767 Upon successful completion, the function shall return a value of zero. Otherwise, the function

33768 shall return an error number. If the *pthread\_kill()* function fails, no signal shall be sent.33769 **ERRORS**33770 The *pthread\_kill()* function shall fail if:33771 [ESRCH] No thread could be found corresponding to that specified by the given thread  
33772 ID.33773 [EINVAL] The value of the *sig* argument is an invalid or unsupported signal number.33774 The *pthread\_kill()* function shall not return an error code of [EINTR].33775 **EXAMPLES**

33776 None.

33777 **APPLICATION USAGE**33778 The *pthread\_kill()* function provides a mechanism for asynchronously directing a signal at a  
33779 thread in the calling process. This could be used, for example, by one thread to affect broadcast  
33780 delivery of a signal to a set of threads.33781 Note that *pthread\_kill()* only causes the signal to be handled in the context of the given thread;  
33782 the signal action (termination or stopping) affects the process as a whole.33783 **RATIONALE**

33784 None.

33785 **FUTURE DIRECTIONS**

33786 None.

33787 **SEE ALSO**33788 *kill()*, *pthread\_self()*, *raise()*, the Base Definitions volume of IEEE Std 1003.1-2001, <signal.h>33789 **CHANGE HISTORY**

33790 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33791 **Issue 6**33792 The *pthread\_kill()* function is marked as part of the Threads option.

33793 The APPLICATION USAGE section is added.

## 33794 NAME

33795 pthread\_mutex\_destroy, pthread\_mutex\_init — destroy and initialize a mutex

## 33796 SYNOPSIS

33797 THR #include <pthread.h>

```
33798 int pthread_mutex_destroy(pthread_mutex_t *mutex);
33799 int pthread_mutex_init(pthread_mutex_t *restrict mutex,
33800 const pthread_mutexattr_t *restrict attr);
33801 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
33802
```

## 33803 DESCRIPTION

33804 The *pthread\_mutex\_destroy()* function shall destroy the mutex object referenced by *mutex*; the  
 33805 mutex object becomes, in effect, uninitialized. An implementation may cause  
 33806 *pthread\_mutex\_destroy()* to set the object referenced by *mutex* to an invalid value. A destroyed  
 33807 mutex object can be reinitialized using *pthread\_mutex\_init()*; the results of otherwise referencing  
 33808 the object after it has been destroyed are undefined.

33809 It shall be safe to destroy an initialized mutex that is unlocked. Attempting to destroy a locked  
 33810 mutex results in undefined behavior.

33811 The *pthread\_mutex\_init()* function shall initialize the mutex referenced by *mutex* with attributes  
 33812 specified by *attr*. If *attr* is NULL, the default mutex attributes are used; the effect shall be the  
 33813 same as passing the address of a default mutex attributes object. Upon successful initialization,  
 33814 the state of the mutex becomes initialized and unlocked.

33815 Only *mutex* itself may be used for performing synchronization. The result of referring to copies  
 33816 of *mutex* in calls to *pthread\_mutex\_lock()*, *pthread\_mutex\_trylock()*, *pthread\_mutex\_unlock()*, and  
 33817 *pthread\_mutex\_destroy()* is undefined.

33818 Attempting to initialize an already initialized mutex results in undefined behavior.

33819 In cases where default mutex attributes are appropriate, the macro  
 33820 PTHREAD\_MUTEX\_INITIALIZER can be used to initialize mutexes that are statically allocated.  
 33821 The effect shall be equivalent to dynamic initialization by a call to *pthread\_mutex\_init()* with  
 33822 parameter *attr* specified as NULL, except that no error checks are performed.

## 33823 RETURN VALUE

33824 If successful, the *pthread\_mutex\_destroy()* and *pthread\_mutex\_init()* functions shall return zero;  
 33825 otherwise, an error number shall be returned to indicate the error.

33826 The [EBUSY] and [EINVAL] error checks, if implemented, act as if they were performed  
 33827 immediately at the beginning of processing for the function and shall cause an error return prior  
 33828 to modifying the state of the mutex specified by *mutex*.

## 33829 ERRORS

33830 The *pthread\_mutex\_destroy()* function may fail if:

33831 [EBUSY] The implementation has detected an attempt to destroy the object referenced  
 33832 by *mutex* while it is locked or referenced (for example, while being used in a  
 33833 *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()*) by another thread.

33834 [EINVAL] The value specified by *mutex* is invalid.

33835 The *pthread\_mutex\_init()* function shall fail if:

33836 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize  
 33837 another mutex.

- 33838 [ENOMEM] Insufficient memory exists to initialize the mutex.
- 33839 [EPERM] The caller does not have the privilege to perform the operation.
- 33840 The *pthread\_mutex\_init()* function may fail if:
- 33841 [EBUSY] The implementation has detected an attempt to reinitialize the object  
33842 referenced by *mutex*, a previously initialized, but not yet destroyed, mutex.
- 33843 [EINVAL] The value specified by *attr* is invalid.
- 33844 These functions shall not return an error code of [EINTR].

33845 **EXAMPLES**

33846 None.

33847 **APPLICATION USAGE**

33848 None.

33849 **RATIONALE**33850 **Alternate Implementations Possible**

33851 This volume of IEEE Std 1003.1-2001 supports several alternative implementations of mutexes.  
33852 An implementation may store the lock directly in the object of type **pthread\_mutex\_t**.  
33853 Alternatively, an implementation may store the lock in the heap and merely store a pointer,  
33854 handle, or unique ID in the mutex object. Either implementation has advantages or may be  
33855 required on certain hardware configurations. So that portable code can be written that is  
33856 invariant to this choice, this volume of IEEE Std 1003.1-2001 does not define assignment or  
33857 equality for this type, and it uses the term “initialize” to reinforce the (more restrictive) notion  
33858 that the lock may actually reside in the mutex object itself.

33859 Note that this precludes an over-specification of the type of the mutex or condition variable and  
33860 motivates the opaqueness of the type.

33861 An implementation is permitted, but not required, to have *pthread\_mutex\_destroy()* store an  
33862 illegal value into the mutex. This may help detect erroneous programs that try to lock (or  
33863 otherwise reference) a mutex that has already been destroyed.

33864 **Tradeoff Between Error Checks and Performance Supported**

33865 Many of the error checks were made optional in order to let implementations trade off  
33866 performance *versus* degree of error checking according to the needs of their specific applications  
33867 and execution environment. As a general rule, errors or conditions caused by the system (such as  
33868 insufficient memory) always need to be reported, but errors due to an erroneously coded  
33869 application (such as failing to provide adequate synchronization to prevent a mutex from being  
33870 deleted while in use) are made optional.

33871 A wide range of implementations is thus made possible. For example, an implementation  
33872 intended for application debugging may implement all of the error checks, but an  
33873 implementation running a single, provably correct application under very tight performance  
33874 constraints in an embedded computer might implement minimal checks. An implementation  
33875 might even be provided in two versions, similar to the options that compilers provide: a full-  
33876 checking, but slower version; and a limited-checking, but faster version. To forbid this  
33877 optionality would be a disservice to users.

33878 By carefully limiting the use of “undefined behavior” only to things that an erroneous (badly  
33879 coded) application might do, and by defining that resource-not-available errors are mandatory,  
33880 this volume of IEEE Std 1003.1-2001 ensures that a fully-conforming application is portable

33881 across the full range of implementations, while not forcing all implementations to add overhead  
33882 to check for numerous things that a correct program never does.

### 33883 **Why No Limits are Defined**

33884 Defining symbols for the maximum number of mutexes and condition variables was considered  
33885 but rejected because the number of these objects may change dynamically. Furthermore, many  
33886 implementations place these objects into application memory; thus, there is no explicit  
33887 maximum.

### 33888 **Static Initializers for Mutexes and Condition Variables**

33889 Providing for static initialization of statically allocated synchronization objects allows modules  
33890 with private static synchronization variables to avoid runtime initialization tests and overhead.  
33891 Furthermore, it simplifies the coding of self-initializing modules. Such modules are common in  
33892 C libraries, where for various reasons the design calls for self-initialization instead of requiring  
33893 an explicit module initialization function to be called. An example use of static initialization  
33894 follows.

33895 Without static initialization, a self-initializing routine *foo()* might look as follows:

```
33896 static pthread_once_t foo_once = PTHREAD_ONCE_INIT;
33897 static pthread_mutex_t foo_mutex;

33898 void foo_init()
33899 {
33900 pthread_mutex_init(&foo_mutex, NULL);
33901 }

33902 void foo()
33903 {
33904 pthread_once(&foo_once, foo_init);
33905 pthread_mutex_lock(&foo_mutex);
33906 /* Do work. */
33907 pthread_mutex_unlock(&foo_mutex);
33908 }
```

33909 With static initialization, the same routine could be coded as follows:

```
33910 static pthread_mutex_t foo_mutex = PTHREAD_MUTEX_INITIALIZER;
33911 void foo()
33912 {
33913 pthread_mutex_lock(&foo_mutex);
33914 /* Do work. */
33915 pthread_mutex_unlock(&foo_mutex);
33916 }
```

33917 Note that the static initialization both eliminates the need for the initialization test inside  
33918 *pthread\_once()* and the fetch of *&foo\_mutex* to learn the address to be passed to  
33919 *pthread\_mutex\_lock()* or *pthread\_mutex\_unlock()*.

33920 Thus, the C code written to initialize static objects is simpler on all systems and is also faster on a  
33921 large class of systems; those where the (entire) synchronization object can be stored in  
33922 application memory.

33923 Yet the locking performance question is likely to be raised for machines that require mutexes to  
33924 be allocated out of special memory. Such machines actually have to have mutexes and possibly

33925 condition variables contain pointers to the actual hardware locks. For static initialization to work  
33926 on such machines, *pthread\_mutex\_lock()* also has to test whether or not the pointer to the actual  
33927 lock has been allocated. If it has not, *pthread\_mutex\_lock()* has to initialize it before use. The  
33928 reservation of such resources can be made when the program is loaded, and hence return codes  
33929 have not been added to mutex locking and condition variable waiting to indicate failure to  
33930 complete initialization.

33931 This runtime test in *pthread\_mutex\_lock()* would at first seem to be extra work; an extra test is  
33932 required to see whether the pointer has been initialized. On most machines this would actually  
33933 be implemented as a fetch of the pointer, testing the pointer against zero, and then using the  
33934 pointer if it has already been initialized. While the test might seem to add extra work, the extra  
33935 effort of testing a register is usually negligible since no extra memory references are actually  
33936 done. As more and more machines provide caches, the real expenses are memory references, not  
33937 instructions executed.

33938 Alternatively, depending on the machine architecture, there are often ways to eliminate *all*  
33939 overhead in the most important case: on the lock operations that occur *after* the lock has been  
33940 initialized. This can be done by shifting more overhead to the less frequent operation:  
33941 initialization. Since out-of-line mutex allocation also means that an address has to be  
33942 dereferenced to find the actual lock, one technique that is widely applicable is to have static  
33943 initialization store a bogus value for that address; in particular, an address that causes a machine  
33944 fault to occur. When such a fault occurs upon the first attempt to lock such a mutex, validity  
33945 checks can be done, and then the correct address for the actual lock can be filled in. Subsequent  
33946 lock operations incur no extra overhead since they do not “fault”. This is merely one technique  
33947 that can be used to support static initialization, while not adversely affecting the performance of  
33948 lock acquisition. No doubt there are other techniques that are highly machine-dependent.

33949 The locking overhead for machines doing out-of-line mutex allocation is thus similar for  
33950 modules being implicitly initialized, where it is improved for those doing mutex allocation  
33951 entirely inline. The inline case is thus made much faster, and the out-of-line case is not  
33952 significantly worse.

33953 Besides the issue of locking performance for such machines, a concern is raised that it is possible  
33954 that threads would serialize contending for initialization locks when attempting to finish  
33955 initializing statically allocated mutexes. (Such finishing would typically involve taking an  
33956 internal lock, allocating a structure, storing a pointer to the structure in the mutex, and releasing  
33957 the internal lock.) First, many implementations would reduce such serialization by hashing on  
33958 the mutex address. Second, such serialization can only occur a bounded number of times. In  
33959 particular, it can happen at most as many times as there are statically allocated synchronization  
33960 objects. Dynamically allocated objects would still be initialized via *pthread\_mutex\_init()* or  
33961 *pthread\_cond\_init()*.

33962 Finally, if none of the above optimization techniques for out-of-line allocation yields sufficient  
33963 performance for an application on some implementation, the application can avoid static  
33964 initialization altogether by explicitly initializing all synchronization objects with the  
33965 corresponding *pthread\_\*\_init()* functions, which are supported by all implementations. An  
33966 implementation can also document the tradeoffs and advise which initialization technique is  
33967 more efficient for that particular implementation.

33968 **Destroying Mutexes**

33969 A mutex can be destroyed immediately after it is unlocked. For example, consider the following  
33970 code:

```
33971 struct obj {
33972 pthread_mutex_t om;
33973 int refcnt;
33974 ...
33975 };
33976 obj_done(struct obj *op)
33977 {
33978 pthread_mutex_lock(&op->om);
33979 if (--op->refcnt == 0) {
33980 pthread_mutex_unlock(&op->om);
33981 (A) pthread_mutex_destroy(&op->om);
33982 (B) free(op);
33983 } else
33984 (C) pthread_mutex_unlock(&op->om);
33985 }
```

33986 In this case *obj* is reference counted and *obj\_done()* is called whenever a reference to the object is  
33987 dropped. Implementations are required to allow an object to be destroyed and freed and  
33988 potentially unmapped (for example, lines A and B) immediately after the object is unlocked (line  
33989 C).

33990 **FUTURE DIRECTIONS**

33991 None.

33992 **SEE ALSO**

33993 *pthread\_mutex\_getprioceiling()*, *pthread\_mutex\_lock()*, *pthread\_mutex\_timedlock()*,  
33994 *pthread\_mutexattr\_getpshared()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
33995 <pthread.h>

33996 **CHANGE HISTORY**

33997 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33998 **Issue 6**

33999 The *pthread\_mutex\_destroy()* and *pthread\_mutex\_init()* functions are marked as part of the  
34000 Threads option.

34001 The *pthread\_mutex\_timedlock()* function is added to the SEE ALSO section for alignment with  
34002 IEEE Std 1003.1d-1999.

34003 IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

34004 The **restrict** keyword is added to the *pthread\_mutex\_init()* prototype for alignment with the  
34005 ISO/IEC 9899:1999 standard.

34006 **NAME**

34007 pthread\_mutex\_getprioceiling, pthread\_mutex\_setprioceiling — get and set the priority ceiling  
 34008 of a mutex (**REALTIME THREADS**)

34009 **SYNOPSIS**

34010 THR TPP #include <pthread.h>

```
34011 int pthread_mutex_getprioceiling(const pthread_mutex_t *restrict mutex,
34012 int *restrict prioceiling);
```

```
34013 int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,
34014 int prioceiling, int *restrict old_ceiling);
```

34015

34016 **DESCRIPTION**

34017 The *pthread\_mutex\_getprioceiling()* function shall return the current priority ceiling of the mutex.

34018 The *pthread\_mutex\_setprioceiling()* function shall either lock the mutex if it is unlocked, or block  
 34019 until it can successfully lock the mutex, then it shall change the mutex's priority ceiling and  
 34020 release the mutex. When the change is successful, the previous value of the priority ceiling shall  
 34021 be returned in *old\_ceiling*. The process of locking the mutex need not adhere to the priority  
 34022 protect protocol.

34023 If the *pthread\_mutex\_setprioceiling()* function fails, the mutex priority ceiling shall not be  
 34024 changed.

34025 **RETURN VALUE**

34026 If successful, the *pthread\_mutex\_getprioceiling()* and *pthread\_mutex\_setprioceiling()* functions shall  
 34027 return zero; otherwise, an error number shall be returned to indicate the error.

34028 **ERRORS**

34029 The *pthread\_mutex\_getprioceiling()* and *pthread\_mutex\_setprioceiling()* functions may fail if:

34030 [EINVAL] The priority requested by *prioceiling* is out of range.

34031 [EINVAL] The value specified by *mutex* does not refer to a currently existing mutex.

34032 [EPERM] The caller does not have the privilege to perform the operation.

34033 These functions shall not return an error code of [EINTR].

34034 **EXAMPLES**

34035 None.

34036 **APPLICATION USAGE**

34037 None.

34038 **RATIONALE**

34039 None.

34040 **FUTURE DIRECTIONS**

34041 None.

34042 **SEE ALSO**

34043 *pthread\_mutex\_destroy()*, *pthread\_mutex\_lock()*, *pthread\_mutex\_timedlock()*, the Base Definitions  
 34044 volume of IEEE Std 1003.1-2001, <pthread.h>

34045 **CHANGE HISTORY**

34046 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34047 Marked as part of the Realtime Threads Feature Group.

34048 **Issue 6**

34049 The *pthread\_mutex\_getprioceiling()* and *pthread\_mutex\_setprioceiling()* functions are marked as  
34050 part of the Threads and Thread Priority Protection options.

34051 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
34052 implementation does not support the Thread Priority Protection option.

34053 The [ENOSYS] error denoting non-support of the priority ceiling protocol for mutexes has been  
34054 removed. This is because if the implementation provides the functions (regardless of whether  
34055 `_POSIX_PTHREAD_PRIO_PROTECT` is defined), they must function as in the DESCRIPTION  
34056 and therefore the priority ceiling protocol for mutexes is supported.

34057 The *pthread\_mutex\_timedlock()* function is added to the SEE ALSO section for alignment with  
34058 IEEE Std 1003.1d-1999.

34059 The **restrict** keyword is added to the *pthread\_mutex\_getprioceiling()* and  
34060 *pthread\_mutex\_setprioceiling()* prototypes for alignment with the ISO/IEC 9899:1999 standard.

34061 **NAME**

34062 pthread\_mutex\_init — initialize a mutex

34063 **SYNOPSIS**

34064 THR #include &lt;pthread.h&gt;

34065 int pthread\_mutex\_init(pthread\_mutex\_t \*restrict mutex,

34066 const pthread\_mutexattr\_t \*restrict attr);

34067 pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

34068

34069 **DESCRIPTION**34070 Refer to *pthread\_mutex\_destroy()*.

## 34071 NAME

34072 pthread\_mutex\_lock, pthread\_mutex\_trylock, pthread\_mutex\_unlock — lock and unlock a  
34073 mutex

## 34074 SYNOPSIS

```
34075 THR #include <pthread.h>
```

```
34076 int pthread_mutex_lock(pthread_mutex_t *mutex);
34077 int pthread_mutex_trylock(pthread_mutex_t *mutex);
34078 int pthread_mutex_unlock(pthread_mutex_t *mutex);
34079
```

## 34080 DESCRIPTION

34081 The mutex object referenced by *mutex* shall be locked by calling *pthread\_mutex\_lock()*. If the  
34082 mutex is already locked, the calling thread shall block until the mutex becomes available. This  
34083 operation shall return with the mutex object referenced by *mutex* in the locked state with the  
34084 calling thread as its owner.

34085 XSI If the mutex type is PTHREAD\_MUTEX\_NORMAL, deadlock detection shall not be provided.  
34086 Attempting to relock the mutex causes deadlock. If a thread attempts to unlock a mutex that it  
34087 has not locked or a mutex which is unlocked, undefined behavior results.

34088 If the mutex type is PTHREAD\_MUTEX\_ERRORCHECK, then error checking shall be provided.  
34089 If a thread attempts to relock a mutex that it has already locked, an error shall be returned. If a  
34090 thread attempts to unlock a mutex that it has not locked or a mutex which is unlocked, an error  
34091 shall be returned.

34092 If the mutex type is PTHREAD\_MUTEX\_RECURSIVE, then the mutex shall maintain the  
34093 concept of a lock count. When a thread successfully acquires a mutex for the first time, the lock  
34094 count shall be set to one. Every time a thread relocks this mutex, the lock count shall be  
34095 incremented by one. Each time the thread unlocks the mutex, the lock count shall be  
34096 decremented by one. When the lock count reaches zero, the mutex shall become available for  
34097 other threads to acquire. If a thread attempts to unlock a mutex that it has not locked or a mutex  
34098 which is unlocked, an error shall be returned.

34099 If the mutex type is PTHREAD\_MUTEX\_DEFAULT, attempting to recursively lock the mutex  
34100 results in undefined behavior. Attempting to unlock the mutex if it was not locked by the calling  
34101 thread results in undefined behavior. Attempting to unlock the mutex if it is not locked results in  
34102 undefined behavior.

34103 The *pthread\_mutex\_trylock()* function shall be equivalent to *pthread\_mutex\_lock()*, except that if  
34104 the mutex object referenced by *mutex* is currently locked (by any thread, including the current  
34105 thread), the call shall return immediately. If the mutex type is PTHREAD\_MUTEX\_RECURSIVE  
34106 and the mutex is currently owned by the calling thread, the mutex lock count shall be  
34107 incremented by one and the *pthread\_mutex\_trylock()* function shall immediately return success.

34108 XSI The *pthread\_mutex\_unlock()* function shall release the mutex object referenced by *mutex*. The  
34109 manner in which a mutex is released is dependent upon the mutex's type attribute. If there are  
34110 threads blocked on the mutex object referenced by *mutex* when *pthread\_mutex\_unlock()* is called,  
34111 resulting in the mutex becoming available, the scheduling policy shall determine which thread  
34112 shall acquire the mutex.

34113 XSI (In the case of PTHREAD\_MUTEX\_RECURSIVE mutexes, the mutex shall become available  
34114 when the count reaches zero and the calling thread no longer has any locks on this mutex.)

34115 If a signal is delivered to a thread waiting for a mutex, upon return from the signal handler the  
34116 thread shall resume waiting for the mutex as if it was not interrupted.

34117 **RETURN VALUE**

34118 If successful, the *pthread\_mutex\_lock()* and *pthread\_mutex\_unlock()* functions shall return zero;  
 34119 otherwise, an error number shall be returned to indicate the error.

34120 The *pthread\_mutex\_trylock()* function shall return zero if a lock on the mutex object referenced by  
 34121 *mutex* is acquired. Otherwise, an error number is returned to indicate the error.

34122 **ERRORS**

34123 The *pthread\_mutex\_lock()* and *pthread\_mutex\_trylock()* functions shall fail if:

34124 [EINVAL] The *mutex* was created with the protocol attribute having the value  
 34125 PTHREAD\_PRIO\_PROTECT and the calling thread's priority is higher than  
 34126 the mutex's current priority ceiling.

34127 The *pthread\_mutex\_trylock()* function shall fail if:

34128 [EBUSY] The *mutex* could not be acquired because it was already locked.

34129 The *pthread\_mutex\_lock()*, *pthread\_mutex\_trylock()*, and *pthread\_mutex\_unlock()* functions may  
 34130 fail if:

34131 [EINVAL] The value specified by *mutex* does not refer to an initialized mutex object.

34132 XSI [EAGAIN] The mutex could not be acquired because the maximum number of recursive  
 34133 locks for *mutex* has been exceeded.

34134 The *pthread\_mutex\_lock()* function may fail if:

34135 [EDEADLK] The current thread already owns the mutex.

34136 The *pthread\_mutex\_unlock()* function may fail if:

34137 [EPERM] The current thread does not own the mutex.

34138 These functions shall not return an error code of [EINTR].

34139 **EXAMPLES**

34140 None.

34141 **APPLICATION USAGE**

34142 None.

34143 **RATIONALE**

34144 Mutex objects are intended to serve as a low-level primitive from which other thread  
 34145 synchronization functions can be built. As such, the implementation of mutexes should be as  
 34146 efficient as possible, and this has ramifications on the features available at the interface.

34147 The mutex functions and the particular default settings of the mutex attributes have been  
 34148 motivated by the desire to not preclude fast, inlined implementations of mutex locking and  
 34149 unlocking.

34150 For example, deadlocking on a double-lock is explicitly allowed behavior in order to avoid  
 34151 requiring more overhead in the basic mechanism than is absolutely necessary. (More "friendly"  
 34152 mutexes that detect deadlock or that allow multiple locking by the same thread are easily  
 34153 constructed by the user via the other mechanisms provided. For example, *pthread\_self()* can be  
 34154 used to record mutex ownership.) Implementations might also choose to provide such extended  
 34155 features as options via special mutex attributes.

34156 Since most attributes only need to be checked when a thread is going to be blocked, the use of  
 34157 attributes does not slow the (common) mutex-locking case.

34158 Likewise, while being able to extract the thread ID of the owner of a mutex might be desirable, it  
34159 would require storing the current thread ID when each mutex is locked, and this could incur  
34160 unacceptable levels of overhead. Similar arguments apply to a *mutex\_tryunlock* operation.

34161 **FUTURE DIRECTIONS**

34162 None.

34163 **SEE ALSO**

34164 *pthread\_mutex\_destroy()*, *pthread\_mutex\_timedlock()*, the Base Definitions volume of  
34165 IEEE Std 1003.1-2001, <pthread.h>

34166 **CHANGE HISTORY**

34167 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34168 **Issue 6**

34169 The *pthread\_mutex\_lock()*, *pthread\_mutex\_trylock()*, and *pthread\_mutex\_unlock()* functions are  
34170 marked as part of the Threads option.

34171 The following new requirements on POSIX implementations derive from alignment with the  
34172 Single UNIX Specification:

- 34173 • The behavior when attempting to relock a mutex is defined.

34174 The *pthread\_mutex\_timedlock()* function is added to the SEE ALSO section for alignment with  
34175 IEEE Std 1003.1d-1999.

34176 **NAME**

34177 pthread\_mutex\_setprioceiling — change the priority ceiling of a mutex (**REALTIME**  
34178 **THREADS**)

34179 **SYNOPSIS**

```
34180 THR TPP #include <pthread.h>
```

```
34181 int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,
34182 int prioceiling, int *restrict old_ceiling);
```

34183

34184 **DESCRIPTION**

34185 Refer to *pthread\_mutex\_getprioceiling()*.

## 34186 NAME

34187 pthread\_mutex\_timedlock — lock a mutex (ADVANCED REALTIME)

## 34188 SYNOPSIS

34189 THR TMO #include &lt;pthread.h&gt;

34190 #include &lt;time.h&gt;

34191 int pthread\_mutex\_timedlock(pthread\_mutex\_t \*restrict mutex,

34192 const struct timespec \*restrict abs\_timeout);

34193

## 34194 DESCRIPTION

34195 The *pthread\_mutex\_timedlock()* function shall lock the mutex object referenced by *mutex*. If the  
 34196 mutex is already locked, the calling thread shall block until the mutex becomes available as in  
 34197 the *pthread\_mutex\_lock()* function. If the mutex cannot be locked without waiting for another  
 34198 thread to unlock the mutex, this wait shall be terminated when the specified timeout expires.

34199 The timeout shall expire when the absolute time specified by *abs\_timeout* passes, as measured by  
 34200 the clock on which timeouts are based (that is, when the value of that clock equals or exceeds  
 34201 *abs\_timeout*), or if the absolute time specified by *abs\_timeout* has already been passed at the time  
 34202 of the call.

34203 TMR If the Timers option is supported, the timeout shall be based on the CLOCK\_REALTIME clock; if  
 34204 the Timers option is not supported, the timeout shall be based on the system clock as returned  
 34205 by the *time()* function.

34206 The resolution of the timeout shall be the resolution of the clock on which it is based. The  
 34207 **timespec** data type is defined in the <time.h> header.

34208 Under no circumstance shall the function fail with a timeout if the mutex can be locked  
 34209 immediately. The validity of the *abs\_timeout* parameter need not be checked if the mutex can be  
 34210 locked immediately.

34211 As a consequence of the priority inheritance rules (for mutexes initialized with the  
 34212 PRIO\_INHERIT protocol), if a timed mutex wait is terminated because its timeout expires, the  
 34213 priority of the owner of the mutex shall be adjusted as necessary to reflect the fact that this  
 34214 thread is no longer among the threads waiting for the mutex.

## 34215 RETURN VALUE

34216 If successful, the *pthread\_mutex\_timedlock()* function shall return zero; otherwise, an error  
 34217 number shall be returned to indicate the error.

## 34218 ERRORS

34219 The *pthread\_mutex\_timedlock()* function shall fail if:

34220 [EINVAL] The mutex was created with the protocol attribute having the value  
 34221 PTHREAD\_PRIO\_PROTECT and the calling thread's priority is higher than  
 34222 the mutex' current priority ceiling.

34223 [EINVAL] The process or thread would have blocked, and the *abs\_timeout* parameter  
 34224 specified a nanoseconds field value less than zero or greater than or equal to  
 34225 1 000 million.

34226 [ETIMEDOUT] The mutex could not be locked before the specified timeout expired.

34227 The *pthread\_mutex\_timedlock()* function may fail if:

34228 [EINVAL] The value specified by *mutex* does not refer to an initialized mutex object.

34229 XSI [EAGAIN] The mutex could not be acquired because the maximum number of recursive  
34230 locks for *mutex* has been exceeded.

34231 [EDEADLK] The current thread already owns the mutex.

34232 This function shall not return an error code of [EINTR].

#### 34233 EXAMPLES

34234 None.

#### 34235 APPLICATION USAGE

34236 The *pthread\_mutex\_timedlock()* function is part of the Threads and Timeouts options and need  
34237 not be provided on all implementations.

#### 34238 RATIONALE

34239 None.

#### 34240 FUTURE DIRECTIONS

34241 None.

#### 34242 SEE ALSO

34243 *pthread\_mutex\_destroy()*, *pthread\_mutex\_lock()*, *pthread\_mutex\_trylock()*, *time()*, the Base  
34244 Definitions volume of IEEE Std 1003.1-2001, <pthread.h>, <time.h>

#### 34245 CHANGE HISTORY

34246 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

34247 **NAME**

34248 pthread\_mutex\_trylock, pthread\_mutex\_unlock — lock and unlock a mutex

34249 **SYNOPSIS**

34250 THR #include <pthread.h>

34251 int pthread\_mutex\_trylock(pthread\_mutex\_t \*mutex);

34252 int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);

34253

34254 **DESCRIPTION**

34255 Refer to *pthread\_mutex\_lock()*.

34256 **NAME**

34257 pthread\_mutexattr\_destroy, pthread\_mutexattr\_init — destroy and initialize the mutex  
 34258 attributes object

34259 **SYNOPSIS**

34260 THR #include <pthread.h>

34261 int pthread\_mutexattr\_destroy(pthread\_mutexattr\_t \*attr);

34262 int pthread\_mutexattr\_init(pthread\_mutexattr\_t \*attr);

34263

34264 **DESCRIPTION**

34265 The *pthread\_mutexattr\_destroy()* function shall destroy a mutex attributes object; the object  
 34266 becomes, in effect, uninitialized. An implementation may cause *pthread\_mutexattr\_destroy()* to  
 34267 set the object referenced by *attr* to an invalid value. A destroyed *attr* attributes object can be  
 34268 reinitialized using *pthread\_mutexattr\_init()*; the results of otherwise referencing the object after it  
 34269 has been destroyed are undefined.

34270 The *pthread\_mutexattr\_init()* function shall initialize a mutex attributes object *attr* with the  
 34271 default value for all of the attributes defined by the implementation.

34272 Results are undefined if *pthread\_mutexattr\_init()* is called specifying an already initialized *attr*  
 34273 attributes object.

34274 After a mutex attributes object has been used to initialize one or more mutexes, any function  
 34275 affecting the attributes object (including destruction) shall not affect any previously initialized  
 34276 mutexes.

34277 **RETURN VALUE**

34278 Upon successful completion, *pthread\_mutexattr\_destroy()* and *pthread\_mutexattr\_init()* shall  
 34279 return zero; otherwise, an error number shall be returned to indicate the error.

34280 **ERRORS**

34281 The *pthread\_mutexattr\_destroy()* function may fail if:

34282 [EINVAL] The value specified by *attr* is invalid.

34283 The *pthread\_mutexattr\_init()* function shall fail if:

34284 [ENOMEM] Insufficient memory exists to initialize the mutex attributes object.

34285 These functions shall not return an error code of [EINTR].

34286 **EXAMPLES**

34287 None.

34288 **APPLICATION USAGE**

34289 None.

34290 **RATIONALE**

34291 See *pthread\_attr\_init()* for a general explanation of attributes. Attributes objects allow  
 34292 implementations to experiment with useful extensions and permit extension of this volume of  
 34293 IEEE Std 1003.1-2001 without changing the existing functions. Thus, they provide for future  
 34294 extensibility of this volume of IEEE Std 1003.1-2001 and reduce the temptation to standardize  
 34295 prematurely on semantics that are not yet widely implemented or understood.

34296 Examples of possible additional mutex attributes that have been discussed are *spin\_only*,  
 34297 *limited\_spin*, *no\_spin*, *recursive*, and *metered*. (To explain what the latter attributes might mean:  
 34298 recursive mutexes would allow for multiple re-locking by the current owner; metered mutexes  
 34299 would transparently keep records of queue length, wait time, and so on.) Since there is not yet

34300 wide agreement on the usefulness of these resulting from shared implementation and usage  
34301 experience, they are not yet specified in this volume of IEEE Std 1003.1-2001. Mutex attributes  
34302 objects, however, make it possible to test out these concepts for possible standardization at a  
34303 later time.

#### 34304 **Mutex Attributes and Performance**

34305 Care has been taken to ensure that the default values of the mutex attributes have been defined  
34306 such that mutexes initialized with the defaults have simple enough semantics so that the locking  
34307 and unlocking can be done with the equivalent of a test-and-set instruction (plus possibly a few  
34308 other basic instructions).

34309 There is at least one implementation method that can be used to reduce the cost of testing at  
34310 lock-time if a mutex has non-default attributes. One such method that an implementation can  
34311 employ (and this can be made fully transparent to fully conforming POSIX applications) is to  
34312 secretly pre-lock any mutexes that are initialized to non-default attributes. Any later attempt to  
34313 lock such a mutex causes the implementation to branch to the “slow path” as if the mutex were  
34314 unavailable; then, on the slow path, the implementation can do the “real work” to lock a non-  
34315 default mutex. The underlying unlock operation is more complicated since the implementation  
34316 never really wants to release the pre-lock on this kind of mutex. This illustrates that, depending  
34317 on the hardware, there may be certain optimizations that can be used so that whatever mutex  
34318 attributes are considered “most frequently used” can be processed most efficiently.

#### 34319 **Process Shared Memory and Synchronization**

34320 The existence of memory mapping functions in this volume of IEEE Std 1003.1-2001 leads to the  
34321 possibility that an application may allocate the synchronization objects from this section in  
34322 memory that is accessed by multiple processes (and therefore, by threads of multiple processes).

34323 In order to permit such usage, while at the same time keeping the usual case (that is, usage  
34324 within a single process) efficient, a *process-shared* option has been defined.

34325 If an implementation supports the `_POSIX_THREAD_PROCESS_SHARED` option, then the  
34326 *process-shared* attribute can be used to indicate that mutexes or condition variables may be  
34327 accessed by threads of multiple processes.

34328 The default setting of `PTHREAD_PROCESS_PRIVATE` has been chosen for the *process-shared*  
34329 attribute so that the most efficient forms of these synchronization objects are created by default.

34330 Synchronization variables that are initialized with the `PTHREAD_PROCESS_PRIVATE` *process-*  
34331 *shared* attribute may only be operated on by threads in the process that initialized them.  
34332 Synchronization variables that are initialized with the `PTHREAD_PROCESS_SHARED` *process-*  
34333 *shared* attribute may be operated on by any thread in any process that has access to it. In  
34334 particular, these processes may exist beyond the lifetime of the initializing process. For example,  
34335 the following code implements a simple counting semaphore in a mapped file that may be used  
34336 by many processes.

```
34337 /* sem.h */
34338 struct semaphore {
34339 pthread_mutex_t lock;
34340 pthread_cond_t nonzero;
34341 unsigned count;
34342 };
34343 typedef struct semaphore semaphore_t;

34344 semaphore_t *semaphore_create(char *semaphore_name);
34345 semaphore_t *semaphore_open(char *semaphore_name);
```

```
34346 void semaphore_post(semaphore_t *semap);
34347 void semaphore_wait(semaphore_t *semap);
34348 void semaphore_close(semaphore_t *semap);

34349 /* sem.c */
34350 #include <sys/types.h>
34351 #include <sys/stat.h>
34352 #include <sys/mman.h>
34353 #include <fcntl.h>
34354 #include <pthread.h>
34355 #include "sem.h"

34356 semaphore_t *
34357 semaphore_create(char *semaphore_name)
34358 {
34359 int fd;
34360 semaphore_t *semap;
34361 pthread_mutexattr_t psharedm;
34362 pthread_condattr_t psharedc;

34363 fd = open(semaphore_name, O_RDWR | O_CREAT | O_EXCL, 0666);
34364 if (fd < 0)
34365 return (NULL);
34366 (void) ftruncate(fd, sizeof(semaphore_t));
34367 (void) pthread_mutexattr_init(&psharedm);
34368 (void) pthread_mutexattr_setpshared(&psharedm,
34369 PTHREAD_PROCESS_SHARED);
34370 (void) pthread_condattr_init(&psharedc);
34371 (void) pthread_condattr_setpshared(&psharedc,
34372 PTHREAD_PROCESS_SHARED);
34373 semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
34374 PROT_READ | PROT_WRITE, MAP_SHARED,
34375 fd, 0);
34376 close (fd);
34377 (void) pthread_mutex_init(&semap->lock, &psharedm);
34378 (void) pthread_cond_init(&semap->nonzero, &psharedc);
34379 semap->count = 0;
34380 return (semap);
34381 }

34382 semaphore_t *
34383 semaphore_open(char *semaphore_name)
34384 {
34385 int fd;
34386 semaphore_t *semap;

34387 fd = open(semaphore_name, O_RDWR, 0666);
34388 if (fd < 0)
34389 return (NULL);
34390 semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
34391 PROT_READ | PROT_WRITE, MAP_SHARED,
34392 fd, 0);
34393 close (fd);
34394 return (semap);
34395 }
```

```

34396 void
34397 semaphore_post(semaphore_t *semap)
34398 {
34399 pthread_mutex_lock(&semap->lock);
34400 if (semap->count == 0)
34401 pthread_cond_signal(&semap->nonzero);
34402 semap->count++;
34403 pthread_mutex_unlock(&semap->lock);
34404 }

34405 void
34406 semaphore_wait(semaphore_t *semap)
34407 {
34408 pthread_mutex_lock(&semap->lock);
34409 while (semap->count == 0)
34410 pthread_cond_wait(&semap->nonzero, &semap->lock);
34411 semap->count--;
34412 pthread_mutex_unlock(&semap->lock);
34413 }

34414 void
34415 semaphore_close(semaphore_t *semap)
34416 {
34417 munmap((void *) semap, sizeof(semaphore_t));
34418 }

```

34419 The following code is for three separate processes that create, post, and wait on a semaphore in  
34420 the file **/tmp/semaphore**. Once the file is created, the post and wait programs increment and  
34421 decrement the counting semaphore (waiting and waking as required) even though they did not  
34422 initialize the semaphore.

```

34423 /* create.c */
34424 #include "pthread.h"
34425 #include "sem.h"

34426 int
34427 main()
34428 {
34429 semaphore_t *semap;

34430 semap = semaphore_create("/tmp/semaphore");
34431 if (semap == NULL)
34432 exit(1);
34433 semaphore_close(semap);
34434 return (0);
34435 }

34436 /* post */
34437 #include "pthread.h"
34438 #include "sem.h"

34439 int
34440 main()
34441 {
34442 semaphore_t *semap;

```

```
34443 semap = semaphore_open("/tmp/semaphore");
34444 if (semap == NULL)
34445 exit(1);
34446 semaphore_post(semap);
34447 semaphore_close(semap);
34448 return (0);
34449 }
34450 /* wait */
34451 #include "pthread.h"
34452 #include "sem.h"
34453 int
34454 main()
34455 {
34456 semaphore_t *semap;
34457 semap = semaphore_open("/tmp/semaphore");
34458 if (semap == NULL)
34459 exit(1);
34460 semaphore_wait(semap);
34461 semaphore_close(semap);
34462 return (0);
34463 }
```

**34464 FUTURE DIRECTIONS**

34465 None.

**34466 SEE ALSO**

34467 *pthread\_cond\_destroy()*, *pthread\_create()*, *pthread\_mutex\_destroy()*, *pthread\_mutexattr\_destroy()*, the  
34468 Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

**34469 CHANGE HISTORY**

34470 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**34471 Issue 6**

34472 The *pthread\_mutexattr\_destroy()* and *pthread\_mutexattr\_init()* functions are marked as part of the  
34473 Threads option.

34474 IEEE PASC Interpretation 1003.1c #27 is applied, updating the ERRORS section.

34475 **NAME**

34476 pthread\_mutexattr\_getprioceiling, pthread\_mutexattr\_setprioceiling — get and set the  
 34477 prioceiling attribute of the mutex attributes object (**REALTIME THREADS**)

34478 **SYNOPSIS**

34479 THR TPP #include <pthread.h>

```
34480 int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t *
34481 restrict attr, int *restrict prioceiling);
34482 int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
34483 int prioceiling);
34484
```

34485 **DESCRIPTION**

34486 The *pthread\_mutexattr\_getprioceiling()* and *pthread\_mutexattr\_setprioceiling()* functions,  
 34487 respectively, shall get and set the priority ceiling attribute of a mutex attributes object pointed to  
 34488 by *attr* which was previously created by the function *pthread\_mutexattr\_init()*.

34489 The *prioceiling* attribute contains the priority ceiling of initialized mutexes. The values of  
 34490 *prioceiling* are within the maximum range of priorities defined by SCHED\_FIFO.

34491 The *prioceiling* attribute defines the priority ceiling of initialized mutexes, which is the minimum  
 34492 priority level at which the critical section guarded by the mutex is executed. In order to avoid  
 34493 priority inversion, the priority ceiling of the mutex shall be set to a priority higher than or equal  
 34494 to the highest priority of all the threads that may lock that mutex. The values of *prioceiling* are  
 34495 within the maximum range of priorities defined under the SCHED\_FIFO scheduling policy.

34496 **RETURN VALUE**

34497 Upon successful completion, the *pthread\_mutexattr\_getprioceiling()* and  
 34498 *pthread\_mutexattr\_setprioceiling()* functions shall return zero; otherwise, an error number shall be  
 34499 returned to indicate the error.

34500 **ERRORS**

34501 The *pthread\_mutexattr\_getprioceiling()* and *pthread\_mutexattr\_setprioceiling()* functions may fail if:

34502 [EINVAL] The value specified by *attr* or *prioceiling* is invalid.

34503 [EPERM] The caller does not have the privilege to perform the operation.

34504 These functions shall not return an error code of [EINTR].

34505 **EXAMPLES**

34506 None.

34507 **APPLICATION USAGE**

34508 None.

34509 **RATIONALE**

34510 None.

34511 **FUTURE DIRECTIONS**

34512 None.

34513 **SEE ALSO**

34514 *pthread\_cond\_destroy()*, *pthread\_create()*, *pthread\_mutex\_destroy()*, the Base Definitions volume of  
 34515 IEEE Std 1003.1-2001, <pthread.h>

34516 **CHANGE HISTORY**

34517 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34518 Marked as part of the Realtime Threads Feature Group.

34519 **Issue 6**

34520 The `pthread_mutexattr_getprioceiling()` and `pthread_mutexattr_setprioceiling()` functions are marked as part of the Threads and Thread Priority Protection options.

34522 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Priority Protection option.

34524 The [ENOTSUP] error condition has been removed since these functions do not have a *protocol* argument.

34526 The **restrict** keyword is added to the `pthread_mutexattr_getprioceiling()` prototype for alignment with the ISO/IEC 9899:1999 standard.

34527

## 34528 NAME

34529 pthread\_mutexattr\_getprotocol, pthread\_mutexattr\_setprotocol — get and set the protocol  
 34530 attribute of the mutex attributes object (**REALTIME THREADS**)

## 34531 SYNOPSIS

```
34532 THR #include <pthread.h>
34533 TPP|TPI int pthread_mutexattr_getprotocol(const pthread_mutexattr_t *
34534 restrict attr, int *restrict protocol);
34535 int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
34536 int protocol);
34537
```

## 34538 DESCRIPTION

34539 The *pthread\_mutexattr\_getprotocol()* and *pthread\_mutexattr\_setprotocol()* functions, respectively,  
 34540 shall get and set the protocol attribute of a mutex attributes object pointed to by *attr* which was  
 34541 previously created by the function *pthread\_mutexattr\_init()*.

34542 The *protocol* attribute defines the protocol to be followed in utilizing mutexes. The value of  
 34543 *protocol* may be one of:

```
34544 PTHREAD_PRIO_NONE
34545 TPI PTHREAD_PRIO_INHERIT
34546 TPP PTHREAD_PRIO_PROTECT
34547
```

34548 which are defined in the **<pthread.h>** header.

34549 When a thread owns a mutex with the *PTHREAD\_PRIO\_NONE* *protocol* attribute, its priority  
 34550 and scheduling shall not be affected by its mutex ownership.

34551 TPI When a thread is blocking higher priority threads because of owning one or more mutexes with  
 34552 the *PTHREAD\_PRIO\_INHERIT* *protocol* attribute, it shall execute at the higher of its priority or  
 34553 the priority of the highest priority thread waiting on any of the mutexes owned by this thread  
 34554 and initialized with this protocol.

34555 TPP When a thread owns one or more mutexes initialized with the *PTHREAD\_PRIO\_PROTECT*  
 34556 protocol, it shall execute at the higher of its priority or the highest of the priority ceilings of all  
 34557 the mutexes owned by this thread and initialized with this attribute, regardless of whether other  
 34558 threads are blocked on any of these mutexes or not.

34559 While a thread is holding a mutex which has been initialized with the  
 34560 *PTHREAD\_PRIO\_INHERIT* or *PTHREAD\_PRIO\_PROTECT* protocol attributes, it shall not be  
 34561 subject to being moved to the tail of the scheduling queue at its priority in the event that its  
 34562 original priority is changed, such as by a call to *sched\_setparam()*. Likewise, when a thread  
 34563 unlocks a mutex that has been initialized with the *PTHREAD\_PRIO\_INHERIT* or  
 34564 *PTHREAD\_PRIO\_PROTECT* protocol attributes, it shall not be subject to being moved to the tail  
 34565 of the scheduling queue at its priority in the event that its original priority is changed.

34566 If a thread simultaneously owns several mutexes initialized with different protocols, it shall  
 34567 execute at the highest of the priorities that it would have obtained by each of these protocols.

34568 TPI When a thread makes a call to *pthread\_mutex\_lock()*, the mutex was initialized with the protocol  
 34569 attribute having the value *PTHREAD\_PRIO\_INHERIT*, when the calling thread is blocked  
 34570 because the mutex is owned by another thread, that owner thread shall inherit the priority level  
 34571 of the calling thread as long as it continues to own the mutex. The implementation shall update  
 34572 its execution priority to the maximum of its assigned priority and all its inherited priorities.  
 34573 Furthermore, if this owner thread itself becomes blocked on another mutex, the same priority

34574 inheritance effect shall be propagated to this other owner thread, in a recursive manner.

#### 34575 RETURN VALUE

34576 Upon successful completion, the *pthread\_mutexattr\_getprotocol()* and  
34577 *pthread\_mutexattr\_setprotocol()* functions shall return zero; otherwise, an error number shall be  
34578 returned to indicate the error.

#### 34579 ERRORS

34580 The *pthread\_mutexattr\_setprotocol()* function shall fail if:

34581 [ENOTSUP] The value specified by *protocol* is an unsupported value.

34582 The *pthread\_mutexattr\_getprotocol()* and *pthread\_mutexattr\_setprotocol()* functions may fail if:

34583 [EINVAL] The value specified by *attr* or *protocol* is invalid.

34584 [EPERM] The caller does not have the privilege to perform the operation.

34585 These functions shall not return an error code of [EINTR].

#### 34586 EXAMPLES

34587 None.

#### 34588 APPLICATION USAGE

34589 None.

#### 34590 RATIONALE

34591 None.

#### 34592 FUTURE DIRECTIONS

34593 None.

#### 34594 SEE ALSO

34595 *pthread\_cond\_destroy()*, *pthread\_create()*, *pthread\_mutex\_destroy()*, the Base Definitions volume of  
34596 IEEE Std 1003.1-2001, <pthread.h>

#### 34597 CHANGE HISTORY

34598 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34599 Marked as part of the Realtime Threads Feature Group.

#### 34600 Issue 6

34601 The *pthread\_mutexattr\_getprotocol()* and *pthread\_mutexattr\_setprotocol()* functions are marked as  
34602 part of the Threads option and either the Thread Priority Protection or Thread Priority  
34603 Inheritance options.

34604 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
34605 implementation does not support the Thread Priority Protection or Thread Priority Inheritance  
34606 options.

34607 The **restrict** keyword is added to the *pthread\_mutexattr\_getprotocol()* prototype for alignment  
34608 with the ISO/IEC 9899:1999 standard.

34609 **NAME**

34610 pthread\_mutexattr\_getpshared, pthread\_mutexattr\_setpshared — get and set the process-  
 34611 shared attribute

34612 **SYNOPSIS**

34613 THR TSH #include <pthread.h>

```
34614 int pthread_mutexattr_getpshared(const pthread_mutexattr_t *
34615 restrict attr, int *restrict pshared);
```

```
34616 int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
34617 int pshared);
```

34618

34619 **DESCRIPTION**

34620 The *pthread\_mutexattr\_getpshared()* function shall obtain the value of the *process-shared* attribute  
 34621 from the attributes object referenced by *attr*. The *pthread\_mutexattr\_setpshared()* function shall  
 34622 set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

34623 The *process-shared* attribute is set to PTHREAD\_PROCESS\_SHARED to permit a mutex to be  
 34624 operated upon by any thread that has access to the memory where the mutex is allocated, even if  
 34625 the mutex is allocated in memory that is shared by multiple processes. If the *process-shared*  
 34626 attribute is PTHREAD\_PROCESS\_PRIVATE, the mutex shall only be operated upon by threads  
 34627 created within the same process as the thread that initialized the mutex; if threads of differing  
 34628 processes attempt to operate on such a mutex, the behavior is undefined. The default value of  
 34629 the attribute shall be PTHREAD\_PROCESS\_PRIVATE.

34630 **RETURN VALUE**

34631 Upon successful completion, *pthread\_mutexattr\_setpshared()* shall return zero; otherwise, an error  
 34632 number shall be returned to indicate the error.

34633 Upon successful completion, *pthread\_mutexattr\_getpshared()* shall return zero and store the value  
 34634 of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter.  
 34635 Otherwise, an error number shall be returned to indicate the error.

34636 **ERRORS**

34637 The *pthread\_mutexattr\_getpshared()* and *pthread\_mutexattr\_setpshared()* functions may fail if:

34638 [EINVAL] The value specified by *attr* is invalid.

34639 The *pthread\_mutexattr\_setpshared()* function may fail if:

34640 [EINVAL] The new value specified for the attribute is outside the range of legal values  
 34641 for that attribute.

34642 These functions shall not return an error code of [EINTR].

34643 **EXAMPLES**

34644 None.

34645 **APPLICATION USAGE**

34646 None.

34647 **RATIONALE**

34648 None.

34649 **FUTURE DIRECTIONS**

34650 None.

34651 **SEE ALSO**

34652 *pthread\_cond\_destroy()*, *pthread\_create()*, *pthread\_mutex\_destroy()*, *pthread\_mutexattr\_destroy()*, the  
34653 Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

34654 **CHANGE HISTORY**

34655 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34656 **Issue 6**

34657 The *pthread\_mutexattr\_getpshared()* and *pthread\_mutexattr\_setpshared()* functions are marked as  
34658 part of the Threads and Thread Process-Shared Synchronization options.

34659 The **restrict** keyword is added to the *pthread\_mutexattr\_getpshared()* prototype for alignment  
34660 with the ISO/IEC 9899:1999 standard.

## 34661 NAME

34662 pthread\_mutexattr\_gettype, pthread\_mutexattr\_settype — get and set the mutex type attribute

## 34663 SYNOPSIS

```
34664 XSI #include <pthread.h>
```

```
34665 int pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict attr,
34666 int *restrict type);
```

```
34667 int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);
```

34668

## 34669 DESCRIPTION

34670 The *pthread\_mutexattr\_gettype()* and *pthread\_mutexattr\_settype()* functions, respectively, shall get  
34671 and set the mutex *type* attribute. This attribute is set in the *type* parameter to these functions. The  
34672 default value of the *type* attribute is PTHREAD\_MUTEX\_DEFAULT.

34673 The type of mutex is contained in the *type* attribute of the mutex attributes. Valid mutex types  
34674 include:

## 34675 PTHREAD\_MUTEX\_NORMAL

34676 This type of mutex does not detect deadlock. A thread attempting to relock this mutex  
34677 without first unlocking it shall deadlock. Attempting to unlock a mutex locked by a  
34678 different thread results in undefined behavior. Attempting to unlock an unlocked mutex  
34679 results in undefined behavior.

## 34680 PTHREAD\_MUTEX\_ERRORCHECK

34681 This type of mutex provides error checking. A thread attempting to relock this mutex  
34682 without first unlocking it shall return with an error. A thread attempting to unlock a mutex  
34683 which another thread has locked shall return with an error. A thread attempting to unlock  
34684 an unlocked mutex shall return with an error.

## 34685 PTHREAD\_MUTEX\_RECURSIVE

34686 A thread attempting to relock this mutex without first unlocking it shall succeed in locking  
34687 the mutex. The relocking deadlock which can occur with mutexes of type  
34688 PTHREAD\_MUTEX\_NORMAL cannot occur with this type of mutex. Multiple locks of this  
34689 mutex shall require the same number of unlocks to release the mutex before another thread  
34690 can acquire the mutex. A thread attempting to unlock a mutex which another thread has  
34691 locked shall return with an error. A thread attempting to unlock an unlocked mutex shall  
34692 return with an error.

## 34693 PTHREAD\_MUTEX\_DEFAULT

34694 Attempting to recursively lock a mutex of this type results in undefined behavior.  
34695 Attempting to unlock a mutex of this type which was not locked by the calling thread  
34696 results in undefined behavior. Attempting to unlock a mutex of this type which is not  
34697 locked results in undefined behavior. An implementation may map this mutex to one of the  
34698 other mutex types.

## 34699 RETURN VALUE

34700 Upon successful completion, the *pthread\_mutexattr\_gettype()* function shall return zero and store  
34701 the value of the *type* attribute of *attr* into the object referenced by the *type* parameter. Otherwise,  
34702 an error shall be returned to indicate the error.

34703 If successful, the *pthread\_mutexattr\_settype()* function shall return zero; otherwise, an error  
34704 number shall be returned to indicate the error.

34705 **ERRORS**

34706 The *pthread\_mutexattr\_settype()* function shall fail if:

34707 [EINVAL] The value *type* is invalid.

34708 The *pthread\_mutexattr\_gettype()* and *pthread\_mutexattr\_settype()* functions may fail if:

34709 [EINVAL] The value specified by *attr* is invalid.

34710 These functions shall not return an error code of [EINTR].

34711 **EXAMPLES**

34712 None.

34713 **APPLICATION USAGE**

34714 It is advised that an application should not use a PTHREAD\_MUTEX\_RECURSIVE mutex with  
34715 condition variables because the implicit unlock performed for a *pthread\_cond\_timedwait()* or  
34716 *pthread\_cond\_wait()* may not actually release the mutex (if it had been locked multiple times). If  
34717 this happens, no other thread can satisfy the condition of the predicate.

34718 **RATIONALE**

34719 None.

34720 **FUTURE DIRECTIONS**

34721 None.

34722 **SEE ALSO**

34723 *pthread\_cond\_timedwait()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>

34724 **CHANGE HISTORY**

34725 First released in Issue 5.

34726 **Issue 6**

34727 The Open Group Corrigendum U033/3 is applied. The SYNOPSIS for  
34728 *pthread\_mutexattr\_gettype()* is updated so that the first argument is of type **const**  
34729 **pthread\_mutexattr\_t\***.

34730 The **restrict** keyword is added to the *pthread\_mutexattr\_gettype()* prototype for alignment with  
34731 the ISO/IEC 9899:1999 standard.

34732 **NAME**

34733 pthread\_mutexattr\_init — initialize the mutex attributes object

34734 **SYNOPSIS**

34735 THR #include <pthread.h>

34736 int pthread\_mutexattr\_init(pthread\_mutexattr\_t \*attr);

34737

34738 **DESCRIPTION**

34739 Refer to *pthread\_mutexattr\_destroy()*.

34740 **NAME**

34741 pthread\_mutexattr\_setprioceiling — set the prioceiling attribute of the mutex attributes object  
34742 (**REALTIME THREADS**)

34743 **SYNOPSIS**

```
34744 THR TPP #include <pthread.h>
```

```
34745 int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
34746 int prioceiling);
```

34747

34748 **DESCRIPTION**

34749 Refer to *pthread\_mutexattr\_getprioceiling()*.

34750 **NAME**

34751 pthread\_mutexattr\_setprotocol — set the protocol attribute of the mutex attributes object  
34752 (**REALTIME THREADS**)

34753 **SYNOPSIS**

34754 THR #include <pthread.h>

34755 TPP|TPI int pthread\_mutexattr\_setprotocol(pthread\_mutexattr\_t \*attr,  
34756 int protocol);

34757

34758 **DESCRIPTION**

34759 Refer to *pthread\_mutexattr\_getprotocol()*.

34760 **NAME**

34761 pthread\_mutexattr\_setpshared — set the process-shared attribute

34762 **SYNOPSIS**

34763 THR TSH #include &lt;pthread.h&gt;

34764 int pthread\_mutexattr\_setpshared(pthread\_mutexattr\_t \*attr,  
34765 int pshared);

34766

34767 **DESCRIPTION**34768 Refer to *pthread\_mutexattr\_getpshared()*.

34769 **NAME**

34770 pthread\_mutexattr\_settype — set the mutex type attribute

34771 **SYNOPSIS**

34772 XSI #include <pthread.h>

34773 int pthread\_mutexattr\_settype(pthread\_mutexattr\_t \*attr, int type);

34774

34775 **DESCRIPTION**

34776 Refer to *pthread\_mutexattr\_gettype()*.

34777 **NAME**

34778 pthread\_once — dynamic package initialization

34779 **SYNOPSIS**

34780 THR #include &lt;pthread.h&gt;

```

34781 int pthread_once(pthread_once_t *once_control,
34782 void (*init_routine)(void));
34783 pthread_once_t once_control = PTHREAD_ONCE_INIT;
34784
```

34785 **DESCRIPTION**

34786 The first call to *pthread\_once()* by any thread in a process, with a given *once\_control*, shall call the  
 34787 *init\_routine* with no arguments. Subsequent calls of *pthread\_once()* with the same *once\_control*  
 34788 shall not call the *init\_routine*. On return from *pthread\_once()*, *init\_routine* shall have completed.  
 34789 The *once\_control* parameter shall determine whether the associated initialization routine has  
 34790 been called.

34791 The *pthread\_once()* function is not a cancellation point. However, if *init\_routine* is a cancellation  
 34792 point and is canceled, the effect on *once\_control* shall be as if *pthread\_once()* was never called.

34793 The constant PTHREAD\_ONCE\_INIT is defined in the <pthread.h> header.

34794 The behavior of *pthread\_once()* is undefined if *once\_control* has automatic storage duration or is  
 34795 not initialized by PTHREAD\_ONCE\_INIT.

34796 **RETURN VALUE**

34797 Upon successful completion, *pthread\_once()* shall return zero; otherwise, an error number shall  
 34798 be returned to indicate the error.

34799 **ERRORS**

34800 The *pthread\_once()* function may fail if:

34801 [EINVAL] If either *once\_control* or *init\_routine* is invalid.

34802 The *pthread\_once()* function shall not return an error code of [EINTR].

34803 **EXAMPLES**

34804 None.

34805 **APPLICATION USAGE**

34806 None.

34807 **RATIONALE**

34808 Some C libraries are designed for dynamic initialization. That is, the global initialization for the  
 34809 library is performed when the first procedure in the library is called. In a single-threaded  
 34810 program, this is normally implemented using a static variable whose value is checked on entry  
 34811 to a routine, as follows:

```

34812 static int random_is_initialized = 0;
34813 extern int initialize_random();

34814 int random_function()
34815 {
34816 if (random_is_initialized == 0) {
34817 initialize_random();
34818 random_is_initialized = 1;
34819 }
34820 ... /* Operations performed after initialization. */
34821 }
```

34822 To keep the same structure in a multi-threaded program, a new primitive is needed. Otherwise,  
34823 library initialization has to be accomplished by an explicit call to a library-exported initialization  
34824 function prior to any use of the library.

34825 For dynamic library initialization in a multi-threaded process, a simple initialization flag is not  
34826 sufficient; the flag needs to be protected against modification by multiple threads  
34827 simultaneously calling into the library. Protecting the flag requires the use of a mutex; however,  
34828 mutexes have to be initialized before they are used. Ensuring that the mutex is only initialized  
34829 once requires a recursive solution to this problem.

34830 The use of *pthread\_once()* not only supplies an implementation-guaranteed means of dynamic  
34831 initialization, it provides an aid to the reliable construction of multi-threaded and realtime  
34832 systems. The preceding example then becomes:

```
34833 #include <pthread.h>
34834 static pthread_once_t random_is_initialized = PTHREAD_ONCE_INIT;
34835 extern int initialize_random();

34836 int random_function()
34837 {
34838 (void) pthread_once(&random_is_initialized, initialize_random);
34839 ... /* Operations performed after initialization. */
34840 }
```

34841 Note that a **pthread\_once\_t** cannot be an array because some compilers do not accept the  
34842 construct **&<array\_name>**.

#### 34843 FUTURE DIRECTIONS

34844 None.

#### 34845 SEE ALSO

34846 The Base Definitions volume of IEEE Std 1003.1-2001, **<pthread.h>**

#### 34847 CHANGE HISTORY

34848 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 34849 Issue 6

34850 The *pthread\_once()* function is marked as part of the Threads option.

34851 The [EINVAL] error is added as a may fail case for if either argument is invalid.

34852 **NAME**

34853 pthread\_rwlock\_destroy, pthread\_rwlock\_init — destroy and initialize a read-write lock object

34854 **SYNOPSIS**

34855 THR #include &lt;pthread.h&gt;

```
34856 int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);
34857 int pthread_rwlock_init(pthread_rwlock_t *restrict rwlock,
34858 const pthread_rwlockattr_t *restrict attr);
34859
```

34860 **DESCRIPTION**

34861 The *pthread\_rwlock\_destroy()* function shall destroy the read-write lock object referenced by  
 34862 *rwlock* and release any resources used by the lock. The effect of subsequent use of the lock is  
 34863 undefined until the lock is reinitialized by another call to *pthread\_rwlock\_init()*. An  
 34864 implementation may cause *pthread\_rwlock\_destroy()* to set the object referenced by *rwlock* to an  
 34865 invalid value. Results are undefined if *pthread\_rwlock\_destroy()* is called when any thread holds  
 34866 *rwlock*. Attempting to destroy an uninitialized read-write lock results in undefined behavior.

34867 The *pthread\_rwlock\_init()* function shall allocate any resources required to use the read-write  
 34868 lock referenced by *rwlock* and initializes the lock to an unlocked state with attributes referenced  
 34869 by *attr*. If *attr* is NULL, the default read-write lock attributes shall be used; the effect is the same  
 34870 as passing the address of a default read-write lock attributes object. Once initialized, the lock can  
 34871 be used any number of times without being reinitialized. Results are undefined if  
 34872 *pthread\_rwlock\_init()* is called specifying an already initialized read-write lock. Results are  
 34873 undefined if a read-write lock is used without first being initialized.

34874 If the *pthread\_rwlock\_init()* function fails, *rwlock* shall not be initialized and the contents of *rwlock*  
 34875 are undefined.

34876 Only the object referenced by *rwlock* may be used for performing synchronization. The result of  
 34877 referring to copies of that object in calls to *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_rdlock()*,  
 34878 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_tryrdlock()*,  
 34879 *pthread\_rwlock\_trywrlock()*, *pthread\_rwlock\_unlock()*, or *pthread\_rwlock\_wrlock()* is undefined.

34880 **RETURN VALUE**

34881 If successful, the *pthread\_rwlock\_destroy()* and *pthread\_rwlock\_init()* functions shall return zero;  
 34882 otherwise, an error number shall be returned to indicate the error.

34883 The [EBUSY] and [EINVAL] error checks, if implemented, act as if they were performed  
 34884 immediately at the beginning of processing for the function and caused an error return prior to  
 34885 modifying the state of the read-write lock specified by *rwlock*.

34886 **ERRORS**34887 The *pthread\_rwlock\_destroy()* function may fail if:

34888 [EBUSY] The implementation has detected an attempt to destroy the object referenced  
 34889 by *rwlock* while it is locked.

34890 [EINVAL] The value specified by *rwlock* is invalid.

34891 The *pthread\_rwlock\_init()* function shall fail if:

34892 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize  
 34893 another read-write lock.

34894 [ENOMEM] Insufficient memory exists to initialize the read-write lock.

34895 [EPERM] The caller does not have the privilege to perform the operation.

- 34896 The *pthread\_rwlock\_init()* function may fail if:
- 34897 [EBUSY] The implementation has detected an attempt to reinitialize the object  
34898 referenced by *rwlock*, a previously initialized but not yet destroyed read-write  
34899 lock.
- 34900 [EINVAL] The value specified by *attr* is invalid.
- 34901 These functions shall not return an error code of [EINTR].
- 34902 **EXAMPLES**
- 34903 None.
- 34904 **APPLICATION USAGE**
- 34905 Applications using these and related read-write lock functions may be subject to priority  
34906 inversion, as discussed in the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.285,  
34907 Priority Inversion.
- 34908 **RATIONALE**
- 34909 None.
- 34910 **FUTURE DIRECTIONS**
- 34911 None.
- 34912 **SEE ALSO**
- 34913 *pthread\_rwlock\_rdlock()*, *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*,  
34914 *pthread\_rwlock\_tryrdlock()*, *pthread\_rwlock\_trywrlock()*, *pthread\_rwlock\_unlock()*,  
34915 *pthread\_rwlock\_wrlock()*, the Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>
- 34916 **CHANGE HISTORY**
- 34917 First released in Issue 5.
- 34918 **Issue 6**
- 34919 The following changes are made for alignment with IEEE Std 1003.1j-2000:
- 34920 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is  
34921 now part of the Threads option (previously it was part of the Read-Write Locks option in  
34922 IEEE Std 1003.1j-2000 and also part of the XSI extension). The initializer macro is also deleted  
34923 from the SYNOPSIS.
  - 34924 • The DESCRIPTION is updated as follows:
    - 34925 — It explicitly notes allocation of resources upon initialization of a read-write lock object.
    - 34926 — A paragraph is added specifying that copies of read-write lock objects may not be used.
  - 34927 • An [EINVAL] error is added to the ERRORS section for *pthread\_rwlock\_init()*, indicating that  
34928 the *rwlock* value is invalid.
  - 34929 • The SEE ALSO section is updated.
- 34930 The **restrict** keyword is added to the *pthread\_rwlock\_init()* prototype for alignment with the  
34931 ISO/IEC 9899:1999 standard.
- 34932 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/45 is applied, adding APPLICATION  
34933 USAGE relating to priority inversion.

34934 **NAME**

34935 pthread\_rwlock\_rdlock, pthread\_rwlock\_tryrdlock — lock a read-write lock object for reading

34936 **SYNOPSIS**

34937 THR #include &lt;pthread.h&gt;

34938 int pthread\_rwlock\_rdlock(pthread\_rwlock\_t \*rwlock);

34939 int pthread\_rwlock\_tryrdlock(pthread\_rwlock\_t \*rwlock);

34940

34941 **DESCRIPTION**

34942 The *pthread\_rwlock\_rdlock()* function shall apply a read lock to the read-write lock referenced by  
 34943 *rwlock*. The calling thread acquires the read lock if a writer does not hold the lock and there are  
 34944 no writers blocked on the lock.

34945 TPS If the Thread Execution Scheduling option is supported, and the threads involved in the lock are  
 34946 executing with the scheduling policies SCHED\_FIFO or SCHED\_RR, the calling thread shall not  
 34947 acquire the lock if a writer holds the lock or if writers of higher or equal priority are blocked on  
 34948 the lock; otherwise, the calling thread shall acquire the lock.

34949 TPS TSP If the Threads Execution Scheduling option is supported, and the threads involved in the lock  
 34950 are executing with the SCHED\_SPORADIC scheduling policy, the calling thread shall not  
 34951 acquire the lock if a writer holds the lock or if writers of higher or equal priority are blocked on  
 34952 the lock; otherwise, the calling thread shall acquire the lock.

34953 If the Thread Execution Scheduling option is not supported, it is implementation-defined  
 34954 whether the calling thread acquires the lock when a writer does not hold the lock and there are  
 34955 writers blocked on the lock. If a writer holds the lock, the calling thread shall not acquire the  
 34956 read lock. If the read lock is not acquired, the calling thread shall block until it can acquire the  
 34957 lock. The calling thread may deadlock if at the time the call is made it holds a write lock.

34958 A thread may hold multiple concurrent read locks on *rwlock* (that is, successfully call the  
 34959 *pthread\_rwlock\_rdlock()* function *n* times). If so, the application shall ensure that the thread  
 34960 performs matching unlocks (that is, it calls the *pthread\_rwlock\_unlock()* function *n* times).

34961 The maximum number of simultaneous read locks that an implementation guarantees can be  
 34962 applied to a read-write lock shall be implementation-defined. The *pthread\_rwlock\_rdlock()*  
 34963 function may fail if this maximum would be exceeded.

34964 The *pthread\_rwlock\_tryrdlock()* function shall apply a read lock as in the *pthread\_rwlock\_rdlock()*  
 34965 function, with the exception that the function shall fail if the equivalent *pthread\_rwlock\_rdlock()*  
 34966 call would have blocked the calling thread. In no case shall the *pthread\_rwlock\_tryrdlock()*  
 34967 function ever block; it always either acquires the lock or fails and returns immediately.

34968 Results are undefined if any of these functions are called with an uninitialized read-write lock.

34969 If a signal is delivered to a thread waiting for a read-write lock for reading, upon return from the  
 34970 signal handler the thread resumes waiting for the read-write lock for reading as if it was not  
 34971 interrupted.

34972 **RETURN VALUE**

34973 If successful, the *pthread\_rwlock\_rdlock()* function shall return zero; otherwise, an error number  
 34974 shall be returned to indicate the error.

34975 The *pthread\_rwlock\_tryrdlock()* function shall return zero if the lock for reading on the read-write  
 34976 lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to  
 34977 indicate the error.

34978 **ERRORS**

34979 The *pthread\_rwlock\_tryrdlock()* function shall fail if:

34980 [EBUSY] The read-write lock could not be acquired for reading because a writer holds  
34981 the lock or a writer with the appropriate priority was blocked on it.

34982 The *pthread\_rwlock\_rdlock()* and *pthread\_rwlock\_tryrdlock()* functions may fail if:

34983 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock  
34984 object.

34985 [EAGAIN] The read lock could not be acquired because the maximum number of read  
34986 locks for *rwlock* has been exceeded.

34987 The *pthread\_rwlock\_rdlock()* function may fail if:

34988 [EDEADLK] The current thread already owns the read-write lock for writing.

34989 These functions shall not return an error code of [EINTR].

34990 **EXAMPLES**

34991 None.

34992 **APPLICATION USAGE**

34993 Applications using these functions may be subject to priority inversion, as discussed in the Base  
34994 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

34995 **RATIONALE**

34996 None.

34997 **FUTURE DIRECTIONS**

34998 None.

34999 **SEE ALSO**

35000 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*,  
35001 *pthread\_rwlock\_trywrlock()*, *pthread\_rwlock\_unlock()*, *pthread\_rwlock\_wrlock()*, the Base Definitions  
35002 volume of IEEE Std 1003.1-2001, <pthread.h>

35003 **CHANGE HISTORY**

35004 First released in Issue 5.

35005 **Issue 6**

35006 The following changes are made for alignment with IEEE Std 1003.1j-2000:

35007 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is  
35008 now part of the Threads option (previously it was part of the Read-Write Locks option in  
35009 IEEE Std 1003.1j-2000 and also part of the XSI extension).

35010 • The DESCRIPTION is updated as follows:

35011 — Conditions under which writers have precedence over readers are specified.

35012 — Failure of *pthread\_rwlock\_tryrdlock()* is clarified.

35013 — A paragraph on the maximum number of read locks is added.

35014 • In the ERRORS sections, [EBUSY] is modified to take into account write priority, and  
35015 [EDEADLK] is deleted as a *pthread\_rwlock\_tryrdlock()* error.

35016 • The SEE ALSO section is updated.

35017 **NAME**

35018 pthread\_rwlock\_timedrdlock — lock a read-write lock for reading

35019 **SYNOPSIS**

35020 THR TMO #include &lt;pthread.h&gt;

35021 #include &lt;time.h&gt;

```
35022 int pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict rwlock,
35023 const struct timespec *restrict abs_timeout);
35024
```

35025 **DESCRIPTION**

35026 The *pthread\_rwlock\_timedrdlock()* function shall apply a read lock to the read-write lock  
 35027 referenced by *rwlock* as in the *pthread\_rwlock\_rdlock()* function. However, if the lock cannot be  
 35028 acquired without waiting for other threads to unlock the lock, this wait shall be terminated  
 35029 when the specified timeout expires. The timeout shall expire when the absolute time specified  
 35030 by *abs\_timeout* passes, as measured by the clock on which timeouts are based (that is, when the  
 35031 value of that clock equals or exceeds *abs\_timeout*), or if the absolute time specified by *abs\_timeout*  
 35032 has already been passed at the time of the call.

35033 TMR If the Timers option is supported, the timeout shall be based on the `CLOCK_REALTIME` clock. If  
 35034 the Timers option is not supported, the timeout shall be based on the system clock as returned  
 35035 by the *time()* function. The resolution of the timeout shall be the resolution of the clock on which  
 35036 it is based. The **timespec** data type is defined in the `<time.h>` header. Under no circumstances  
 35037 shall the function fail with a timeout if the lock can be acquired immediately. The validity of the  
 35038 *abs\_timeout* parameter need not be checked if the lock can be immediately acquired.

35039 If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-  
 35040 write lock via a call to *pthread\_rwlock\_timedrdlock()*, upon return from the signal handler the  
 35041 thread shall resume waiting for the lock as if it was not interrupted.

35042 The calling thread may deadlock if at the time the call is made it holds a write lock on *rwlock*.  
 35043 The results are undefined if this function is called with an uninitialized read-write lock.

35044 **RETURN VALUE**

35045 The *pthread\_rwlock\_timedrdlock()* function shall return zero if the lock for reading on the read-  
 35046 write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned  
 35047 to indicate the error.

35048 **ERRORS**35049 The *pthread\_rwlock\_timedrdlock()* function shall fail if:

35050 [ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

35051 The *pthread\_rwlock\_timedrdlock()* function may fail if:35052 [EAGAIN] The read lock could not be acquired because the maximum number of read  
35053 locks for lock would be exceeded.35054 [EDEADLK] The calling thread already holds a write lock on *rwlock*.35055 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock  
35056 object, or the *abs\_timeout* nanosecond value is less than zero or greater than or  
35057 equal to 1 000 million.

35058 This function shall not return an error code of [EINTR].

35059 **EXAMPLES**

35060 None.

35061 **APPLICATION USAGE**

35062 Applications using this function may be subject to priority inversion, as discussed in the Base  
35063 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

35064 The *pthread\_rwlock\_timedrdlock()* function is part of the Threads and Timeouts options and need  
35065 not be provided on all implementations.

35066 **RATIONALE**

35067 None.

35068 **FUTURE DIRECTIONS**

35069 None.

35070 **SEE ALSO**

35071 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_rdlock()*, *pthread\_rwlock\_timedwrlock()*,  
35072 *pthread\_rwlock\_tryrdlock()*, *pthread\_rwlock\_trywrlock()*, *pthread\_rwlock\_unlock()*,  
35073 *pthread\_rwlock\_wrlock()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>,  
35074 <**time.h**>

35075 **CHANGE HISTORY**

35076 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

35077 **NAME**

35078 pthread\_rwlock\_timedwrlock — lock a read-write lock for writing

35079 **SYNOPSIS**

35080 THR TMO #include &lt;pthread.h&gt;

35081 #include &lt;time.h&gt;

35082 int pthread\_rwlock\_timedwrlock(pthread\_rwlock\_t \*restrict *rwlock*,35083 const struct timespec \*restrict *abs\_timeout*);

35084

35085 **DESCRIPTION**

35086 The *pthread\_rwlock\_timedwrlock()* function shall apply a write lock to the read-write lock  
 35087 referenced by *rwlock* as in the *pthread\_rwlock\_wrlock()* function. However, if the lock cannot be  
 35088 acquired without waiting for other threads to unlock the lock, this wait shall be terminated  
 35089 when the specified timeout expires. The timeout shall expire when the absolute time specified  
 35090 by *abs\_timeout* passes, as measured by the clock on which timeouts are based (that is, when the  
 35091 value of that clock equals or exceeds *abs\_timeout*), or if the absolute time specified by *abs\_timeout*  
 35092 has already been passed at the time of the call.

35093 TMR If the Timers option is supported, the timeout shall be based on the `CLOCK_REALTIME` clock. If  
 35094 the Timers option is not supported, the timeout shall be based on the system clock as returned  
 35095 by the *time()* function. The resolution of the timeout shall be the resolution of the clock on which  
 35096 it is based. The `timespec` data type is defined in the `<time.h>` header. Under no circumstances  
 35097 shall the function fail with a timeout if the lock can be acquired immediately. The validity of the  
 35098 *abs\_timeout* parameter need not be checked if the lock can be immediately acquired.

35099 If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-  
 35100 write lock via a call to *pthread\_rwlock\_timedwrlock()*, upon return from the signal handler the  
 35101 thread shall resume waiting for the lock as if it was not interrupted.

35102 The calling thread may deadlock if at the time the call is made it holds the read-write lock. The  
 35103 results are undefined if this function is called with an uninitialized read-write lock.

35104 **RETURN VALUE**

35105 The *pthread\_rwlock\_timedwrlock()* function shall return zero if the lock for writing on the read-  
 35106 write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned  
 35107 to indicate the error.

35108 **ERRORS**35109 The *pthread\_rwlock\_timedwrlock()* function shall fail if:

35110 [ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

35111 The *pthread\_rwlock\_timedwrlock()* function may fail if:35112 [EDEADLK] The calling thread already holds the *rwlock*.

35113 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock  
 35114 object, or the *abs\_timeout* nanosecond value is less than zero or greater than or  
 35115 equal to 1 000 million.

35116 This function shall not return an error code of [EINTR].

## 35117 EXAMPLES

35118 None.

## 35119 APPLICATION USAGE

35120 Applications using this function may be subject to priority inversion, as discussed in the Base  
35121 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

35122 The *pthread\_rwlock\_timedwrlock()* function is part of the Threads and Timeouts options and need  
35123 not be provided on all implementations.

## 35124 RATIONALE

35125 None.

## 35126 FUTURE DIRECTIONS

35127 None.

## 35128 SEE ALSO

35129 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_rdlock()*, *pthread\_rwlock\_timedrdlock()*,  
35130 *pthread\_rwlock\_tryrdlock()*, *pthread\_rwlock\_trywrlock()*, *pthread\_rwlock\_unlock()*,  
35131 *pthread\_rwlock\_wrlock()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**pthread.h**>,  
35132 <**time.h**>

## 35133 CHANGE HISTORY

35134 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

35135 **NAME**

35136 pthread\_rwlock\_tryrdlock — lock a read-write lock object for reading

35137 **SYNOPSIS**

35138 THR #include <pthread.h>

35139 int pthread\_rwlock\_tryrdlock(pthread\_rwlock\_t \*rwlock);

35140

35141 **DESCRIPTION**

35142 Refer to *pthread\_rwlock\_rdlock()*.

35143 **NAME**

35144 pthread\_rwlock\_trywrlock, pthread\_rwlock\_wrlock — lock a read-write lock object for writing

35145 **SYNOPSIS**

```
35146 THR #include <pthread.h>
```

```
35147 int pthread_rwlock_trywrlock(pthread_rwlock_t *rwlock);
```

```
35148 int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);
```

35149

35150 **DESCRIPTION**

35151 The *pthread\_rwlock\_trywrlock()* function shall apply a write lock like the *pthread\_rwlock\_wrlock()*  
35152 function, with the exception that the function shall fail if any thread currently holds *rwlock* (for  
35153 reading or writing).

35154 The *pthread\_rwlock\_wrlock()* function shall apply a write lock to the read-write lock referenced  
35155 by *rwlock*. The calling thread acquires the write lock if no other thread (reader or writer) holds  
35156 the read-write lock *rwlock*. Otherwise, the thread shall block until it can acquire the lock. The  
35157 calling thread may deadlock if at the time the call is made it holds the read-write lock (whether a  
35158 read or write lock).

35159 Implementations may favor writers over readers to avoid writer starvation.

35160 Results are undefined if any of these functions are called with an uninitialized read-write lock.

35161 If a signal is delivered to a thread waiting for a read-write lock for writing, upon return from the  
35162 signal handler the thread resumes waiting for the read-write lock for writing as if it was not  
35163 interrupted.

35164 **RETURN VALUE**

35165 The *pthread\_rwlock\_trywrlock()* function shall return zero if the lock for writing on the read-write  
35166 lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to  
35167 indicate the error.

35168 If successful, the *pthread\_rwlock\_wrlock()* function shall return zero; otherwise, an error number  
35169 shall be returned to indicate the error.

35170 **ERRORS**

35171 The *pthread\_rwlock\_trywrlock()* function shall fail if:

35172 [EBUSY] The read-write lock could not be acquired for writing because it was already  
35173 locked for reading or writing.

35174 The *pthread\_rwlock\_trywrlock()* and *pthread\_rwlock\_wrlock()* functions may fail if:

35175 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock  
35176 object.

35177 The *pthread\_rwlock\_wrlock()* function may fail if:

35178 [EDEADLK] The current thread already owns the read-write lock for writing or reading.

35179 These functions shall not return an error code of [EINTR].

35180 **EXAMPLES**

35181 None.

35182 **APPLICATION USAGE**

35183 Applications using these functions may be subject to priority inversion, as discussed in the Base  
35184 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

35185 **RATIONALE**

35186 None.

35187 **FUTURE DIRECTIONS**

35188 None.

35189 **SEE ALSO**

35190 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_rdlock()*, *pthread\_rwlock\_timedrdlock()*,  
35191 *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_tryrdlock()*, *pthread\_rwlock\_unlock()*, the Base  
35192 Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

35193 **CHANGE HISTORY**

35194 First released in Issue 5.

35195 **Issue 6**

35196 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 35197 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is  
35198 now part of the Threads option (previously it was part of the Read-Write Locks option in  
35199 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 35200 • The [EDEADLK] error is deleted as a *pthread\_rwlock\_trywrlock()* error.
- 35201 • The SEE ALSO section is updated.

35202 **NAME**

35203 pthread\_rwlock\_unlock — unlock a read-write lock object

35204 **SYNOPSIS**

35205 THR #include &lt;pthread.h&gt;

35206 int pthread\_rwlock\_unlock(pthread\_rwlock\_t \*rwlock);

35207

35208 **DESCRIPTION**

35209 The *pthread\_rwlock\_unlock()* function shall release a lock held on the read-write lock object  
35210 referenced by *rwlock*. Results are undefined if the read-write lock *rwlock* is not held by the  
35211 calling thread.

35212 If this function is called to release a read lock from the read-write lock object and there are other  
35213 read locks currently held on this read-write lock object, the read-write lock object remains in the  
35214 read locked state. If this function releases the last read lock for this read-write lock object, the  
35215 read-write lock object shall be put in the unlocked state with no owners.

35216 If this function is called to release a write lock for this read-write lock object, the read-write lock  
35217 object shall be put in the unlocked state.

35218 If there are threads blocked on the lock when it becomes available, the scheduling policy shall  
35219 TPS determine which thread(s) shall acquire the lock. If the Thread Execution Scheduling option is  
35220 supported, when threads executing with the scheduling policies SCHED\_FIFO, SCHED\_RR, or  
35221 SCHED\_SPORADIC are waiting on the lock, they shall acquire the lock in priority order when  
35222 the lock becomes available. For equal priority threads, write locks shall take precedence over  
35223 read locks. If the Thread Execution Scheduling option is not supported, it is implementation-  
35224 defined whether write locks take precedence over read locks.

35225 Results are undefined if any of these functions are called with an uninitialized read-write lock.

35226 **RETURN VALUE**

35227 If successful, the *pthread\_rwlock\_unlock()* function shall return zero; otherwise, an error number  
35228 shall be returned to indicate the error.

35229 **ERRORS**

35230 The *pthread\_rwlock\_unlock()* function may fail if:

35231 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock  
35232 object.

35233 [EPERM] The current thread does not hold a lock on the read-write lock.

35234 The *pthread\_rwlock\_unlock()* function shall not return an error code of [EINTR].

35235 **EXAMPLES**

35236 None.

35237 **APPLICATION USAGE**

35238 None.

35239 **RATIONALE**

35240 None.

35241 **FUTURE DIRECTIONS**

35242 None.

35243 **SEE ALSO**

35244 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_rdlock()*, *pthread\_rwlock\_timedrdlock()*,  
35245 *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_tryrdlock()*, *pthread\_rwlock\_trywrlock()*,  
35246 *pthread\_rwlock\_wrlock()*, the Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

35247 **CHANGE HISTORY**

35248 First released in Issue 5.

35249 **Issue 6**

35250 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 35251 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is  
35252 now part of the Threads option (previously it was part of the Read-Write Locks option in  
35253 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 35254 • The DESCRIPTION is updated as follows:
  - 35255 — The conditions under which writers have precedence over readers are specified.
  - 35256 — The concept of read-write lock owner is deleted.
- 35257 • The SEE ALSO section is updated.

35258 **NAME**

35259 pthread\_rwlock\_wrlock — lock a read-write lock object for writing

35260 **SYNOPSIS**

35261 THR #include <pthread.h>

35262 int pthread\_rwlock\_wrlock(pthread\_rwlock\_t \*rwlock);

35263

35264 **DESCRIPTION**

35265 Refer to *pthread\_rwlock\_trywrlock()*.

35266 **NAME**

35267 pthread\_rwlockattr\_destroy, pthread\_rwlockattr\_init — destroy and initialize the read-write  
 35268 lock attributes object

35269 **SYNOPSIS**

```
35270 THR #include <pthread.h>
```

```
35271 int pthread_rwlockattr_destroy(pthread_rwlockattr_t *attr);
```

```
35272 int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);
```

```
35273
```

35274 **DESCRIPTION**

35275 The *pthread\_rwlockattr\_destroy()* function shall destroy a read-write lock attributes object. A  
 35276 destroyed *attr* attributes object can be reinitialized using *pthread\_rwlockattr\_init()*; the results of  
 35277 otherwise referencing the object after it has been destroyed are undefined. An implementation  
 35278 may cause *pthread\_rwlockattr\_destroy()* to set the object referenced by *attr* to an invalid value.

35279 The *pthread\_rwlockattr\_init()* function shall initialize a read-write lock attributes object *attr* with  
 35280 the default value for all of the attributes defined by the implementation.

35281 Results are undefined if *pthread\_rwlockattr\_init()* is called specifying an already initialized *attr*  
 35282 attributes object.

35283 After a read-write lock attributes object has been used to initialize one or more read-write locks,  
 35284 any function affecting the attributes object (including destruction) shall not affect any previously  
 35285 initialized read-write locks.

35286 **RETURN VALUE**

35287 If successful, the *pthread\_rwlockattr\_destroy()* and *pthread\_rwlockattr\_init()* functions shall return  
 35288 zero; otherwise, an error number shall be returned to indicate the error.

35289 **ERRORS**

35290 The *pthread\_rwlockattr\_destroy()* function may fail if:

35291 [EINVAL] The value specified by *attr* is invalid.

35292 The *pthread\_rwlockattr\_init()* function shall fail if:

35293 [ENOMEM] Insufficient memory exists to initialize the read-write lock attributes object.

35294 These functions shall not return an error code of [EINTR].

35295 **EXAMPLES**

35296 None.

35297 **APPLICATION USAGE**

35298 None.

35299 **RATIONALE**

35300 None.

35301 **FUTURE DIRECTIONS**

35302 None.

35303 **SEE ALSO**

35304 *pthread\_rwlock\_destroy()*, *pthread\_rwlockattr\_getpshared()*, *pthread\_rwlockattr\_setpshared()*, the  
 35305 Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

## 35306 CHANGE HISTORY

35307 First released in Issue 5.

## 35308 Issue 6

35309 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 35310 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is
- 35311 now part of the Threads option (previously it was part of the Read-Write Locks option in
- 35312 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 35313 • The SEE ALSO section is updated.

35314 **NAME**

35315 pthread\_rwlockattr\_getpshared, pthread\_rwlockattr\_setpshared — get and set the process-  
 35316 shared attribute of the read-write lock attributes object

35317 **SYNOPSIS**

35318 THR TSH #include <pthread.h>

```
35319 int pthread_rwlockattr_getpshared(const pthread_rwlockattr_t *
35320 restrict attr, int *restrict pshared);
35321 int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
35322 int pshared);
35323
```

35324 **DESCRIPTION**

35325 The *pthread\_rwlockattr\_getpshared()* function shall obtain the value of the *process-shared* attribute  
 35326 from the initialized attributes object referenced by *attr*. The *pthread\_rwlockattr\_setpshared()*  
 35327 function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

35328 The *process-shared* attribute shall be set to PTHREAD\_PROCESS\_SHARED to permit a read-  
 35329 write lock to be operated upon by any thread that has access to the memory where the read-  
 35330 write lock is allocated, even if the read-write lock is allocated in memory that is shared by  
 35331 multiple processes. If the *process-shared* attribute is PTHREAD\_PROCESS\_PRIVATE, the read-  
 35332 write lock shall only be operated upon by threads created within the same process as the thread  
 35333 that initialized the read-write lock; if threads of differing processes attempt to operate on such a  
 35334 read-write lock, the behavior is undefined. The default value of the *process-shared* attribute shall  
 35335 be PTHREAD\_PROCESS\_PRIVATE.

35336 Additional attributes, their default values, and the names of the associated functions to get and  
 35337 set those attribute values are implementation-defined.

35338 **RETURN VALUE**

35339 Upon successful completion, the *pthread\_rwlockattr\_getpshared()* function shall return zero and  
 35340 store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared*  
 35341 parameter. Otherwise, an error number shall be returned to indicate the error.

35342 If successful, the *pthread\_rwlockattr\_setpshared()* function shall return zero; otherwise, an error  
 35343 number shall be returned to indicate the error.

35344 **ERRORS**

35345 The *pthread\_rwlockattr\_getpshared()* and *pthread\_rwlockattr\_setpshared()* functions may fail if:

35346 [EINVAL] The value specified by *attr* is invalid.

35347 The *pthread\_rwlockattr\_setpshared()* function may fail if:

35348 [EINVAL] The new value specified for the attribute is outside the range of legal values  
 35349 for that attribute.

35350 These functions shall not return an error code of [EINTR].

35351 **EXAMPLES**

35352 None.

35353 **APPLICATION USAGE**

35354 None.

35355 **RATIONALE**

35356 None.

35357 **FUTURE DIRECTIONS**

35358 None.

35359 **SEE ALSO**

35360 *pthread\_rwlock\_destroy()*, *pthread\_rwlockattr\_destroy()*, *pthread\_rwlockattr\_init()*, the Base  
35361 Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

35362 **CHANGE HISTORY**

35363 First released in Issue 5.

35364 **Issue 6**

35365 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 35366 • The margin code in the SYNOPSIS is changed to THR TSH to indicate that the functionality  
35367 is now part of the Threads option (previously it was part of the Read-Write Locks option in  
35368 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 35369 • The DESCRIPTION notes that additional attributes are implementation-defined.
- 35370 • The SEE ALSO section is updated.

35371 The **restrict** keyword is added to the *pthread\_rwlockattr\_getpshared()* prototype for alignment  
35372 with the ISO/IEC 9899:1999 standard.

35373 **NAME**

35374 pthread\_rwlockattr\_init — initialize the read-write lock attributes object

35375 **SYNOPSIS**

35376 THR #include <pthread.h>

35377 int pthread\_rwlockattr\_init(pthread\_rwlockattr\_t \*attr);

35378

35379 **DESCRIPTION**

35380 Refer to *pthread\_rwlockattr\_destroy()*.

35381 **NAME**

35382 pthread\_rwlockattr\_setpshared — set the process-shared attribute of the read-write lock  
35383 attributes object

35384 **SYNOPSIS**

```
35385 THR TSH #include <pthread.h>
```

```
35386 int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
35387 int pshared);
```

35388

35389 **DESCRIPTION**

35390 Refer to *pthread\_rwlockattr\_getpshared()*.

35391 **NAME**

35392 pthread\_self — get the calling thread ID

35393 **SYNOPSIS**

35394 THR #include <pthread.h>

35395 pthread\_t pthread\_self(void);

35396

35397 **DESCRIPTION**

35398 The *pthread\_self()* function shall return the thread ID of the calling thread.

35399 **RETURN VALUE**

35400 Refer to the DESCRIPTION.

35401 **ERRORS**

35402 No errors are defined.

35403 The *pthread\_self()* function shall not return an error code of [EINTR].

35404 **EXAMPLES**

35405 None.

35406 **APPLICATION USAGE**

35407 None.

35408 **RATIONALE**

35409 The *pthread\_self()* function provides a capability similar to the *getpid()* function for processes  
35410 and the rationale is the same: the creation call does not provide the thread ID to the created  
35411 thread.

35412 **FUTURE DIRECTIONS**

35413 None.

35414 **SEE ALSO**

35415 *pthread\_create()*, *pthread\_equal()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
35416 <pthread.h>

35417 **CHANGE HISTORY**

35418 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

35419 **Issue 6**

35420 The *pthread\_self()* function is marked as part of the Threads option.

35421 **NAME**

35422 pthread\_setcancelstate, pthread\_setcanceltype, pthread\_testcancel — set cancelability state

35423 **SYNOPSIS**

35424 THR #include &lt;pthread.h&gt;

35425 int pthread\_setcancelstate(int *state*, int \**oldstate*);35426 int pthread\_setcanceltype(int *type*, int \**oldtype*);

35427 void pthread\_testcancel(void);

35428

35429 **DESCRIPTION**

35430 The *pthread\_setcancelstate()* function shall atomically both set the calling thread's cancelability  
 35431 state to the indicated *state* and return the previous cancelability state at the location referenced  
 35432 by *oldstate*. Legal values for *state* are PTHREAD\_CANCEL\_ENABLE and  
 35433 PTHREAD\_CANCEL\_DISABLE.

35434 The *pthread\_setcanceltype()* function shall atomically both set the calling thread's cancelability  
 35435 type to the indicated *type* and return the previous cancelability type at the location referenced by  
 35436 *oldtype*. Legal values for *type* are PTHREAD\_CANCEL\_DEFERRED and  
 35437 PTHREAD\_CANCEL\_ASYNCHRONOUS.

35438 The cancelability state and type of any newly created threads, including the thread in which  
 35439 *main()* was first invoked, shall be PTHREAD\_CANCEL\_ENABLE and  
 35440 PTHREAD\_CANCEL\_DEFERRED respectively.

35441 The *pthread\_testcancel()* function shall create a cancellation point in the calling thread. The  
 35442 *pthread\_testcancel()* function shall have no effect if cancelability is disabled.

35443 **RETURN VALUE**

35444 If successful, the *pthread\_setcancelstate()* and *pthread\_setcanceltype()* functions shall return zero;  
 35445 otherwise, an error number shall be returned to indicate the error.

35446 **ERRORS**35447 The *pthread\_setcancelstate()* function may fail if:

35448 [EINVAL] The specified state is not PTHREAD\_CANCEL\_ENABLE or  
 35449 PTHREAD\_CANCEL\_DISABLE.

35450 The *pthread\_setcanceltype()* function may fail if:

35451 [EINVAL] The specified type is not PTHREAD\_CANCEL\_DEFERRED or  
 35452 PTHREAD\_CANCEL\_ASYNCHRONOUS.

35453 These functions shall not return an error code of [EINTR].

35454 **EXAMPLES**

35455 None.

35456 **APPLICATION USAGE**

35457 None.

35458 **RATIONALE**

35459 The *pthread\_setcancelstate()* and *pthread\_setcanceltype()* functions control the points at which a  
 35460 thread may be asynchronously canceled. For cancellation control to be usable in modular  
 35461 fashion, some rules need to be followed.

35462 An object can be considered to be a generalization of a procedure. It is a set of procedures and  
 35463 global variables written as a unit and called by clients not known by the object. Objects may  
 35464 depend on other objects.

35465 First, cancelability should only be disabled on entry to an object, never explicitly enabled. On  
35466 exit from an object, the cancelability state should always be restored to its value on entry to the  
35467 object.

35468 This follows from a modularity argument: if the client of an object (or the client of an object that  
35469 uses that object) has disabled cancelability, it is because the client does not want to be concerned  
35470 about cleaning up if the thread is canceled while executing some sequence of actions. If an object  
35471 is called in such a state and it enables cancelability and a cancellation request is pending for that  
35472 thread, then the thread is canceled, contrary to the wish of the client that disabled.

35473 Second, the cancelability type may be explicitly set to either *deferred* or *asynchronous* upon entry  
35474 to an object. But as with the cancelability state, on exit from an object the cancelability type  
35475 should always be restored to its value on entry to the object.

35476 Finally, only functions that are cancel-safe may be called from a thread that is asynchronously  
35477 cancelable.

#### 35478 **FUTURE DIRECTIONS**

35479 None.

#### 35480 **SEE ALSO**

35481 *pthread\_cancel()*, the Base Definitions volume of IEEE Std 1003.1-2001, <pthread.h>

#### 35482 **CHANGE HISTORY**

35483 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 35484 **Issue 6**

35485 The *pthread\_setcancelstate()*, *pthread\_setcanceltype()*, and *pthread\_testcancel()* functions are marked  
35486 as part of the Threads option.

35487 **NAME**

35488 pthread\_setconcurrency — set the level of concurrency

35489 **SYNOPSIS**

35490 XSI `#include <pthread.h>`

35491 `int pthread_setconcurrency(int new_level);`

35492

35493 **DESCRIPTION**

35494 Refer to *pthread\_getconcurrency()*.

35495 **NAME**

35496 pthread\_setschedparam — dynamic thread scheduling parameters access (**REALTIME**  
35497 **THREADS**)

35498 **SYNOPSIS**

```
35499 THR TPS #include <pthread.h>
```

```
35500 int pthread_setschedparam(pthread_t thread, int policy,
35501 const struct sched_param *param);
```

35502

35503 **DESCRIPTION**

35504 Refer to *pthread\_getschedparam()*.

35505 **NAME**

35506 pthread\_setschedprio — dynamic thread scheduling parameters access (**REALTIME**  
35507 **THREADS**)

35508 **SYNOPSIS**

```
35509 THR TPS #include <pthread.h>
```

```
35510 int pthread_setschedprio(pthread_t thread, int prio);
```

35511

35512 **DESCRIPTION**

35513 The *pthread\_setschedprio()* function shall set the scheduling priority for the thread whose thread  
35514 ID is given by *thread* to the value given by *prio*. See **Scheduling Policies** (on page 44) for a  
35515 description on how this function call affects the ordering of the thread in the thread list for its  
35516 new priority.

35517 If the *pthread\_setschedprio()* function fails, the scheduling priority of the target thread shall not be  
35518 changed.

35519 **RETURN VALUE**

35520 If successful, the *pthread\_setschedprio()* function shall return zero; otherwise, an error number  
35521 shall be returned to indicate the error.

35522 **ERRORS**

35523 The *pthread\_setschedprio()* function may fail if:

35524 [EINVAL] The value of *prio* is invalid for the scheduling policy of the specified thread.

35525 [ENOTSUP] An attempt was made to set the priority to an unsupported value.

35526 [EPERM] The caller does not have the appropriate permission to set the scheduling  
35527 policy of the specified thread.

35528 [EPERM] The implementation does not allow the application to modify the priority to  
35529 the value specified.

35530 [ESRCH] The value specified by *thread* does not refer to an existing thread.

35531 The *pthread\_setschedprio()* function shall not return an error code of [EINTR].

35532 **EXAMPLES**

35533 None.

35534 **APPLICATION USAGE**

35535 None.

35536 **RATIONALE**

35537 The *pthread\_setschedprio()* function provides a way for an application to temporarily raise its  
35538 priority and then lower it again, without having the undesired side effect of yielding to other  
35539 threads of the same priority. This is necessary if the application is to implement its own  
35540 strategies for bounding priority inversion, such as priority inheritance or priority ceilings. This  
35541 capability is especially important if the implementation does not support the Thread Priority  
35542 Protection or Thread Priority Inheritance options, but even if those options are supported it is  
35543 needed if the application is to bound priority inheritance for other resources, such as  
35544 semaphores.

35545 The standard developers considered that while it might be preferable conceptually to solve this  
35546 problem by modifying the specification of *pthread\_setschedparam()*, it was too late to make such a  
35547 change, as there may be implementations that would need to be changed. Therefore, this new  
35548 function was introduced.

35549 **FUTURE DIRECTIONS**

35550 None.

35551 **SEE ALSO**

35552 **Scheduling Policies** (on page 44), *pthread\_getschedparam()*, the Base Definitions volume of  
35553 IEEE Std 1003.1-2001, <**pthread.h**>

35554 **CHANGE HISTORY**

35555 First released in Issue 6. Included as a response to IEEE PASC Interpretation 1003.1 #96.

35556 **NAME**

35557 pthread\_setspecific — thread-specific data management

35558 **SYNOPSIS**

35559 THR #include <pthread.h>

35560 int pthread\_setspecific(pthread\_key\_t key, const void \*value);

35561

35562 **DESCRIPTION**

35563 Refer to *pthread\_getspecific()*.

35564 **NAME**

35565 pthread\_sigmask, sigprocmask — examine and change blocked signals

35566 **SYNOPSIS**

35567 #include &lt;signal.h&gt;

35568 THR int pthread\_sigmask(int how, const sigset\_t \*restrict set,  
35569 sigset\_t \*restrict oset);35570 CX int sigprocmask(int how, const sigset\_t \*restrict set,  
35571 sigset\_t \*restrict oset);

35572

35573 **DESCRIPTION**35574 THR The *pthread\_sigmask()* function shall examine or change (or both) the calling thread's signal  
35575 mask, regardless of the number of threads in the process. The function shall be equivalent to  
35576 *sigprocmask()*, without the restriction that the call be made in a single-threaded process.35577 In a single-threaded process, the *sigprocmask()* function shall examine or change (or both) the  
35578 signal mask of the calling thread.35579 If the argument *set* is not a null pointer, it points to a set of signals to be used to change the  
35580 currently blocked set.35581 The argument *how* indicates the way in which the set is changed, and the application shall  
35582 ensure it consists of one of the following values:35583 SIG\_BLOCK The resulting set shall be the union of the current set and the signal set  
35584 pointed to by *set*.35585 SIG\_SETMASK The resulting set shall be the signal set pointed to by *set*.35586 SIG\_UNBLOCK The resulting set shall be the intersection of the current set and the  
35587 complement of the signal set pointed to by *set*.35588 If the argument *oset* is not a null pointer, the previous mask shall be stored in the location  
35589 pointed to by *oset*. If *set* is a null pointer, the value of the argument *how* is not significant and the  
35590 process' signal mask shall be unchanged; thus the call can be used to enquire about currently  
35591 blocked signals.35592 If there are any pending unblocked signals after the call to *sigprocmask()*, at least one of those  
35593 signals shall be delivered before the call to *sigprocmask()* returns.35594 It is not possible to block those signals which cannot be ignored. This shall be enforced by the  
35595 system without causing an error to be indicated.35596 If any of the SIGFPE, SIGILL, SIGSEGV, or SIGBUS signals are generated while they are blocked,  
35597 the result is undefined, unless the signal was generated by the *kill()* function, the *sigqueue()*  
35598 function, or the *raise()* function.35599 If *sigprocmask()* fails, the thread's signal mask shall not be changed.35600 The use of the *sigprocmask()* function is unspecified in a multi-threaded process.35601 **RETURN VALUE**35602 THR Upon successful completion *pthread\_sigmask()* shall return 0; otherwise, it shall return the  
35603 corresponding error number.35604 Upon successful completion, *sigprocmask()* shall return 0; otherwise, -1 shall be returned, *errno*  
35605 shall be set to indicate the error, and the process' signal mask shall be unchanged.

35606 **ERRORS**

35607 THR The *pthread\_sigmask()* and *sigprocmask()* functions shall fail if:

35608 [EINVAL] The value of the *how* argument is not equal to one of the defined values.

35609 THR The *pthread\_sigmask()* function shall not return an error code of [EINTR].

35610 **EXAMPLES**

35611 None.

35612 **APPLICATION USAGE**

35613 None.

35614 **RATIONALE**

35615 When a process' signal mask is changed in a signal-catching function that is installed by  
35616 *sigaction()*, the restoration of the signal mask on return from the signal-catching function  
35617 overrides that change (see *sigaction()*). If the signal-catching function was installed with  
35618 *signal()*, it is unspecified whether this occurs.

35619 See *kill()* for a discussion of the requirement on delivery of signals.

35620 **FUTURE DIRECTIONS**

35621 None.

35622 **SEE ALSO**

35623 *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*, *sigpending()*,  
35624 *sigqueue()*, *sigsuspend()*, the Base Definitions volume of IEEE Std 1003.1-2001, <signal.h>

35625 **CHANGE HISTORY**

35626 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

35627 **Issue 5**

35628 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

35629 The *pthread\_sigmask()* function is added for alignment with the POSIX Threads Extension.

35630 **Issue 6**

35631 The *pthread\_sigmask()* function is marked as part of the Threads option.

35632 The SYNOPSIS for *sigprocmask()* is marked as a CX extension to note that the presence of this  
35633 function in the <signal.h> header is an extension to the ISO C standard.

35634 The following changes are made for alignment with the ISO POSIX-1:1996 standard:

- 35635 • The DESCRIPTION is updated to explicitly state the functions which may generate the  
35636 signal.

35637 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

35638 The **restrict** keyword is added to the *pthread\_sigmask()* and *sigprocmask()* prototypes for  
35639 alignment with the ISO/IEC 9899:1999 standard.

35640 **NAME**

35641 pthread\_spin\_destroy, pthread\_spin\_init — destroy or initialize a spin lock object (**ADVANCED**  
 35642 **REALTIME THREADS**)

35643 **SYNOPSIS**

35644 THR SPI #include <pthread.h>

```
35645 int pthread_spin_destroy(pthread_spinlock_t *lock);
35646 int pthread_spin_init(pthread_spinlock_t *lock, int pshared);
35647
```

35648 **DESCRIPTION**

35649 The *pthread\_spin\_destroy()* function shall destroy the spin lock referenced by *lock* and release any  
 35650 resources used by the lock. The effect of subsequent use of the lock is undefined until the lock is  
 35651 reinitialized by another call to *pthread\_spin\_init()*. The results are undefined if  
 35652 *pthread\_spin\_destroy()* is called when a thread holds the lock, or if this function is called with an  
 35653 uninitialized thread spin lock.

35654 The *pthread\_spin\_init()* function shall allocate any resources required to use the spin lock  
 35655 referenced by *lock* and initialize the lock to an unlocked state.

35656 TSH If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is  
 35657 PTHREAD\_PROCESS\_SHARED, the implementation shall permit the spin lock to be operated  
 35658 upon by any thread that has access to the memory where the spin lock is allocated, even if it is  
 35659 allocated in memory that is shared by multiple processes.

35660 If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is  
 35661 PTHREAD\_PROCESS\_PRIVATE, or if the option is not supported, the spin lock shall only be  
 35662 operated upon by threads created within the same process as the thread that initialized the spin  
 35663 lock. If threads of differing processes attempt to operate on such a spin lock, the behavior is  
 35664 undefined.

35665 The results are undefined if *pthread\_spin\_init()* is called specifying an already initialized spin  
 35666 lock. The results are undefined if a spin lock is used without first being initialized.

35667 If the *pthread\_spin\_init()* function fails, the lock is not initialized and the contents of *lock* are  
 35668 undefined.

35669 Only the object referenced by *lock* may be used for performing synchronization.

35670 The result of referring to copies of that object in calls to *pthread\_spin\_destroy()*,  
 35671 *pthread\_spin\_lock()*, *pthread\_spin\_trylock()*, or *pthread\_spin\_unlock()* is undefined.

35672 **RETURN VALUE**

35673 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
 35674 be returned to indicate the error.

35675 **ERRORS**

35676 These functions may fail if:

35677 [EBUSY] The implementation has detected an attempt to initialize or destroy a spin  
 35678 lock while it is in use (for example, while being used in a *pthread\_spin\_lock()*  
 35679 call) by another thread.

35680 [EINVAL] The value specified by *lock* is invalid.

35681 The *pthread\_spin\_init()* function shall fail if:

35682 [EAGAIN] The system lacks the necessary resources to initialize another spin lock.

35683 [ENOMEM] Insufficient memory exists to initialize the lock.

35684 These functions shall not return an error code of [EINTR].

35685 **EXAMPLES**

35686 None.

35687 **APPLICATION USAGE**

35688 The *pthread\_spin\_destroy()* and *pthread\_spin\_init()* functions are part of the Spin Locks option  
35689 and need not be provided on all implementations.

35690 **RATIONALE**

35691 None.

35692 **FUTURE DIRECTIONS**

35693 None.

35694 **SEE ALSO**

35695 *pthread\_spin\_lock()*, *pthread\_spin\_unlock()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
35696 <pthread.h>

35697 **CHANGE HISTORY**

35698 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

35699 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

35700 **NAME**

35701 pthread\_spin\_lock, pthread\_spin\_trylock — lock a spin lock object (**ADVANCED REALTIME**  
35702 **THREADS**)

35703 **SYNOPSIS**

35704 THR SPI #include <pthread.h>

```
35705 int pthread_spin_lock(pthread_spinlock_t *lock);
35706 int pthread_spin_trylock(pthread_spinlock_t *lock);
35707
```

35708 **DESCRIPTION**

35709 The *pthread\_spin\_lock()* function shall lock the spin lock referenced by *lock*. The calling thread  
35710 shall acquire the lock if it is not held by another thread. Otherwise, the thread shall spin (that is,  
35711 shall not return from the *pthread\_spin\_lock()* call) until the lock becomes available. The results  
35712 are undefined if the calling thread holds the lock at the time the call is made. The  
35713 *pthread\_spin\_trylock()* function shall lock the spin lock referenced by *lock* if it is not held by any  
35714 thread. Otherwise, the function shall fail.

35715 The results are undefined if any of these functions is called with an uninitialized spin lock.

35716 **RETURN VALUE**

35717 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
35718 be returned to indicate the error.

35719 **ERRORS**

35720 These functions may fail if:

35721 [EINVAL] The value specified by *lock* does not refer to an initialized spin lock object.

35722 The *pthread\_spin\_lock()* function may fail if:

35723 [EDEADLK] The calling thread already holds the lock.

35724 The *pthread\_spin\_trylock()* function shall fail if:

35725 [EBUSY] A thread currently holds the lock.

35726 These functions shall not return an error code of [EINTR].

35727 **EXAMPLES**

35728 None.

35729 **APPLICATION USAGE**

35730 Applications using this function may be subject to priority inversion, as discussed in the Base  
35731 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

35732 The *pthread\_spin\_lock()* and *pthread\_spin\_trylock()* functions are part of the Spin Locks option  
35733 and need not be provided on all implementations.

35734 **RATIONALE**

35735 None.

35736 **FUTURE DIRECTIONS**

35737 None.

35738 **SEE ALSO**

35739 *pthread\_spin\_destroy()*, *pthread\_spin\_unlock()*, the Base Definitions volume of  
35740 IEEE Std 1003.1-2001, <pthread.h>

35741 **CHANGE HISTORY**

35742 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

35743 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

35744 **NAME**35745 pthread\_spin\_unlock — unlock a spin lock object (**ADVANCED REALTIME THREADS**)35746 **SYNOPSIS**

35747 THR SPI #include &lt;pthread.h&gt;

35748 int pthread\_spin\_unlock(pthread\_spinlock\_t \*lock);

35749

35750 **DESCRIPTION**

35751 The *pthread\_spin\_unlock()* function shall release the spin lock referenced by *lock* which was  
35752 locked via the *pthread\_spin\_lock()* or *pthread\_spin\_trylock()* functions. The results are undefined if  
35753 the lock is not held by the calling thread. If there are threads spinning on the lock when  
35754 *pthread\_spin\_unlock()* is called, the lock becomes available and an unspecified spinning thread  
35755 shall acquire the lock.

35756 The results are undefined if this function is called with an uninitialized thread spin lock.

35757 **RETURN VALUE**

35758 Upon successful completion, the *pthread\_spin\_unlock()* function shall return zero; otherwise, an  
35759 error number shall be returned to indicate the error.

35760 **ERRORS**35761 The *pthread\_spin\_unlock()* function may fail if:

35762 [EINVAL] An invalid argument was specified.

35763 [EPERM] The calling thread does not hold the lock.

35764 This function shall not return an error code of [EINTR].

35765 **EXAMPLES**

35766 None.

35767 **APPLICATION USAGE**

35768 The *pthread\_spin\_unlock()* function is part of the Spin Locks option and need not be provided on  
35769 all implementations.

35770 **RATIONALE**

35771 None.

35772 **FUTURE DIRECTIONS**

35773 None.

35774 **SEE ALSO**

35775 *pthread\_spin\_destroy()*, *pthread\_spin\_lock()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
35776 <pthread.h>

35777 **CHANGE HISTORY**

35778 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

35779 In the SYNOPSIS, the inclusion of &lt;sys/types.h&gt; is no longer required.

35780 **NAME**

35781 pthread\_testcancel — set cancelability state

35782 **SYNOPSIS**

35783 THR #include <pthread.h>

35784 void pthread\_testcancel(void);

35785

35786 **DESCRIPTION**

35787 Refer to *pthread\_setcancelstate()*.

35788 **NAME**

35789 ptsname — get name of the slave pseudo-terminal device

35790 **SYNOPSIS**

35791 XSI #include &lt;stdlib.h&gt;

35792 char \*ptsname(int *fildes*);

35793

35794 **DESCRIPTION**

35795 The *ptsname()* function shall return the name of the slave pseudo-terminal device associated  
 35796 with a master pseudo-terminal device. The *fildes* argument is a file descriptor that refers to the  
 35797 master device. The *ptsname()* function shall return a pointer to a string containing the pathname  
 35798 of the corresponding slave device.

35799 The *ptsname()* function need not be reentrant. A function that is not required to be reentrant is  
 35800 not required to be thread-safe.

35801 **RETURN VALUE**

35802 Upon successful completion, *ptsname()* shall return a pointer to a string which is the name of the  
 35803 pseudo-terminal slave device. Upon failure, *ptsname()* shall return a null pointer. This could  
 35804 occur if *fildes* is an invalid file descriptor or if the slave device name does not exist in the file  
 35805 system.

35806 **ERRORS**

35807 No errors are defined.

35808 **EXAMPLES**

35809 None.

35810 **APPLICATION USAGE**35811 The value returned may point to a static data area that is overwritten by each call to *ptsname()*.35812 **RATIONALE**

35813 None.

35814 **FUTURE DIRECTIONS**

35815 None.

35816 **SEE ALSO**

35817 *grantpt()*, *open()*, *ttyname()*, *unlockpt()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
 35818 <stdlib.h>

35819 **CHANGE HISTORY**

35820 First released in Issue 4, Version 2.

35821 **Issue 5**

35822 Moved from X/OPEN UNIX extension to BASE.

35823 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

35824 **NAME**

35825       putc — put a byte on a stream

35826 **SYNOPSIS**

35827       #include &lt;stdio.h&gt;

35828       int putc(int *c*, FILE \**stream*);35829 **DESCRIPTION**

35830 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
35831 conflict between the requirements described here and the ISO C standard is unintentional. This  
35832 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

35833       The *putc()* function shall be equivalent to *fputc()*, except that if it is implemented as a macro it  
35834 may evaluate *stream* more than once, so the argument should never be an expression with side  
35835 effects.

35836 **RETURN VALUE**35837       Refer to *fputc()*.35838 **ERRORS**35839       Refer to *fputc()*.35840 **EXAMPLES**

35841       None.

35842 **APPLICATION USAGE**

35843       Since it may be implemented as a macro, *putc()* may treat a *stream* argument with side effects  
35844 incorrectly. In particular, *putc(c,\*f++)* does not necessarily work correctly. Therefore, use of this  
35845 function is not recommended in such situations; *fputc()* should be used instead.

35846 **RATIONALE**

35847       None.

35848 **FUTURE DIRECTIONS**

35849       None.

35850 **SEE ALSO**35851       *fputc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>35852 **CHANGE HISTORY**

35853       First released in Issue 1. Derived from Issue 1 of the SVID.

35854 **NAME**35855        `putc_unlocked` — stdio with explicit client locking35856 **SYNOPSIS**35857 TSF        `#include <stdio.h>`35858        `int putc_unlocked(int c, FILE *stream);`

35859

35860 **DESCRIPTION**35861        Refer to `getc_unlocked()`.

35862 **NAME**

35863            putchar — put a byte on a stdout stream

35864 **SYNOPSIS**

35865            #include <stdio.h>

35866            int putchar(int c);

35867 **DESCRIPTION**

35868 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
35869            conflict between the requirements described here and the ISO C standard is unintentional. This  
35870            volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

35871            The function call *putchar(c)* shall be equivalent to *putc(c,stdout)*.

35872 **RETURN VALUE**

35873            Refer to *fputc()*.

35874 **ERRORS**

35875            Refer to *fputc()*.

35876 **EXAMPLES**

35877            None.

35878 **APPLICATION USAGE**

35879            None.

35880 **RATIONALE**

35881            None.

35882 **FUTURE DIRECTIONS**

35883            None.

35884 **SEE ALSO**

35885            *putc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

35886 **CHANGE HISTORY**

35887            First released in Issue 1. Derived from Issue 1 of the SVID.

35888 **NAME**

35889 putchar\_unlocked — stdio with explicit client locking

35890 **SYNOPSIS**

35891 TSF #include &lt;stdio.h&gt;

35892 int putchar\_unlocked(int c);

35893

35894 **DESCRIPTION**35895 Refer to *getc\_unlocked()*.

35896 **NAME**

35897 putenv — change or add a value to an environment

35898 **SYNOPSIS**35899 XSI `#include <stdlib.h>`35900 `int putenv(char *string);`

35901

35902 **DESCRIPTION**

35903 The *putenv()* function shall use the *string* argument to set environment variable values. The  
 35904 *string* argument should point to a string of the form "*name=value*". The *putenv()* function shall  
 35905 make the value of the environment variable *name* equal to *value* by altering an existing variable  
 35906 or creating a new one. In either case, the string pointed to by *string* shall become part of the  
 35907 environment, so altering the string shall change the environment. The space used by *string* is no  
 35908 longer used once a new string which defines *name* is passed to *putenv()*.

35909 The *putenv()* function need not be reentrant. A function that is not required to be reentrant is not  
 35910 required to be thread-safe.

35911 **RETURN VALUE**

35912 Upon successful completion, *putenv()* shall return 0; otherwise, it shall return a non-zero value  
 35913 and set *errno* to indicate the error.

35914 **ERRORS**35915 The *putenv()* function may fail if:

35916 [ENOMEM] Insufficient memory was available.

35917 **EXAMPLES**35918 **Changing the Value of an Environment Variable**

35919 The following example changes the value of the *HOME* environment variable to the value  
 35920 */usr/home*.

35921 `#include <stdlib.h>`35922 `...`35923 `static char *var = "HOME=/usr/home";`35924 `int ret;`35925 `ret = putenv(var);`35926 **APPLICATION USAGE**

35927 The *putenv()* function manipulates the environment pointed to by *environ*, and can be used in  
 35928 conjunction with *getenv()*.

35929 See *exec*, for restrictions on changing the environment in multi-threaded applications.

35930 This routine may use *malloc()* to enlarge the environment.

35931 A potential error is to call *putenv()* with an automatic variable as the argument, then return from  
 35932 the calling function while *string* is still part of the environment.

35933 The *setenv()* function is preferred over this function.

35934 **RATIONALE**

35935 The standard developers noted that *putenv()* is the only function available to add to the  
 35936 environment without permitting memory leaks.

35937 **FUTURE DIRECTIONS**

35938 None.

35939 **SEE ALSO**35940 *exec*, *getenv()*, *malloc()*, *setenv()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>35941 **CHANGE HISTORY**

35942 First released in Issue 1. Derived from Issue 1 of the SVID.

35943 **Issue 5**35944 The type of the argument to this function is changed from **const char \*** to **char \***. This was  
35945 indicated as a FUTURE DIRECTION in previous issues.

35946 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

35947 **Issue 6**35948 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/48 is applied, clarifying wording in the  
35949 DESCRIPTION and adding a new paragraph into APPLICATION USAGE referring readers to  
35950 *exec*. |

## 35951 NAME

35952 putmsg, putpmsg — send a message on a STREAM (STREAMS)

## 35953 SYNOPSIS

35954 XSR #include &lt;stropts.h&gt;

```

35955 int putmsg(int fildes, const struct strbuf *ctlptr,
35956 const struct strbuf *dataptr, int flags);
35957 int putpmsg(int fildes, const struct strbuf *ctlptr,
35958 const struct strbuf *dataptr, int band, int flags);
35959

```

## 35960 DESCRIPTION

35961 The *putmsg()* function shall create a message from a process buffer(s) and send the message to a  
 35962 STREAMS file. The message may contain either a data part, a control part, or both. The data and  
 35963 control parts are distinguished by placement in separate buffers, as described below. The  
 35964 semantics of each part are defined by the STREAMS module that receives the message.

35965 The *putpmsg()* function is equivalent to *putmsg()*, except that the process can send messages in  
 35966 different priority bands. Except where noted, all requirements on *putmsg()* also pertain to  
 35967 *putpmsg()*.

35968 The *fildes* argument specifies a file descriptor referencing an open STREAM. The *ctlptr* and  
 35969 *dataptr* arguments each point to a **strbuf** structure.

35970 The *ctlptr* argument points to the structure describing the control part, if any, to be included in  
 35971 the message. The *buf* member in the **strbuf** structure points to the buffer where the control  
 35972 information resides, and the *len* member indicates the number of bytes to be sent. The *maxlen*  
 35973 member is not used by *putmsg()*. In a similar manner, the argument *dataptr* specifies the data, if  
 35974 any, to be included in the message. The *flags* argument indicates what type of message should be  
 35975 sent and is described further below.

35976 To send the data part of a message, the application shall ensure that *dataptr* is not a null pointer  
 35977 and the *len* member of *dataptr* is 0 or greater. To send the control part of a message, the  
 35978 application shall ensure that the corresponding values are set for *ctlptr*. No data (control) part  
 35979 shall be sent if either *dataptr(ctlptr)* is a null pointer or the *len* member of *dataptr(ctlptr)* is set to  
 35980 -1.

35981 For *putmsg()*, if a control part is specified and *flags* is set to RS\_HIPRI, a high priority message  
 35982 shall be sent. If no control part is specified, and *flags* is set to RS\_HIPRI, *putmsg()* shall fail and  
 35983 set *errno* to [EINVAL]. If *flags* is set to 0, a normal message (priority band equal to 0) shall be  
 35984 sent. If a control part and data part are not specified and *flags* is set to 0, no message shall be  
 35985 sent and 0 shall be returned.

35986 For *putpmsg()*, the flags are different. The *flags* argument is a bitmask with the following  
 35987 mutually-exclusive flags defined: MSG\_HIPRI and MSG\_BAND. If *flags* is set to 0, *putpmsg()*  
 35988 shall fail and set *errno* to [EINVAL]. If a control part is specified and *flags* is set to MSG\_HIPRI  
 35989 and *band* is set to 0, a high-priority message shall be sent. If *flags* is set to MSG\_HIPRI and either  
 35990 no control part is specified or *band* is set to a non-zero value, *putpmsg()* shall fail and set *errno* to  
 35991 [EINVAL]. If *flags* is set to MSG\_BAND, then a message shall be sent in the priority band  
 35992 specified by *band*. If a control part and data part are not specified and *flags* is set to MSG\_BAND,  
 35993 no message shall be sent and 0 shall be returned.

35994 The *putmsg()* function shall block if the STREAM write queue is full due to internal flow control  
 35995 conditions, with the following exceptions:

- 35996 • For high-priority messages, *putmsg()* shall not block on this condition and continues  
 35997 processing the message.

35998           • For other messages, *putmsg()* shall not block but shall fail when the write queue is full and  
35999            O\_NONBLOCK is set.

36000           The *putmsg()* function shall also block, unless prevented by lack of internal resources, while  
36001            waiting for the availability of message blocks in the STREAM, regardless of priority or whether  
36002            O\_NONBLOCK has been specified. No partial message shall be sent.

#### 36003 RETURN VALUE

36004           Upon successful completion, *putmsg()* and *putpmsg()* shall return 0; otherwise, they shall return  
36005            -1 and set *errno* to indicate the error.

#### 36006 ERRORS

36007           The *putmsg()* and *putpmsg()* functions shall fail if:

36008           [EAGAIN]           A non-priority message was specified, the O\_NONBLOCK flag is set, and the  
36009            STREAM write queue is full due to internal flow control conditions; or buffers  
36010            could not be allocated for the message that was to be created.

36011           [EBADF]           *fildes* is not a valid file descriptor open for writing.

36012           [EINTR]           A signal was caught during *putmsg()*.

36013           [EINVAL]           An undefined value is specified in *flags*, or *flags* is set to RS\_HIPRI or  
36014            MSG\_HIPRI and no control part is supplied, or the STREAM or multiplexer  
36015            referenced by *fildes* is linked (directly or indirectly) downstream from a  
36016            multiplexer, or *flags* is set to MSG\_HIPRI and *band* is non-zero (for *putpmsg()*  
36017            only).

36018           [ENOSR]           Buffers could not be allocated for the message that was to be created due to  
36019            insufficient STREAMS memory resources.

36020           [ENOSTR]          A STREAM is not associated with *fildes*.

36021           [ENXIO]           A hangup condition was generated downstream for the specified STREAM.

36022           [EPIPE] or [EIO]   The *fildes* argument refers to a STREAMS-based pipe and the other end of the  
36023            pipe is closed. A SIGPIPE signal is generated for the calling thread.

36024           [ERANGE]          The size of the data part of the message does not fall within the range  
36025            specified by the maximum and minimum packet sizes of the topmost  
36026            STREAM module. This value is also returned if the control part of the message  
36027            is larger than the maximum configured size of the control part of a message,  
36028            or if the data part of a message is larger than the maximum configured size of  
36029            the data part of a message.

36030           In addition, *putmsg()* and *putpmsg()* shall fail if the STREAM head had processed an  
36031            asynchronous error before the call. In this case, the value of *errno* does not reflect the result of  
36032            *putmsg()* or *putpmsg()*, but reflects the prior error.

## 36033 EXAMPLES

36034 **Sending a High-Priority Message**

36035 The value of *fd* is assumed to refer to an open STREAMS file. This call to *putmsg()* does the  
36036 following:

- 36037 1. Creates a high-priority message with a control part and a data part, using the buffers  
36038 pointed to by *ctrlbuf* and *databuf*, respectively.
- 36039 2. Sends the message to the STREAMS file identified by *fd*.

```
36040 #include <stropts.h>
36041 #include <string.h>
36042 ...
36043 int fd;
36044 char *ctrlbuf = "This is the control part";
36045 char *databuf = "This is the data part";
36046 struct strbuf ctrl;
36047 struct strbuf data;
36048 int ret;

36049 ctrl.buf = ctrlbuf;
36050 ctrl.len = strlen(ctrlbuf);

36051 data.buf = databuf;
36052 data.len = strlen(databuf);

36053 ret = putmsg(fd, &ctrl, &data, MSG_HIPRI);
```

36054 **Using putpmsg()**

36055 This example has the same effect as the previous example. In this example, however, the  
36056 *putpmsg()* function creates and sends the message to the STREAMS file.

```
36057 #include <stropts.h>
36058 #include <string.h>
36059 ...
36060 int fd;
36061 char *ctrlbuf = "This is the control part";
36062 char *databuf = "This is the data part";
36063 struct strbuf ctrl;
36064 struct strbuf data;
36065 int ret;

36066 ctrl.buf = ctrlbuf;
36067 ctrl.len = strlen(ctrlbuf);

36068 data.buf = databuf;
36069 data.len = strlen(databuf);

36070 ret = putpmsg(fd, &ctrl, &data, 0, MSG_HIPRI);
```

36071 **APPLICATION USAGE**

36072 None.

36073 **RATIONALE**

36074 None.

36075 **FUTURE DIRECTIONS**

36076 None.

36077 **SEE ALSO**36078 Section 2.6 (on page 38), *getmsg()*, *poll()*, *read()*, *write()*, the Base Definitions volume of  
36079 IEEE Std 1003.1-2001, <**stropts.h**>36080 **CHANGE HISTORY**

36081 First released in Issue 4, Version 2.

36082 **Issue 5**

36083 Moved from X/OPEN UNIX extension to BASE.

36084 The following text is removed from the DESCRIPTION: “The STREAM head guarantees that the  
36085 control part of a message generated by *putmsg()* is at least 64 bytes in length”.36086 **Issue 6**

36087 This function is marked as part of the XSI STREAMS Option Group.

36088 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

36089 **NAME**

36090 puts — put a string on standard output

36091 **SYNOPSIS**

36092 #include &lt;stdio.h&gt;

36093 int puts(const char \*s);

36094 **DESCRIPTION**

36095 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 36096 conflict between the requirements described here and the ISO C standard is unintentional. This  
 36097 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

36098 The *puts()* function shall write the string pointed to by *s*, followed by a <newline>, to the  
 36099 standard output stream *stdout*. The terminating null byte shall not be written.

36100 cx The *st\_ctime* and *st\_mtime* fields of the file shall be marked for update between the successful  
 36101 execution of *puts()* and the next successful completion of a call to *fflush()* or *fclose()* on the same  
 36102 stream or a call to *exit()* or *abort()*.

36103 **RETURN VALUE**

36104 Upon successful completion, *puts()* shall return a non-negative number. Otherwise, it shall  
 36105 cx return EOF, shall set an error indicator for the stream, and *errno* shall be set to indicate the error.

36106 **ERRORS**36107 Refer to *fputc()*.36108 **EXAMPLES**36109 **Printing to Standard Output**

36110 The following example gets the current time, converts it to a string using *localtime()* and  
 36111 *asctime()*, and prints it to standard output using *puts()*. It then prints the number of minutes to  
 36112 an event for which it is waiting.

```

36113 #include <time.h>
36114 #include <stdio.h>
36115 ...
36116 time_t now;
36117 int minutes_to_event;
36118 ...
36119 time(&now);
36120 printf("The time is ");
36121 puts(asctime(localtime(&now)));
36122 printf("There are %d minutes to the event.\n",
36123 minutes_to_event);
36124 ...

```

36125 **APPLICATION USAGE**36126 The *puts()* function appends a <newline>, while *fputs()* does not.36127 **RATIONALE**

36128 None.

36129 **FUTURE DIRECTIONS**

36130 None.

36131 **SEE ALSO**

36132 *fopen()*, *fputs()*, *putc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

36133 **CHANGE HISTORY**

36134 First released in Issue 1. Derived from Issue 1 of the SVID.

36135 **Issue 6**

36136 Extensions beyond the ISO C standard are marked.

36137 **NAME**

36138 pututxline — put an entry into the user accounting database

36139 **SYNOPSIS**

36140 xSI `#include <utmpx.h>`

36141 `struct utmpx *pututxline(const struct utmpx *utmpx);`

36142

36143 **DESCRIPTION**

36144 Refer to *endutxent()*.

36145 **NAME**

36146 putwc — put a wide character on a stream

36147 **SYNOPSIS**

36148 #include &lt;stdio.h&gt;

36149 #include &lt;wchar.h&gt;

36150 wint\_t putwc(wchar\_t *wc*, FILE \**stream*);36151 **DESCRIPTION**

36152 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
36153 conflict between the requirements described here and the ISO C standard is unintentional. This  
36154 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

36155 The *putwc()* function shall be equivalent to *fputwc()*, except that if it is implemented as a macro  
36156 it may evaluate *stream* more than once, so the argument should never be an expression with side  
36157 effects.

36158 **RETURN VALUE**36159 Refer to *fputwc()*.36160 **ERRORS**36161 Refer to *fputwc()*.36162 **EXAMPLES**

36163 None.

36164 **APPLICATION USAGE**

36165 Since it may be implemented as a macro, *putwc()* may treat a *stream* argument with side effects  
36166 incorrectly. In particular, *putwc(wc,\*f++)* need not work correctly. Therefore, use of this function  
36167 is not recommended; *fputwc()* should be used instead.

36168 **RATIONALE**

36169 None.

36170 **FUTURE DIRECTIONS**

36171 None.

36172 **SEE ALSO**36173 *fputwc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>, <wchar.h>36174 **CHANGE HISTORY**

36175 First released as a World-wide Portability Interface in Issue 4.

36176 **Issue 5**

36177 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*  
36178 is changed from **wint\_t** to **wchar\_t**.

36179 The Optional Header (OH) marking is removed from &lt;stdio.h&gt;.

36180 **NAME**

36181 putwchar — put a wide character on a stdout stream

36182 **SYNOPSIS**

36183 #include <wchar.h>

36184 wint\_t putwchar(wchar\_t wc);

36185 **DESCRIPTION**

36186 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
36187 conflict between the requirements described here and the ISO C standard is unintentional. This  
36188 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

36189 The function call *putwchar(wc)* shall be equivalent to *putwc(wc,stdout)*.

36190 **RETURN VALUE**

36191 Refer to *fputwc()*.

36192 **ERRORS**

36193 Refer to *fputwc()*.

36194 **EXAMPLES**

36195 None.

36196 **APPLICATION USAGE**

36197 None.

36198 **RATIONALE**

36199 None.

36200 **FUTURE DIRECTIONS**

36201 None.

36202 **SEE ALSO**

36203 *fputwc()*, *putwc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>

36204 **CHANGE HISTORY**

36205 First released in Issue 4.

36206 **Issue 5**

36207 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*  
36208 is changed from **wint\_t** to **wchar\_t**.

36209 **NAME**

36210 pwrite — write on a file

36211 **SYNOPSIS**

36212 #include &lt;unistd.h&gt;

36213 XSI `ssize_t pwrite(int fildev, const void *buf, size_t nbyte,`  
36214 `off_t offset);`

36215

36216 **DESCRIPTION**36217 Refer to *write()*.

36218 **NAME**

36219 qsort — sort a table of data

36220 **SYNOPSIS**

36221 #include &lt;stdlib.h&gt;

```
36222 void qsort(void *base, size_t nel, size_t width,
36223 int (*compar)(const void *, const void *));
```

36224 **DESCRIPTION**

36225 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 36226 conflict between the requirements described here and the ISO C standard is unintentional. This  
 36227 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

36228 The *qsort()* function shall sort an array of *nel* objects, the initial element of which is pointed to by  
 36229 *base*. The size of each object, in bytes, is specified by the *width* argument. If the *nel* argument has  
 36230 the value zero, the comparison function pointed to by *compar* shall not be called and no  
 36231 rearrangement shall take place.

36232 The application shall ensure that the comparison function pointed to by *compar* does not alter the  
 36233 contents of the array. The implementation may reorder elements of the array between calls to the  
 36234 comparison function, but shall not alter the contents of any individual element.

36235 When the same objects (consisting of *width* bytes, irrespective of their current positions in the  
 36236 array) are passed more than once to the comparison function, the results shall be consistent with  
 36237 one another. That is, they shall define a total ordering on the array.

36238 The contents of the array shall be sorted in ascending order according to a comparison function.  
 36239 The *compar* argument is a pointer to the comparison function, which is called with two  
 36240 arguments that point to the elements being compared. The application shall ensure that the  
 36241 function returns an integer less than, equal to, or greater than 0, if the first argument is  
 36242 considered respectively less than, equal to, or greater than the second. If two members compare  
 36243 as equal, their order in the sorted array is unspecified.

36244 **RETURN VALUE**36245 The *qsort()* function shall not return a value.36246 **ERRORS**

36247 No errors are defined.

36248 **EXAMPLES**

36249 None.

36250 **APPLICATION USAGE**

36251 The comparison function need not compare every byte, so arbitrary data may be contained in  
 36252 the elements in addition to the values being compared.

36253 **RATIONALE**

36254 The requirement that each argument (hereafter referred to as *p*) to the comparison function is a  
 36255 pointer to elements of the array implies that for every call, for each argument separately, all of  
 36256 the following expressions are nonzero:

```
36257 ((char *)p - (char *)base) % width == 0
```

```
36258 (char *)p >= (char *)base
```

```
36259 (char *)p < (char *)base + nel * width
```

36260 **FUTURE DIRECTIONS**

36261 None.

36262 **SEE ALSO**

36263 The Base Definitions volume of IEEE Std 1003.1-2001, &lt;stdlib.h&gt;

36264 **CHANGE HISTORY**

36265 First released in Issue 1. Derived from Issue 1 of the SVID.

36266 **Issue 6**

36267 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

36268 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/49 is applied, adding the last sentence to  
36269 the first non-shaded paragraph in the DESCRIPTION, and the following two paragraphs. The  
36270 RATIONALE is also updated. These changes are for alignment with the ISO C standard. |

36271 **NAME**

36272 raise — send a signal to the executing process

36273 **SYNOPSIS**

36274 #include <signal.h>

36275 int raise(int sig);

36276 **DESCRIPTION**

36277 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
36278 conflict between the requirements described here and the ISO C standard is unintentional. This  
36279 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

36280 CX The *raise()* function shall send the signal *sig* to the executing thread or process. If a signal  
36281 handler is called, the *raise()* function shall not return until after the signal handler does.

36282 THR If the implementation supports the Threads option, the effect of the *raise()* function shall be  
36283 equivalent to calling:

36284 pthread\_kill(pthread\_self(), sig);

36285

36286 CX Otherwise, the effect of the *raise()* function shall be equivalent to calling:

36287 kill(getpid(), sig);

36288

36289 **RETURN VALUE**

36290 CX Upon successful completion, 0 shall be returned. Otherwise, a non-zero value shall be returned  
36291 and *errno* shall be set to indicate the error.

36292 **ERRORS**

36293 The *raise()* function shall fail if:

36294 CX [EINVAL] The value of the *sig* argument is an invalid signal number.

36295 **EXAMPLES**

36296 None.

36297 **APPLICATION USAGE**

36298 None.

36299 **RATIONALE**

36300 The term “thread” is an extension to the ISO C standard.

36301 **FUTURE DIRECTIONS**

36302 None.

36303 **SEE ALSO**

36304 *kill()*, *sigaction()*, the Base Definitions volume of IEEE Std 1003.1-2001, <signal.h>,  
36305 <sys/types.h>

36306 **CHANGE HISTORY**

36307 First released in Issue 4. Derived from the ANSI C standard.

36308 **Issue 5**

36309 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

36310 **Issue 6**

36311 Extensions beyond the ISO C standard are marked.

36312 The following new requirements on POSIX implementations derive from alignment with the  
36313 Single UNIX Specification:

- 36314 • In the RETURN VALUE section, the requirement to set *errno* on error is added.
- 36315 • The [EINVAL] error condition is added.

36316 **NAME**

36317 rand, rand\_r, srand — pseudo-random number generator

36318 **SYNOPSIS**

36319 #include &lt;stdlib.h&gt;

36320 int rand(void);

36321 TSF int rand\_r(unsigned \*seed);

36322 void srand(unsigned seed);

36323 **DESCRIPTION**

36324 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 36325 conflict between the requirements described here and the ISO C standard is unintentional. This  
 36326 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

36327 The *rand()* function shall compute a sequence of pseudo-random integers in the range  
 36328 XSI [0,{RAND\_MAX}] with a period of at least  $2^{32}$ .

36329 CX The *rand()* function need not be reentrant. A function that is not required to be reentrant is not  
 36330 required to be thread-safe.

36331 TSF The *rand\_r()* function shall compute a sequence of pseudo-random integers in the range  
 36332 [0,{RAND\_MAX}]. (The value of the {RAND\_MAX} macro shall be at least 32 767.)

36333 If *rand\_r()* is called with the same initial value for the object pointed to by *seed* and that object is  
 36334 not modified between successive returns and calls to *rand\_r()*, the same sequence shall be  
 36335 generated.

36336 The *srand()* function uses the argument as a seed for a new sequence of pseudo-random  
 36337 numbers to be returned by subsequent calls to *rand()*. If *srand()* is then called with the same  
 36338 seed value, the sequence of pseudo-random numbers shall be repeated. If *rand()* is called before  
 36339 any calls to *srand()* are made, the same sequence shall be generated as when *srand()* is first  
 36340 called with a seed value of 1.

36341 The implementation shall behave as if no function defined in this volume of  
 36342 IEEE Std 1003.1-2001 calls *rand()* or *srand()*.

36343 **RETURN VALUE**36344 The *rand()* function shall return the next pseudo-random number in the sequence.36345 TSF The *rand\_r()* function shall return a pseudo-random integer.36346 The *srand()* function shall not return a value.36347 **ERRORS**

36348 No errors are defined.

36349 **EXAMPLES**36350 **Generating a Pseudo-Random Number Sequence**

36351 The following example demonstrates how to generate a sequence of pseudo-random numbers.

36352 #include &lt;stdio.h&gt;

36353 #include &lt;stdlib.h&gt;

36354 ...

36355 long count, i;

36356 char \*keyst;

36357 int elementlen, len;

36358 char c;

```

36359 ...
36360 /* Initial random number generator. */
36361 srand(1);

36362 /* Create keys using only lowercase characters */
36363 len = 0;
36364 for (i=0; i<count; i++) {
36365 while (len < elementlen) {
36366 c = (char) (rand() % 128);
36367 if (islower(c))
36368 keystr[len++] = c;
36369 }

36370 keystr[len] = '\0';
36371 printf("%s Element%0*ld\n", keystr, elementlen, i);
36372 len = 0;
36373 }

```

### 36374 **Generating the Same Sequence on Different Machines**

36375 The following code defines a pair of functions that could be incorporated into applications  
 36376 wishing to ensure that the same sequence of numbers is generated across different machines.

```

36377 static unsigned long next = 1;
36378 int myrand(void) /* RAND_MAX assumed to be 32767. */
36379 {
36380 next = next * 1103515245 + 12345;
36381 return((unsigned)(next/65536) % 32768);
36382 }

36383 void mysrand(unsigned seed)
36384 {
36385 next = seed;
36386 }

```

### 36387 **APPLICATION USAGE**

36388 The *drand48()* function provides a much more elaborate random number generator.

36389 The limitations on the amount of state that can be carried between one function call and another  
 36390 mean the *rand\_r()* function can never be implemented in a way which satisfies all of the  
 36391 requirements on a pseudo-random number generator. Therefore this function should be avoided  
 36392 whenever non-trivial requirements (including safety) have to be fulfilled.

### 36393 **RATIONALE**

36394 The ISO C standard *rand()* and *srand()* functions allow per-process pseudo-random streams  
 36395 shared by all threads. Those two functions need not change, but there has to be mutual-  
 36396 exclusion that prevents interference between two threads concurrently accessing the random  
 36397 number generator.

36398 With regard to *rand()*, there are two different behaviors that may be wanted in a multi-threaded  
 36399 program:

- 36400 1. A single per-process sequence of pseudo-random numbers that is shared by all threads  
 36401 that call *rand()*
- 36402 2. A different sequence of pseudo-random numbers for each thread that calls *rand()*

36403 This is provided by the modified thread-safe function based on whether the seed value is global  
36404 to the entire process or local to each thread.

36405 This does not address the known deficiencies of the *rand()* function implementations, which  
36406 have been approached by maintaining more state. In effect, this specifies new thread-safe forms  
36407 of a deficient function.

36408 **FUTURE DIRECTIONS**

36409 None.

36410 **SEE ALSO**

36411 *drand48()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>

36412 **CHANGE HISTORY**

36413 First released in Issue 1. Derived from Issue 1 of the SVID.

36414 **Issue 5**

36415 The *rand\_r()* function is included for alignment with the POSIX Threads Extension.

36416 A note indicating that the *rand()* function need not be reentrant is added to the DESCRIPTION.

36417 **Issue 6**

36418 Extensions beyond the ISO C standard are marked.

36419 The *rand\_r()* function is marked as part of the Thread-Safe Functions option.

36420 **NAME**

36421 random — generate pseudo-random number

36422 **SYNOPSIS**36423 XSI `#include <stdlib.h>`36424 `long random(void);`

36425

36426 **DESCRIPTION**36427 Refer to *initstate()*.

## 36428 NAME

36429 pread, read — read from a file

## 36430 SYNOPSIS

36431 #include &lt;unistd.h&gt;

36432 XSI `ssize_t pread(int fildev, void *buf, size_t nbyte, off_t offset);`36433 `ssize_t read(int fildev, void *buf, size_t nbyte);`

## 36434 DESCRIPTION

36435 The *read()* function shall attempt to read *nbyte* bytes from the file associated with the open file  
 36436 descriptor, *fildev*, into the buffer pointed to by *buf*. The behavior of multiple concurrent reads on  
 36437 the same pipe, FIFO, or terminal device is unspecified.

36438 Before any action described below is taken, and if *nbyte* is zero, the *read()* function may detect  
 36439 and return errors as described below. In the absence of errors, or if error detection is not  
 36440 performed, the *read()* function shall return zero and have no other results.

36441 On files that support seeking (for example, a regular file), the *read()* shall start at a position in  
 36442 the file given by the file offset associated with *fildev*. The file offset shall be incremented by the  
 36443 number of bytes actually read.

36444 Files that do not support seeking—for example, terminals—always read from the current  
 36445 position. The value of a file offset associated with such a file is undefined.

36446 No data transfer shall occur past the current end-of-file. If the starting position is at or after the  
 36447 end-of-file, 0 shall be returned. If the file refers to a device special file, the result of subsequent  
 36448 *read()* requests is implementation-defined.

36449 If the value of *nbyte* is greater than {SSIZE\_MAX}, the result is implementation-defined.

36450 When attempting to read from an empty pipe or FIFO:

- 36451 • If no process has the pipe open for writing, *read()* shall return 0 to indicate end-of-file.
- 36452 • If some process has the pipe open for writing and O\_NONBLOCK is set, *read()* shall return  
 36453 -1 and set *errno* to [EAGAIN].
- 36454 • If some process has the pipe open for writing and O\_NONBLOCK is clear, *read()* shall block  
 36455 the calling thread until some data is written or the pipe is closed by all processes that had the  
 36456 pipe open for writing.

36457 When attempting to read a file (other than a pipe or FIFO) that supports non-blocking reads and  
 36458 has no data currently available:

- 36459 • If O\_NONBLOCK is set, *read()* shall return -1 and set *errno* to [EAGAIN].
- 36460 • If O\_NONBLOCK is clear, *read()* shall block the calling thread until some data becomes  
 36461 available.
- 36462 • The use of the O\_NONBLOCK flag has no effect if there is some data available.

36463 The *read()* function reads data previously written to a file. If any portion of a regular file prior to  
 36464 the end-of-file has not been written, *read()* shall return bytes with value 0. For example, *lseek()*  
 36465 allows the file offset to be set beyond the end of existing data in the file. If data is later written at  
 36466 this point, subsequent reads in the gap between the previous end of data and the newly written  
 36467 data shall return bytes with value 0 until data is written into the gap.

36468 Upon successful completion, where *nbyte* is greater than 0, *read()* shall mark for update the  
 36469 *st\_atime* field of the file, and shall return the number of bytes read. This number shall never be  
 36470 greater than *nbyte*. The value returned may be less than *nbyte* if the number of bytes left in the

- 36471 file is less than *nbyte*, if the *read()* request was interrupted by a signal, or if the file is a pipe or  
36472 FIFO or special file and has fewer than *nbyte* bytes immediately available for reading. For  
36473 example, a *read()* from a file associated with a terminal may return one typed line of data.
- 36474 If a *read()* is interrupted by a signal before it reads any data, it shall return  $-1$  with *errno* set to  
36475 [EINTR].
- 36476 If a *read()* is interrupted by a signal after it has successfully read some data, it shall return the  
36477 number of bytes read.
- 36478 For regular files, no data transfer shall occur past the offset maximum established in the open  
36479 file description associated with *fdes*.
- 36480 If *fdes* refers to a socket, *read()* shall be equivalent to *recv()* with no flags set.
- 36481 SIO If the O\_DSYNC and O\_RSYNC bits have been set, read I/O operations on the file descriptor  
36482 shall complete as defined by synchronized I/O data integrity completion. If the O\_SYNC and  
36483 O\_RSYNC bits have been set, read I/O operations on the file descriptor shall complete as  
36484 defined by synchronized I/O file integrity completion.
- 36485 SHM If *fdes* refers to a shared memory object, the result of the *read()* function is unspecified.
- 36486 TYM If *fdes* refers to a typed memory object, the result of the *read()* function is unspecified.
- 36487 XSR A *read()* from a STREAMS file can read data in three different modes: *byte-stream* mode,  
36488 *message-nondiscard* mode, and *message-discard* mode. The default shall be byte-stream mode. This  
36489 can be changed using the I\_SRDOPT *ioctl()* request, and can be tested with I\_GRDOPT *ioctl()*.  
36490 In byte-stream mode, *read()* shall retrieve data from the STREAM until as many bytes as were  
36491 requested are transferred, or until there is no more data to be retrieved. Byte-stream mode  
36492 ignores message boundaries.
- 36493 In STREAMS message-nondiscard mode, *read()* shall retrieve data until as many bytes as were  
36494 requested are transferred, or until a message boundary is reached. If *read()* does not retrieve all  
36495 the data in a message, the remaining data shall be left on the STREAM, and can be retrieved by  
36496 the next *read()* call. Message-discard mode also retrieves data until as many bytes as were  
36497 requested are transferred, or a message boundary is reached. However, unread data remaining  
36498 in a message after the *read()* returns shall be discarded, and shall not be available for a  
36499 subsequent *read()*, *getmsg()*, or *getpmsg()* call.
- 36500 How *read()* handles zero-byte STREAMS messages is determined by the current read mode  
36501 setting. In byte-stream mode, *read()* shall accept data until it has read *nbyte* bytes, or until there  
36502 is no more data to read, or until a zero-byte message block is encountered. The *read()* function  
36503 shall then return the number of bytes read, and place the zero-byte message back on the  
36504 STREAM to be retrieved by the next *read()*, *getmsg()*, or *getpmsg()*. In message-nondiscard mode  
36505 or message-discard mode, a zero-byte message shall return 0 and the message shall be removed  
36506 from the STREAM. When a zero-byte message is read as the first message on a STREAM, the  
36507 message shall be removed from the STREAM and 0 shall be returned, regardless of the read  
36508 mode.
- 36509 A *read()* from a STREAMS file shall return the data in the message at the front of the STREAM  
36510 head read queue, regardless of the priority band of the message.
- 36511 By default, STREAMS are in control-normal mode, in which a *read()* from a STREAMS file can  
36512 only process messages that contain a data part but do not contain a control part. The *read()* shall  
36513 fail if a message containing a control part is encountered at the STREAM head. This default  
36514 action can be changed by placing the STREAM in either control-data mode or control-discard  
36515 mode with the I\_SRDOPT *ioctl()* command. In control-data mode, *read()* shall convert any  
36516 control part to data and pass it to the application before passing any data part originally present

36517 in the same message. In control-discard mode, *read()* shall discard message control parts but  
36518 return to the process any data part in the message.

36519 In addition, *read()* shall fail if the STREAM head had processed an asynchronous error before the  
36520 call. In this case, the value of *errno* shall not reflect the result of *read()*, but reflect the prior error.  
36521 If a hangup occurs on the STREAM being read, *read()* shall continue to operate normally until  
36522 the STREAM head read queue is empty. Thereafter, it shall return 0.

36523 XSI The *pread()* function shall be equivalent to *read()*, except that it shall read from a given position  
36524 in the file without changing the file pointer. The first three arguments to *pread()* are the same as  
36525 *read()* with the addition of a fourth argument *offset* for the desired position inside the file. An  
36526 attempt to perform a *pread()* on a file that is incapable of seeking shall result in an error.

#### 36527 RETURN VALUE

36528 XSI Upon successful completion, *read()* and *pread()* shall return a non-negative integer indicating the  
36529 number of bytes actually read. Otherwise, the functions shall return  $-1$  and set *errno* to indicate  
36530 the error.

#### 36531 ERRORS

36532 XSI The *read()* and *pread()* functions shall fail if:

36533 [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor and the process would be  
36534 delayed.

36535 [EBADF] The *fildev* argument is not a valid file descriptor open for reading.

36536 XSR [EBADMSG] The file is a STREAM file that is set to control-normal mode and the message  
36537 waiting to be read includes a control part.

36538 [EINTR] The read operation was terminated due to the receipt of a signal, and no data  
36539 was transferred.

36540 XSR [EINVAL] The STREAM or multiplexer referenced by *fildev* is linked (directly or  
36541 indirectly) downstream from a multiplexer.

36542 [EIO] The process is a member of a background process attempting to read from its  
36543 controlling terminal, the process is ignoring or blocking the SIGTTIN signal,  
36544 or the process group is orphaned. This error may also be generated for  
36545 implementation-defined reasons.

36546 XSI [EISDIR] The *fildev* argument refers to a directory and the implementation does not  
36547 allow the directory to be read using *read()* or *pread()*. The *readdir()* function  
36548 should be used instead.

36549 [EOVERFLOW] The file is a regular file, *nbyte* is greater than 0, the starting position is before  
36550 the end-of-file, and the starting position is greater than or equal to the offset  
36551 maximum established in the open file description associated with *fildev*.

36552 The *read()* function shall fail if:

36553 [EAGAIN] or [EWOULDBLOCK]

36554 The file descriptor is for a socket, is marked O\_NONBLOCK, and no data is  
36555 waiting to be received.

36556 [ECONNRESET] A read was attempted on a socket and the connection was forcibly closed by  
36557 its peer.

36558 [ENOTCONN] A read was attempted on a socket that is not connected.

36559 [ETIMEDOUT] A read was attempted on a socket and a transmission timeout occurred.

- 36560 XSI The `read()` and `pread()` functions may fail if:
- 36561 [EIO] A physical I/O error has occurred.
- 36562 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 36563 [ENOMEM] Insufficient memory was available to fulfill the request.
- 36564 [ENXIO] A request was made of a nonexistent device, or the request was outside the capabilities of the device.
- 36565
- 36566 The `pread()` function shall fail, and the file pointer shall remain unchanged, if:
- 36567 XSI [EINVAL] The `offset` argument is invalid. The value is negative.
- 36568 XSI [EOVERFLOW] The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the file.
- 36569
- 36570 XSI [ENXIO] A request was outside the capabilities of the device.
- 36571 XSI [ESPIPE] `fdes` is associated with a pipe or FIFO.

### 36572 EXAMPLES

#### 36573 Reading Data into a Buffer

36574 The following example reads data from the file associated with the file descriptor `fd` into the buffer pointed to by `buf`.

```
36575
36576 #include <sys/types.h>
36577 #include <unistd.h>
36578 ...
36579 char buf[20];
36580 size_t nbytes;
36581 ssize_t bytes_read;
36582 int fd;
36583 ...
36584 nbytes = sizeof(buf);
36585 bytes_read = read(fd, buf, nbytes);
36586 ...
```

#### 36587 APPLICATION USAGE

36588 None.

#### 36589 RATIONALE

36590 This volume of IEEE Std 1003.1-2001 does not specify the value of the file offset after an error is returned; there are too many cases. For programming errors, such as [EBADF], the concept is meaningless since no file is involved. For errors that are detected immediately, such as [EAGAIN], clearly the pointer should not change. After an interrupt or hardware error, however, an updated value would be very useful and is the behavior of many implementations.

36595 Note that a `read()` of zero bytes does not modify `st_atime`. A `read()` that requests more than zero bytes, but returns zero, shall modify `st_atime`.

36597 Implementations are allowed, but not required, to perform error checking for `read()` requests of zero bytes.

36598

36599 **Input and Output**

36600 The use of I/O with large byte counts has always presented problems. Ideas such as *lread()* and  
36601 *lwrite()* (using and returning **longs**) were considered at one time. The current solution is to use  
36602 abstract types on the ISO C standard function to *read()* and *write()*. The abstract types can be  
36603 declared so that existing functions work, but can also be declared so that larger types can be  
36604 represented in future implementations. It is presumed that whatever constraints limit the  
36605 maximum range of **size\_t** also limit portable I/O requests to the same range. This volume of  
36606 IEEE Std 1003.1-2001 also limits the range further by requiring that the byte count be limited so  
36607 that a signed return value remains meaningful. Since the return type is also a (signed) abstract  
36608 type, the byte count can be defined by the implementation to be larger than an **int** can hold.

36609 The standard developers considered adding atomicity requirements to a pipe or FIFO, but  
36610 recognized that due to the nature of pipes and FIFOs there could be no guarantee of atomicity of  
36611 reads of {PIPE\_BUF} or any other size that would be an aid to applications portability.

36612 This volume of IEEE Std 1003.1-2001 requires that no action be taken for *read()* or *write()* when  
36613 *nbyte* is zero. This is not intended to take precedence over detection of errors (such as invalid  
36614 buffer pointers or file descriptors). This is consistent with the rest of this volume of  
36615 IEEE Std 1003.1-2001, but the phrasing here could be misread to require detection of the zero  
36616 case before any other errors. A value of zero is to be considered a correct value, for which the  
36617 semantics are a no-op.

36618 I/O is intended to be atomic to ordinary files and pipes and FIFOs. Atomic means that all the  
36619 bytes from a single operation that started out together end up together, without interleaving  
36620 from other I/O operations. It is a known attribute of terminals that this is not honored, and  
36621 terminals are explicitly (and implicitly permanently) excepted, making the behavior unspecified.  
36622 The behavior for other device types is also left unspecified, but the wording is intended to imply  
36623 that future standards might choose to specify atomicity (or not).

36624 There were recommendations to add format parameters to *read()* and *write()* in order to handle  
36625 networked transfers among heterogeneous file system and base hardware types. Such a facility  
36626 may be required for support by the OSI presentation of layer services. However, it was  
36627 determined that this should correspond with similar C-language facilities, and that is beyond the  
36628 scope of this volume of IEEE Std 1003.1-2001. The concept was suggested to the developers of  
36629 the ISO C standard for their consideration as a possible area for future work.

36630 In 4.3 BSD, a *read()* or *write()* that is interrupted by a signal before transferring any data does not  
36631 by default return an [EINTR] error, but is restarted. In 4.2 BSD, 4.3 BSD, and the Eighth Edition,  
36632 there is an additional function, *select()*, whose purpose is to pause until specified activity (data  
36633 to read, space to write, and so on) is detected on specified file descriptors. It is common in  
36634 applications written for those systems for *select()* to be used before *read()* in situations (such as  
36635 keyboard input) where interruption of I/O due to a signal is desired.

36636 The issue of which files or file types are interruptible is considered an implementation design  
36637 issue. This is often affected primarily by hardware and reliability issues.

36638 There are no references to actions taken following an “unrecoverable error”. It is considered  
36639 beyond the scope of this volume of IEEE Std 1003.1-2001 to describe what happens in the case of  
36640 hardware errors.

36641 Previous versions of IEEE Std 1003.1-2001 allowed two very different behaviors with regard to  
36642 the handling of interrupts. In order to minimize the resulting confusion, it was decided that  
36643 IEEE Std 1003.1-2001 should support only one of these behaviors. Historical practice on AT&T-  
36644 derived systems was to have *read()* and *write()* return **-1** and set *errno* to [EINTR] when  
36645 interrupted after some, but not all, of the data requested had been transferred. However, the U.S.  
36646 Department of Commerce FIPS 151-1 and FIPS 151-2 require the historical BSD behavior, in

36647 which *read()* and *write()* return the number of bytes actually transferred before the interrupt. If  
 36648 *-1* is returned when any data is transferred, it is difficult to recover from the error on a seekable  
 36649 device and impossible on a non-seekable device. Most new implementations support this  
 36650 behavior. The behavior required by IEEE Std 1003.1-2001 is to return the number of bytes  
 36651 transferred.

36652 IEEE Std 1003.1-2001 does not specify when an implementation that buffers *read()*ss actually  
 36653 moves the data into the user-supplied buffer, so an implementation may chose to do this at the  
 36654 latest possible moment. Therefore, an interrupt arriving earlier may not cause *read()* to return a  
 36655 partial byte count, but rather to return *-1* and set *errno* to [EINTR].

36656 Consideration was also given to combining the two previous options, and setting *errno* to  
 36657 [EINTR] while returning a short count. However, not only is there no existing practice that  
 36658 implements this, it is also contradictory to the idea that when *errno* is set, the function  
 36659 responsible shall return *-1*.

#### 36660 FUTURE DIRECTIONS

36661 None.

#### 36662 SEE ALSO

36663 *fcntl()*, *ioctl()*, *lseek()*, *open()*, *pipe()*, *readv()*, the Base Definitions volume of  
 36664 IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, <**stropts.h**>, <**sys/uio.h**>,  
 36665 <**unistd.h**>

#### 36666 CHANGE HISTORY

36667 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 36668 Issue 5

36669 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
 36670 Threads Extension.

36671 Large File Summit extensions are added.

36672 The *pread()* function is added.

#### 36673 Issue 6

36674 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are  
 36675 marked as part of the XSI STREAMS Option Group.

36676 The following new requirements on POSIX implementations derive from alignment with the  
 36677 Single UNIX Specification:

36678 • The DESCRIPTION now states that if *read()* is interrupted by a signal after it has successfully  
 36679 read some data, it returns the number of bytes read. In Issue 3, it was optional whether *read()*  
 36680 returned the number of bytes read, or whether it returned *-1* with *errno* set to [EINTR]. This  
 36681 is a FIPS requirement.

36682 • In the DESCRIPTION, text is added to indicate that for regular files, no data transfer occurs  
 36683 past the offset maximum established in the open file description associated with *files*. This  
 36684 change is to support large files.

36685 • The [EOVERFLOW] mandatory error condition is added.

36686 • The [ENXIO] optional error condition is added.

36687 Text referring to sockets is added to the DESCRIPTION.

36688 The following changes were made to align with the IEEE P1003.1a draft standard:

36689 • The effect of reading zero bytes is clarified.

- 36690 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
36691 *read()* results are unspecified for typed memory objects.
- 36692 New RATIONALE is added to explain the atomicity requirements for input and output  
36693 operations.
- 36694 The following error conditions are added for operations on sockets: [EAGAIN],  
36695 [ECONNRESET], [ENOTCONN], and [ETIMEDOUT].
- 36696 The [EIO] error is changed to “may fail”.
- 36697 The following error conditions are added for operations on sockets: [ENOBUFS] and  
36698 [ENOMEM].
- 36699 The *readv()* function is split out into a separate reference page.

## 36700 NAME

36701 readdir, readdir\_r — read a directory

## 36702 SYNOPSIS

36703 #include &lt;dirent.h&gt;

36704 struct dirent \*readdir(DIR \*dirp);

36705 TSF int readdir\_r(DIR \*restrict dirp, struct dirent \*restrict entry,

36706 struct dirent \*\*restrict result);

36707

## 36708 DESCRIPTION

36709 The type **DIR**, which is defined in the <**dirent.h**> header, represents a *directory stream*, which is  
 36710 an ordered sequence of all the directory entries in a particular directory. Directory entries  
 36711 represent files; files may be removed from a directory or added to a directory asynchronously to  
 36712 the operation of *readdir()*.

36713 The *readdir()* function shall return a pointer to a structure representing the directory entry at the  
 36714 current position in the directory stream specified by the argument *dirp*, and position the  
 36715 directory stream at the next entry. It shall return a null pointer upon reaching the end of the  
 36716 directory stream. The structure **dirent** defined in the <**dirent.h**> header describes a directory  
 36717 entry.

36718 The *readdir()* function shall not return directory entries containing empty names. If entries for  
 36719 dot or dot-dot exist, one entry shall be returned for dot and one entry shall be returned for dot-  
 36720 dot; otherwise, they shall not be returned.

36721 The pointer returned by *readdir()* points to data which may be overwritten by another call to  
 36722 *readdir()* on the same directory stream. This data is not overwritten by another call to *readdir()*  
 36723 on a different directory stream.

36724 If a file is removed from or added to the directory after the most recent call to *opendir()* or  
 36725 *rewinddir()*, whether a subsequent call to *readdir()* returns an entry for that file is unspecified.

36726 The *readdir()* function may buffer several directory entries per actual read operation; *readdir()*  
 36727 shall mark for update the *st\_atime* field of the directory each time the directory is actually read.

36728 After a call to *fork()*, either the parent or child (but not both) may continue processing the  
 36729 XSI directory stream using *readdir()*, *rewinddir()*, or *seekdir()*. If both the parent and child processes  
 36730 use these functions, the result is undefined.

36731 If the entry names a symbolic link, the value of the *d\_ino* member is unspecified.

36732 The *readdir()* function need not be reentrant. A function that is not required to be reentrant is not  
 36733 required to be thread-safe.

36734 TSF The *readdir\_r()* function shall initialize the **dirent** structure referenced by *entry* to represent the  
 36735 directory entry at the current position in the directory stream referred to by *dirp*, store a pointer  
 36736 to this structure at the location referenced by *result*, and position the directory stream at the next  
 36737 entry.

36738 The storage pointed to by *entry* shall be large enough for a **dirent** with an array of **char** *d\_name*  
 36739 members containing at least {NAME\_MAX}+1 elements.

36740 Upon successful return, the pointer returned at *\*result* shall have the same value as the argument  
 36741 *entry*. Upon reaching the end of the directory stream, this pointer shall have the value NULL.

36742 The *readdir\_r()* function shall not return directory entries containing empty names.

36743 If a file is removed from or added to the directory after the most recent call to *opendir()* or  
 36744 *rewinddir()*, whether a subsequent call to *readdir\_r()* returns an entry for that file is unspecified.

36745 The *readdir\_r()* function may buffer several directory entries per actual read operation; the  
 36746 *readdir\_r()* function shall mark for update the *st\_atime* field of the directory each time the  
 36747 directory is actually read.

36748 Applications wishing to check for error situations should set *errno* to 0 before calling *readdir()*. If  
 36749 *errno* is set to non-zero on return, an error occurred.

#### 36750 RETURN VALUE

36751 Upon successful completion, *readdir()* shall return a pointer to an object of type **struct dirent**.  
 36752 When an error is encountered, a null pointer shall be returned and *errno* shall be set to indicate  
 36753 the error. When the end of the directory is encountered, a null pointer shall be returned and *errno*  
 36754 is not changed.

36755 TSF If successful, the *readdir\_r()* function shall return zero; otherwise, an error number shall be  
 36756 returned to indicate the error.

#### 36757 ERRORS

36758 The *readdir()* function shall fail if:

36759 [EOVERFLOW] One of the values in the structure to be returned cannot be represented  
 36760 correctly.

36761 The *readdir\_r()* function may fail if:

36762 [EBADF] The *dirp* argument does not refer to an open directory stream.

36763 [ENOENT] The current position of the directory stream is invalid.

36764 The *readdir\_r()* function may fail if:

36765 [EBADF] The *dirp* argument does not refer to an open directory stream.

#### 36766 EXAMPLES

36767 The following sample program searches the current directory for each of the arguments supplied  
 36768 on the command line.

```

36769 #include <dirent.h>
36770 #include <errno.h>
36771 #include <stdio.h>
36772 #include <string.h>

36773 static void lookup(const char *arg)
36774 {
36775 DIR *dirp;
36776 struct dirent *dp;

36777 if ((dirp = opendir(".")) == NULL) {
36778 perror("couldn't open '.'");
36779 return;
36780 }

36781 do {
36782 errno = 0;
36783 if ((dp = readdir(dirp)) != NULL) {
36784 if (strcmp(dp->d_name, arg) != 0)
36785 continue;

```

```

36786 (void) printf("found %s\n", arg);
36787 (void) closedir(dirp);
36788 return;
36789 }
36790 } while (dp != NULL);
36791 if (errno != 0)
36792 perror("error reading directory");
36793 else
36794 (void) printf("failed to find %s\n", arg);
36795 (void) closedir(dirp);
36796 return;
36797 }
36798 int main(int argc, char *argv[])
36799 {
36800 int i;
36801 for (i = 1; i < argc; i++)
36802 lookup(argv[i]);
36803 return (0);
36804 }

```

#### 36805 APPLICATION USAGE

36806 The *readdir()* function should be used in conjunction with *opendir()*, *closedir()*, and *rewinddir()* to  
 36807 examine the contents of the directory.

36808 The *readdir\_r()* function is thread-safe and shall return values in a user-supplied buffer instead  
 36809 of possibly using a static data area that may be overwritten by each call.

#### 36810 RATIONALE

36811 The returned value of *readdir()* merely *represents* a directory entry. No equivalence should be  
 36812 inferred.

36813 Historical implementations of *readdir()* obtain multiple directory entries on a single read  
 36814 operation, which permits subsequent *readdir()* operations to operate from the buffered  
 36815 information. Any wording that required each successful *readdir()* operation to mark the  
 36816 directory *st\_atime* field for update would disallow such historical performance-oriented  
 36817 implementations.

36818 Since *readdir()* returns NULL when it detects an error and when the end of the directory is  
 36819 encountered, an application that needs to tell the difference must set *errno* to zero before the call  
 36820 and check it if NULL is returned. Since the function must not change *errno* in the second case  
 36821 and must set it to a non-zero value in the first case, a zero *errno* after a call returning NULL  
 36822 indicates end-of-directory; otherwise, an error.

36823 Routines to deal with this problem more directly were proposed:

```

36824 int derror (dirp)
36825 DIR *dirp;
36826 void clearderr (dirp)
36827 DIR *dirp;

```

36828 The first would indicate whether an error had occurred, and the second would clear the error  
 36829 indication. The simpler method involving *errno* was adopted instead by requiring that *readdir()*  
 36830 not change *errno* when end-of-directory is encountered.

- 36831 An error or signal indicating that a directory has changed while open was considered but  
36832 rejected.
- 36833 The thread-safe version of the directory reading function returns values in a user-supplied buffer  
36834 instead of possibly using a static data area that may be overwritten by each call. Either the  
36835 {NAME\_MAX} compile-time constant or the corresponding *pathconf()* option can be used to  
36836 determine the maximum sizes of returned pathnames.
- 36837 **FUTURE DIRECTIONS**
- 36838 None.
- 36839 **SEE ALSO**
- 36840 *closedir()*, *lstat()*, *opendir()*, *rewinddir()*, *symlink()*, the Base Definitions volume of  
36841 IEEE Std 1003.1-2001, <**dirent.h**>, <**sys/types.h**>
- 36842 **CHANGE HISTORY**
- 36843 First released in Issue 2.
- 36844 **Issue 5**
- 36845 Large File Summit extensions are added.
- 36846 The *readdir\_r()* function is included for alignment with the POSIX Threads Extension.
- 36847 A note indicating that the *readdir()* function need not be reentrant is added to the  
36848 DESCRIPTION.
- 36849 **Issue 6**
- 36850 The *readdir\_r()* function is marked as part of the Thread-Safe Functions option.
- 36851 The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir\_r()*.
- 36852 The Open Group Corrigendum U026/8 is applied, clarifying the wording of the successful  
36853 return for the *readdir\_r()* function.
- 36854 The following new requirements on POSIX implementations derive from alignment with the  
36855 Single UNIX Specification:
- 36856 • The requirement to include <**sys/types.h**> has been removed. Although <**sys/types.h**> was  
36857 required for conforming implementations of previous POSIX specifications, it was not  
36858 required for UNIX applications.
  - 36859 • A statement is added to the DESCRIPTION indicating the disposition of certain fields in  
36860 **struct dirent** when an entry refers to a symbolic link.
  - 36861 • The [EOVERFLOW] mandatory error condition is added. This change is to support large  
36862 files.
  - 36863 • The [ENOENT] optional error condition is added.
- 36864 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
36865 its avoidance of possibly using a static data area.
- 36866 The **restrict** keyword is added to the *readdir\_r()* prototype for alignment with the  
36867 ISO/IEC 9899:1999 standard.
- 36868 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/50 is applied, replacing the EXAMPLES  
36869 section with a new example. |

36870 **NAME**

36871 readlink — read the contents of a symbolic link

36872 **SYNOPSIS**

36873 #include &lt;unistd.h&gt;

36874 ssize\_t readlink(const char \*restrict path, char \*restrict buf,  
36875 size\_t bufsize);36876 **DESCRIPTION**36877 The *readlink()* function shall place the contents of the symbolic link referred to by *path* in the  
36878 buffer *buf* which has size *bufsize*. If the number of bytes in the symbolic link is less than *bufsize*,  
36879 the contents of the remainder of *buf* are unspecified. If the *buf* argument is not large enough to  
36880 contain the link content, the first *bufsize* bytes shall be placed in *buf*.36881 If the value of *bufsize* is greater than {SSIZE\_MAX}, the result is implementation-defined.36882 **RETURN VALUE**36883 Upon successful completion, *readlink()* shall return the count of bytes placed in the buffer.  
36884 Otherwise, it shall return a value of -1, leave the buffer unchanged, and set *errno* to indicate the  
36885 error.36886 **ERRORS**36887 The *readlink()* function shall fail if:36888 [EACCES] Search permission is denied for a component of the path prefix of *path*.36889 [EINVAL] The *path* argument names a file that is not a symbolic link.

36890 [EIO] An I/O error occurred while reading from the file system.

36891 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
36892 argument.

36893 [ENAMETOOLONG]

36894 The length of the *path* argument exceeds {PATH\_MAX} or a pathname  
36895 component is longer than {NAME\_MAX}.36896 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

36897 [ENOTDIR] A component of the path prefix is not a directory.

36898 The *readlink()* function may fail if:

36899 [EACCES] Read permission is denied for the directory.

36900 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
36901 resolution of the *path* argument.

36902 [ENAMETOOLONG]

36903 As a result of encountering a symbolic link in resolution of the *path* argument,  
36904 the length of the substituted pathname string exceeded {PATH\_MAX}.

36905 **EXAMPLES**36906 **Reading the Name of a Symbolic Link**

36907 The following example shows how to read the name of a symbolic link named `/modules/pass1`.

```
36908 #include <unistd.h>
36909 char buf[1024];
36910 ssize_t len;
36911 ...
36912 if ((len = readlink("/modules/pass1", buf, sizeof(buf)-1)) != -1)
36913 buf[len] = '\0';
```

36914 **APPLICATION USAGE**

36915 Conforming applications should not assume that the returned contents of the symbolic link are  
36916 null-terminated.

36917 **RATIONALE**

36918 Since IEEE Std 1003.1-2001 does not require any association of file times with symbolic links,  
36919 there is no requirement that file times be updated by `readlink()`. The type associated with `bufsiz`  
36920 is a `size_t` in order to be consistent with both the ISO C standard and the definition of `read()`.  
36921 The behavior specified for `readlink()` when `bufsiz` is zero represents historical practice. For this  
36922 case, the standard developers considered a change whereby `readlink()` would return the number  
36923 of non-null bytes contained in the symbolic link with the buffer `buf` remaining unchanged;  
36924 however, since the `stat` structure member `st_size` value can be used to determine the size of  
36925 buffer necessary to contain the contents of the symbolic link as returned by `readlink()`, this  
36926 proposal was rejected, and the historical practice retained.

36927 **FUTURE DIRECTIONS**

36928 None.

36929 **SEE ALSO**

36930 `lstat()`, `stat()`, `symlink()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<unistd.h>`

36931 **CHANGE HISTORY**

36932 First released in Issue 4, Version 2.

36933 **Issue 5**

36934 Moved from X/OPEN UNIX extension to BASE.

36935 **Issue 6**

36936 The return type is changed to `ssize_t`, to align with the IEEE P1003.1a draft standard.

36937 The following new requirements on POSIX implementations derive from alignment with the  
36938 Single UNIX Specification:

- 36939 • This function is made mandatory.
- 36940 • In this function it is possible for the return value to exceed the range of the type `ssize_t` (since  
36941 `size_t` has a larger range of positive values than `ssize_t`). A sentence restricting the size of  
36942 the `size_t` object is added to the description to resolve this conflict.

36943 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 36944 • The FUTURE DIRECTIONS section is changed to None.

36945 The following changes were made to align with the IEEE P1003.1a draft standard:

- 36946 • The [ELOOP] optional error condition is added.

36947  
36948

The **restrict** keyword is added to the *readlink()* prototype for alignment with the ISO/IEC 9899:1999 standard.

36949 **NAME**

36950 readv — read a vector

36951 **SYNOPSIS**36952 XSI `#include <sys/uio.h>`36953 `ssize_t readv(int fildes, const struct iovec *iovcnt, int iovcnt);`

36954

36955 **DESCRIPTION**

36956 The `readv()` function shall be equivalent to `read()`, except as described below. The `readv()`  
 36957 function shall place the input data into the `iovcnt` buffers specified by the members of the `iovcnt`  
 36958 array: `iovcnt[0]`, `iovcnt[1]`, ..., `iovcnt[iovcnt-1]`. The `iovcnt` argument is valid if greater than 0 and less than  
 36959 or equal to `{IOV_MAX}`.

36960 Each `iovcnt` entry specifies the base address and length of an area in memory where data should  
 36961 be placed. The `readv()` function shall always fill an area completely before proceeding to the  
 36962 next.

36963 Upon successful completion, `readv()` shall mark for update the `st_atime` field of the file.

36964 **RETURN VALUE**36965 Refer to `read()`.36966 **ERRORS**36967 Refer to `read()`.36968 In addition, the `readv()` function shall fail if:36969 `[EINVAL]` The sum of the `iovcnt[i].iov_len` values in the `iovcnt` array overflowed an `ssize_t`.36970 The `readv()` function may fail if:36971 `[EINVAL]` The `iovcnt` argument was less than or equal to 0, or greater than `{IOV_MAX}`.36972 **EXAMPLES**36973 **Reading Data into an Array**

36974 The following example reads data from the file associated with the file descriptor `fd` into the  
 36975 buffers specified by members of the `iovcnt` array.

```

36976 #include <sys/types.h>
36977 #include <sys/uio.h>
36978 #include <unistd.h>
36979 ...
36980 ssize_t bytes_read;
36981 int fd;
36982 char buf0[20];
36983 char buf1[30];
36984 char buf2[40];
36985 int iocnt;
36986 struct iovec iov[3];

36987 iov[0].iov_base = buf0;
36988 iov[0].iov_len = sizeof(buf0);
36989 iov[1].iov_base = buf1;
36990 iov[1].iov_len = sizeof(buf1);
36991 iov[2].iov_base = buf2;
36992 iov[2].iov_len = sizeof(buf2);

```

```
36993 ...
36994 iovcnt = sizeof(iov) / sizeof(struct iovec);
36995 bytes_read = readv(fd, iov, iovcnt);
36996 ...

36997 APPLICATION USAGE
36998 None.

36999 RATIONALE
37000 Refer to read().

37001 FUTURE DIRECTIONS
37002 None.

37003 SEE ALSO
37004 read(), writenv(), the Base Definitions volume of IEEE Std 1003.1-2001, <sys/uio.h>

37005 CHANGE HISTORY
37006 First released in Issue 4, Version 2.

37007 Issue 6
37008 Split out from the read() reference page.
```

37009 **NAME**

37010           realloc — memory reallocator

37011 **SYNOPSIS**

37012           #include &lt;stdlib.h&gt;

37013           void \*realloc(void \*ptr, size\_t size);

37014 **DESCRIPTION**

37015 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
37016 conflict between the requirements described here and the ISO C standard is unintentional. This  
37017 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

37018       The *realloc()* function shall change the size of the memory object pointed to by *ptr* to the size  
37019 specified by *size*. The contents of the object shall remain unchanged up to the lesser of the new  
37020 and old sizes. If the new size of the memory object would require movement of the object, the  
37021 space for the previous instantiation of the object is freed. If the new size is larger, the contents of  
37022 the newly allocated portion of the object are unspecified. If *size* is 0 and *ptr* is not a null pointer,  
37023 the object pointed to is freed. If the space cannot be allocated, the object shall remain unchanged.

37024       If *ptr* is a null pointer, *realloc()* shall be equivalent to *malloc()* for the specified size.

37025       If *ptr* does not match a pointer returned earlier by *calloc()*, *malloc()*, or *realloc()* or if the space has  
37026 previously been deallocated by a call to *free()* or *realloc()*, the behavior is undefined.

37027       The order and contiguity of storage allocated by successive calls to *realloc()* is unspecified. The  
37028 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to  
37029 a pointer to any type of object and then used to access such an object in the space allocated (until  
37030 the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object  
37031 disjoint from any other object. The pointer returned shall point to the start (lowest byte address)  
37032 of the allocated space. If the space cannot be allocated, a null pointer shall be returned.

37033 **RETURN VALUE**

37034       Upon successful completion with a *size* not equal to 0, *realloc()* shall return a pointer to the  
37035 (possibly moved) allocated space. If *size* is 0, either a null pointer or a unique pointer that can be  
37036 successfully passed to *free()* shall be returned. If there is not enough available memory, *realloc()*  
37037 **CX** shall return a null pointer and set *errno* to [ENOMEM].

37038 **ERRORS**37039       The *realloc()* function shall fail if:

37040 **CX**       [ENOMEM]       Insufficient memory is available.

37041 **EXAMPLES**

37042       None.

37043 **APPLICATION USAGE**

37044       None.

37045 **RATIONALE**

37046       None.

37047 **FUTURE DIRECTIONS**

37048       None.

37049 **SEE ALSO**37050       *calloc()*, *free()*, *malloc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>

37051 **CHANGE HISTORY**

37052 First released in Issue 1. Derived from Issue 1 of the SVID.

37053 **Issue 6**

37054 Extensions beyond the ISO C standard are marked.

37055 The following new requirements on POSIX implementations derive from alignment with the  
37056 Single UNIX Specification:

- 37057 • In the RETURN VALUE section, if there is not enough available memory, the setting of *errno*  
37058 to [ENOMEM] is added.
- 37059 • The [ENOMEM] error condition is added.

## 37060 NAME

37061        realpath — resolve a pathname

## 37062 SYNOPSIS

37063 XSI        #include &lt;stdlib.h&gt;

37064        char \*realpath(const char \*restrict *file\_name*,  
37065                      char \*restrict *resolved\_name*);

37066

## 37067 DESCRIPTION

37068        The *realpath()* function shall derive, from the pathname pointed to by *file\_name*, an absolute  
37069        pathname that names the same file, whose resolution does not involve '.', '..', or symbolic  
37070        links. The generated pathname shall be stored as a null-terminated string, up to a maximum of  
37071        {PATH\_MAX} bytes, in the buffer pointed to by *resolved\_name*.37072        If *resolved\_name* is a null pointer, the behavior of *realpath()* is implementation-defined.

## 37073 RETURN VALUE

37074        Upon successful completion, *realpath()* shall return a pointer to the resolved name. Otherwise,  
37075        *realpath()* shall return a null pointer and set *errno* to indicate the error, and the contents of the  
37076        buffer pointed to by *resolved\_name* are undefined.

## 37077 ERRORS

37078        The *realpath()* function shall fail if:37079        [EACCES]        Read or search permission was denied for a component of *file\_name*.37080        [EINVAL]        The *file\_name* argument is a null pointer.

37081        [EIO]           An error occurred while reading from the file system.

37082        [ELOOP]        A loop exists in symbolic links encountered during resolution of the *path*  
37083        argument.

37084        [ENAMETOOLONG]

37085                      The length of the *file\_name* argument exceeds {PATH\_MAX} or a pathname  
37086                      component is longer than {NAME\_MAX}.37087        [ENOENT]        A component of *file\_name* does not name an existing file or *file\_name* points to  
37088                      an empty string.

37089        [ENOTDIR]       A component of the path prefix is not a directory.

37090        The *realpath()* function may fail if:37091        [ELOOP]        More than {SYMLOOP\_MAX} symbolic links were encountered during  
37092                      resolution of the *path* argument.

37093        [ENAMETOOLONG]

37094                      Pathname resolution of a symbolic link produced an intermediate result  
37095                      whose length exceeds {PATH\_MAX}.

37096        [ENOMEM]        Insufficient storage space is available.

37097 **EXAMPLES**37098 **Generating an Absolute Pathname**

37099 The following example generates an absolute pathname for the file identified by the *symlinkpath*  
 37100 argument. The generated pathname is stored in the *actualpath* array.

```
37101 #include <stdlib.h>
37102 ...
37103 char *symlinkpath = "/tmp/symlink/file";
37104 char actualpath [PATH_MAX+1];
37105 char *ptr;
37106 ptr = realpath(symlinkpath, actualpath);
```

37107 **APPLICATION USAGE**

37108 None.

37109 **RATIONALE**

37110 Since the maximum pathname length is arbitrary unless {PATH\_MAX} is defined, an application  
 37111 generally cannot supply a *resolved\_name* buffer with size {{PATH\_MAX}+1}.

37112 **FUTURE DIRECTIONS**

37113 In the future, passing a null pointer to *realpath()* for the *resolved\_name* argument may be defined  
 37114 to have *realpath()* allocate space for the generated pathname.

37115 **SEE ALSO**

37116 *getcwd()*, *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>

37117 **CHANGE HISTORY**

37118 First released in Issue 4, Version 2.

37119 **Issue 5**

37120 Moved from X/OPEN UNIX extension to BASE.

37121 **Issue 6**

37122 The **restrict** keyword is added to the *realpath()* prototype for alignment with the  
 37123 ISO/IEC 9899:1999 standard.

37124 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
 37125 [ELOOP] error condition is added.

37126 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/51 is applied, adding new text to the  
 37127 DESCRIPTION for the case when *resolved\_name* is a null pointer, changing the [EINVAL] error  
 37128 text, adding text to the RATIONALE, and adding text to FUTURE DIRECTIONS.

37129 **NAME**

37130           recv — receive a message from a connected socket

37131 **SYNOPSIS**

37132           #include <sys/socket.h>

37133           ssize\_t recv(int *socket*, void \**buffer*, size\_t *length*, int *flags*);

37134 **DESCRIPTION**

37135           The *recv()* function shall receive a message from a connection-mode or connectionless-mode  
37136           socket. It is normally used with connected sockets because it does not permit the application to  
37137           retrieve the source address of received data.

37138           The *recv()* function takes the following arguments:

37139           *socket*           Specifies the socket file descriptor.

37140           *buffer*           Points to a buffer where the message should be stored.

37141           *length*          Specifies the length in bytes of the buffer pointed to by the *buffer* argument.

37142           *flags*           Specifies the type of message reception. Values of this argument are formed by  
37143           logically OR'ing zero or more of the following values:

37144                   MSG\_PEEK       Peeks at an incoming message. The data is treated as unread and  
37145                   the next *recv()* or similar function shall still return this data.

37146                   MSG\_OOB       Requests out-of-band data. The significance and semantics of  
37147                   out-of-band data are protocol-specific.

37148                   MSG\_WAITALL   On SOCK\_STREAM sockets this requests that the function block  
37149                   until the full amount of data can be returned. The function may  
37150                   return the smaller amount of data if the socket is a message-  
37151                   based socket, if a signal is caught, if the connection is  
37152                   terminated, if MSG\_PEEK was specified, or if an error is pending  
37153                   for the socket.

37154           The *recv()* function shall return the length of the message written to the buffer pointed to by the  
37155           *buffer* argument. For message-based sockets, such as SOCK\_DGRAM and SOCK\_SEQPACKET,  
37156           the entire message shall be read in a single operation. If a message is too long to fit in the  
37157           supplied buffer, and MSG\_PEEK is not set in the *flags* argument, the excess bytes shall be  
37158           discarded. For stream-based sockets, such as SOCK\_STREAM, message boundaries shall be  
37159           ignored. In this case, data shall be returned to the user as soon as it becomes available, and no  
37160           data shall be discarded.

37161           If the MSG\_WAITALL flag is not set, data shall be returned only up to the end of the first  
37162           message.

37163           If no messages are available at the socket and O\_NONBLOCK is not set on the socket's file  
37164           descriptor, *recv()* shall block until a message arrives. If no messages are available at the socket  
37165           and O\_NONBLOCK is set on the socket's file descriptor, *recv()* shall fail and set *errno* to  
37166           [EAGAIN] or [EWOULDBLOCK].

37167 **RETURN VALUE**

37168           Upon successful completion, *recv()* shall return the length of the message in bytes. If no  
37169           messages are available to be received and the peer has performed an orderly shutdown, *recv()*  
37170           shall return 0. Otherwise, -1 shall be returned and *errno* set to indicate the error.

37171 **ERRORS**

- 37172 The *recv()* function shall fail if:
- 37173 [EAGAIN] or [EWOULDBLOCK]
- 37174 The socket's file descriptor is marked O\_NONBLOCK and no data is waiting  
37175 to be received; or MSG\_OOB is set and no out-of-band data is available and  
37176 either the socket's file descriptor is marked O\_NONBLOCK or the socket does  
37177 not support blocking to await out-of-band data.
- 37178 [EBADF] The *socket* argument is not a valid file descriptor.
- 37179 [ECONNRESET] A connection was forcibly closed by a peer.
- 37180 [EINTR] The *recv()* function was interrupted by a signal that was caught, before any  
37181 data was available.
- 37182 [EINVAL] The MSG\_OOB flag is set and no out-of-band data is available.
- 37183 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.
- 37184 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 37185 [EOPNOTSUPP] The specified flags are not supported for this socket type or protocol.
- 37186 [ETIMEDOUT] The connection timed out during connection establishment, or due to a  
37187 transmission timeout on active connection.
- 37188 The *recv()* function may fail if:
- 37189 [EIO] An I/O error occurred while reading from or writing to the file system.
- 37190 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 37191 [ENOMEM] Insufficient memory was available to fulfill the request.

37192 **EXAMPLES**

37193 None.

37194 **APPLICATION USAGE**

- 37195 The *recv()* function is equivalent to *recvfrom()* with a zero *address\_len* argument, and to *read()* if  
37196 no flags are used.
- 37197 The *select()* and *poll()* functions can be used to determine when data is available to be received.

37198 **RATIONALE**

37199 None.

37200 **FUTURE DIRECTIONS**

37201 None.

37202 **SEE ALSO**

- 37203 *poll()*, *read()*, *recvmsg()*, *recvfrom()*, *select()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*,  
37204 *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**sys/socket.h**>

37205 **CHANGE HISTORY**

37206 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

## 37207 NAME

37208 recvfrom — receive a message from a socket

## 37209 SYNOPSIS

37210 #include &lt;sys/socket.h&gt;

37211 ssize\_t recvfrom(int socket, void \*restrict buffer, size\_t length,

37212 int flags, struct sockaddr \*restrict address,

37213 socklen\_t \*restrict address\_len);

## 37214 DESCRIPTION

37215 The *recvfrom()* function shall receive a message from a connection-mode or connectionless-mode  
 37216 socket. It is normally used with connectionless-mode sockets because it permits the application  
 37217 to retrieve the source address of received data.

37218 The *recvfrom()* function takes the following arguments:

|       |                    |                                                                                                                                                                                                                                                                                                                                                                                   |
|-------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 37219 | <i>socket</i>      | Specifies the socket file descriptor.                                                                                                                                                                                                                                                                                                                                             |
| 37220 | <i>buffer</i>      | Points to the buffer where the message should be stored.                                                                                                                                                                                                                                                                                                                          |
| 37221 | <i>length</i>      | Specifies the length in bytes of the buffer pointed to by the <i>buffer</i> argument.                                                                                                                                                                                                                                                                                             |
| 37222 | <i>flags</i>       | Specifies the type of message reception. Values of this argument are formed<br>37223 by logically OR'ing zero or more of the following values:                                                                                                                                                                                                                                    |
| 37224 | MSG_PEEK           | Peeks at an incoming message. The data is treated as unread<br>37225 and the next <i>recvfrom()</i> or similar function shall still return<br>37226 this data.                                                                                                                                                                                                                    |
| 37227 | MSG_OOB            | Requests out-of-band data. The significance and semantics<br>37228 of out-of-band data are protocol-specific.                                                                                                                                                                                                                                                                     |
| 37229 | MSG_WAITALL        | On SOCK_STREAM sockets this requests that the function<br>37230 block until the full amount of data can be returned. The<br>37231 function may return the smaller amount of data if the socket<br>37232 is a message-based socket, if a signal is caught, if the<br>37233 connection is terminated, if MSG_PEEK was specified, or if<br>37234 an error is pending for the socket. |
| 37235 | <i>address</i>     | A null pointer, or points to a <b>sockaddr</b> structure in which the sending address<br>37236 is to be stored. The length and format of the address depend on the address<br>37237 family of the socket.                                                                                                                                                                         |
| 37238 | <i>address_len</i> | Specifies the length of the <b>sockaddr</b> structure pointed to by the <i>address</i><br>37239 argument.                                                                                                                                                                                                                                                                         |

37240 The *recvfrom()* function shall return the length of the message written to the buffer pointed to by  
 37241 RS the *buffer* argument. For message-based sockets, such as SOCK\_RAW, SOCK\_DGRAM, and  
 37242 SOCK\_SEQPACKET, the entire message shall be read in a single operation. If a message is too  
 37243 long to fit in the supplied buffer, and MSG\_PEEK is not set in the *flags* argument, the excess  
 37244 bytes shall be discarded. For stream-based sockets, such as SOCK\_STREAM, message  
 37245 boundaries shall be ignored. In this case, data shall be returned to the user as soon as it becomes  
 37246 available, and no data shall be discarded.

37247 If the MSG\_WAITALL flag is not set, data shall be returned only up to the end of the first  
 37248 message.

37249 Not all protocols provide the source address for messages. If the *address* argument is not a null  
 37250 pointer and the protocol provides the source address of messages, the source address of the

- 37251 received message shall be stored in the **sockaddr** structure pointed to by the *address* argument,  
 37252 and the length of this address shall be stored in the object pointed to by the *address\_len*  
 37253 argument.
- 37254 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,  
 37255 the stored address shall be truncated.
- 37256 If the *address* argument is not a null pointer and the protocol does not provide the source address  
 37257 of messages, the value stored in the object pointed to by *address* is unspecified.
- 37258 If no messages are available at the socket and O\_NONBLOCK is not set on the socket's file  
 37259 descriptor, *recvfrom()* shall block until a message arrives. If no messages are available at the  
 37260 socket and O\_NONBLOCK is set on the socket's file descriptor, *recvfrom()* shall fail and set *errno*  
 37261 to [EAGAIN] or [EWOULDBLOCK].
- 37262 **RETURN VALUE**
- 37263 Upon successful completion, *recvfrom()* shall return the length of the message in bytes. If no  
 37264 messages are available to be received and the peer has performed an orderly shutdown,  
 37265 *recvfrom()* shall return 0. Otherwise, the function shall return -1 and set *errno* to indicate the  
 37266 error.
- 37267 **ERRORS**
- 37268 The *recvfrom()* function shall fail if:
- 37269 [EAGAIN] or [EWOULDBLOCK]  
 37270 The socket's file descriptor is marked O\_NONBLOCK and no data is waiting  
 37271 to be received; or MSG\_OOB is set and no out-of-band data is available and  
 37272 either the socket's file descriptor is marked O\_NONBLOCK or the socket does  
 37273 not support blocking to await out-of-band data.
- 37274 [EBADF] The *socket* argument is not a valid file descriptor.
- 37275 [ECONNRESET] A connection was forcibly closed by a peer.
- 37276 [EINTR] A signal interrupted *recvfrom()* before any data was available.
- 37277 [EINVAL] The MSG\_OOB flag is set and no out-of-band data is available.
- 37278 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.
- 37279 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 37280 [EOPNOTSUPP] The specified flags are not supported for this socket type.
- 37281 [ETIMEDOUT] The connection timed out during connection establishment, or due to a  
 37282 transmission timeout on active connection.
- 37283 The *recvfrom()* function may fail if:
- 37284 [EIO] An I/O error occurred while reading from or writing to the file system.
- 37285 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 37286 [ENOMEM] Insufficient memory was available to fulfill the request.

37287 **EXAMPLES**

37288 None.

37289 **APPLICATION USAGE**37290 The *select()* and *poll()* functions can be used to determine when data is available to be received.37291 **RATIONALE**

37292 None.

37293 **FUTURE DIRECTIONS**

37294 None.

37295 **SEE ALSO**37296 *poll()*, *read()*, *recv()*, *recvmsg()*, *select()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*, *write()*,37297 the Base Definitions volume of IEEE Std 1003.1-2001, <**sys/socket.h**>37298 **CHANGE HISTORY**

37299 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

## 37300 NAME

37301 recvmsg — receive a message from a socket

## 37302 SYNOPSIS

37303 #include &lt;sys/socket.h&gt;

37304 ssize\_t recvmsg(int socket, struct msghdr \*message, int flags);

## 37305 DESCRIPTION

37306 The *recvmsg()* function shall receive a message from a connection-mode or connectionless-mode  
 37307 socket. It is normally used with connectionless-mode sockets because it permits the application  
 37308 to retrieve the source address of received data.

37309 The *recvmsg()* function takes the following arguments:

|       |                |                                                                                                                                                                                                                                                                                                                                      |
|-------|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 37310 | <i>socket</i>  | Specifies the socket file descriptor.                                                                                                                                                                                                                                                                                                |
| 37311 | <i>message</i> | Points to a <b>msghdr</b> structure, containing both the buffer to store the source address and the buffers for the incoming message. The length and format of the address depend on the address family of the socket. The <i>msg_flags</i> member is ignored on input, but may contain meaningful values on output.                 |
| 37312 |                |                                                                                                                                                                                                                                                                                                                                      |
| 37313 |                |                                                                                                                                                                                                                                                                                                                                      |
| 37314 |                |                                                                                                                                                                                                                                                                                                                                      |
| 37315 | <i>flags</i>   | Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:                                                                                                                                                                                                |
| 37316 |                |                                                                                                                                                                                                                                                                                                                                      |
| 37317 | MSG_OOB        | Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.                                                                                                                                                                                                                                 |
| 37318 |                |                                                                                                                                                                                                                                                                                                                                      |
| 37319 | MSG_PEEK       | Peeks at the incoming message.                                                                                                                                                                                                                                                                                                       |
| 37320 | MSG_WAITALL    | On SOCK_STREAM sockets this requests that the function block until the full amount of data can be returned. The function may return the smaller amount of data if the socket is a message-based socket, if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket. |
| 37321 |                |                                                                                                                                                                                                                                                                                                                                      |
| 37322 |                |                                                                                                                                                                                                                                                                                                                                      |
| 37323 |                |                                                                                                                                                                                                                                                                                                                                      |
| 37324 |                |                                                                                                                                                                                                                                                                                                                                      |
| 37325 |                |                                                                                                                                                                                                                                                                                                                                      |

37326 The *recvmsg()* function shall receive messages from unconnected or connected sockets and shall  
 37327 return the length of the message.

37328 The *recvmsg()* function shall return the total length of the message. For message-based sockets,  
 37329 such as SOCK\_DGRAM and SOCK\_SEQPACKET, the entire message shall be read in a single  
 37330 operation. If a message is too long to fit in the supplied buffers, and MSG\_PEEK is not set in the  
 37331 *flags* argument, the excess bytes shall be discarded, and MSG\_TRUNC shall be set in the  
 37332 *msg\_flags* member of the **msghdr** structure. For stream-based sockets, such as SOCK\_STREAM,  
 37333 message boundaries shall be ignored. In this case, data shall be returned to the user as soon as it  
 37334 becomes available, and no data shall be discarded.

37335 If the MSG\_WAITALL flag is not set, data shall be returned only up to the end of the first  
 37336 message.

37337 If no messages are available at the socket and O\_NONBLOCK is not set on the socket's file  
 37338 descriptor, *recvmsg()* shall block until a message arrives. If no messages are available at the  
 37339 socket and O\_NONBLOCK is set on the socket's file descriptor, the *recvmsg()* function shall fail  
 37340 and set *errno* to [EAGAIN] or [EWOULDBLOCK].

37341 In the **msghdr** structure, the *msg\_name* and *msg\_namelen* members specify the source address if  
 37342 the socket is unconnected. If the socket is connected, the *msg\_name* and *msg\_namelen* members  
 37343 shall be ignored. The *msg\_name* member may be a null pointer if no names are desired or  
 37344 required. The *msg\_iov* and *msg\_iovlen* fields are used to specify where the received data shall be

37345 stored. *msg\_iov* points to an array of **iovec** structures; *msg\_iovlen* shall be set to the dimension of  
 37346 this array. In each **iovec** structure, the *iov\_base* field specifies a storage area and the *iov\_len* field  
 37347 gives its size in bytes. Each storage area indicated by *msg\_iov* is filled with received data in turn  
 37348 until all of the received data is stored or all of the areas have been filled.

37349 Upon successful completion, the *msg\_flags* member of the message header shall be the bitwise-  
 37350 inclusive OR of all of the following flags that indicate conditions detected for the received  
 37351 message:

37352 MSG\_EOR End-of-record was received (if supported by the protocol).

37353 MSG\_OOB Out-of-band data was received.

37354 MSG\_TRUNC Normal data was truncated.

37355 MSG\_CTRUNC Control data was truncated.

#### 37356 RETURN VALUE

37357 Upon successful completion, *recvmsg()* shall return the length of the message in bytes. If no  
 37358 messages are available to be received and the peer has performed an orderly shutdown,  
 37359 *recvmsg()* shall return 0. Otherwise,  $-1$  shall be returned and *errno* set to indicate the error.

#### 37360 ERRORS

37361 The *recvmsg()* function shall fail if:

37362 [EAGAIN] or [EWOULDBLOCK]

37363 The socket's file descriptor is marked O\_NONBLOCK and no data is waiting  
 37364 to be received; or MSG\_OOB is set and no out-of-band data is available and  
 37365 either the socket's file descriptor is marked O\_NONBLOCK or the socket does  
 37366 not support blocking to await out-of-band data.

37367 [EBADF] The *socket* argument is not a valid open file descriptor.

37368 [ECONNRESET] A connection was forcibly closed by a peer.

37369 [EINTR] This function was interrupted by a signal before any data was available.

37370 [EINVAL] The sum of the *iov\_len* values overflows a **ssize\_t**, or the MSG\_OOB flag is set  
 37371 and no out-of-band data is available.

37372 [EMSGSIZE] The *msg\_iovlen* member of the **msghdr** structure pointed to by *message* is less  
 37373 than or equal to 0, or is greater than {IOV\_MAX}.

37374 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

37375 [ENOTSOCK] The *socket* argument does not refer to a socket.

37376 [EOPNOTSUPP] The specified flags are not supported for this socket type.

37377 [ETIMEDOUT] The connection timed out during connection establishment, or due to a  
 37378 transmission timeout on active connection.

37379 The *recvmsg()* function may fail if:

37380 [EIO] An I/O error occurred while reading from or writing to the file system.

37381 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

37382 [ENOMEM] Insufficient memory was available to fulfill the request.

37383 **EXAMPLES**

37384 None.

37385 **APPLICATION USAGE**37386 The *select()* and *poll()* functions can be used to determine when data is available to be received.37387 **RATIONALE**

37388 None.

37389 **FUTURE DIRECTIONS**

37390 None.

37391 **SEE ALSO**37392 *poll()*, *recv()*, *recvfrom()*, *select()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*, the Base

37393 Definitions volume of IEEE Std 1003.1-2001, &lt;sys/socket.h&gt;

37394 **CHANGE HISTORY**

37395 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

37396 **NAME**

37397 regcomp, regerror, regex, regfree — regular expression matching

37398 **SYNOPSIS**

```
37399 #include <regex.h>

37400 int regcomp(regex_t *restrict preg, const char *restrict pattern,
37401 int cflags);
37402 size_t regerror(int errcode, const regex_t *restrict preg,
37403 char *restrict errbuf, size_t errbuf_size);
37404 int regex(const regex_t *restrict preg, const char *restrict string,
37405 size_t nmatch, regmatch_t pmatch[restrict], int eflags);
37406 void regfree(regex_t *preg);
```

37407 **DESCRIPTION**

37408 These functions interpret *basic* and *extended* regular expressions as described in the Base  
37409 Definitions volume of IEEE Std 1003.1-2001, Chapter 9, Regular Expressions.

37410 The **regex\_t** structure is defined in **<regex.h>** and contains at least the following member:

| Member Type | Member Name | Description                             |
|-------------|-------------|-----------------------------------------|
| size_t      | re_nsub     | Number of parenthesized subexpressions. |

37414 The **regmatch\_t** structure is defined in **<regex.h>** and contains at least the following members:

| Member Type | Member Name | Description                                                                                |
|-------------|-------------|--------------------------------------------------------------------------------------------|
| regoff_t    | rm_so       | Byte offset from start of <i>string</i> to start of substring.                             |
| regoff_t    | rm_eo       | Byte offset from start of <i>string</i> of the first character after the end of substring. |

37420 The *regcomp()* function shall compile the regular expression contained in the string pointed to by  
37421 the *pattern* argument and place the results in the structure pointed to by *preg*. The *cflags*  
37422 argument is the bitwise-inclusive OR of zero or more of the following flags, which are defined in  
37423 the **<regex.h>** header:

- 37424 REG\_EXTENDED Use Extended Regular Expressions.
- 37425 REG\_ICASE Ignore case in match. (See the Base Definitions volume of  
37426 IEEE Std 1003.1-2001, Chapter 9, Regular Expressions.)
- 37427 REG\_NOSUB Report only success/fail in *regex()*.
- 37428 REG\_NEWLINE Change the handling of <newline>s, as described in the text.

37429 The default regular expression type for *pattern* is a Basic Regular Expression. The application can  
37430 specify Extended Regular Expressions using the REG\_EXTENDED *cflags* flag.

37431 If the REG\_NOSUB flag was not set in *cflags*, then *regcomp()* shall set *re\_nsub* to the number of  
37432 parenthesized subexpressions (delimited by "\(\)" in basic regular expressions or "( )" in  
37433 extended regular expressions) found in *pattern*.

37434 The *regex()* function compares the null-terminated string specified by *string* with the compiled  
37435 regular expression *preg* initialized by a previous call to *regcomp()*. If it finds a match, *regex()*  
37436 shall return 0; otherwise, it shall return non-zero indicating either no match or an error. The  
37437 *eflags* argument is the bitwise-inclusive OR of zero or more of the following flags, which are  
37438 defined in the **<regex.h>** header:

37439 REG\_NOTBOL The first character of the string pointed to by *string* is not the beginning of the  
 37440 line. Therefore, the circumflex character ('^'), when taken as a special  
 37441 character, shall not match the beginning of *string*.

37442 REG\_NOTEOL The last character of the string pointed to by *string* is not the end of the line.  
 37443 Therefore, the dollar sign ('\$'), when taken as a special character, shall not  
 37444 match the end of *string*.

37445 If *nmatch* is 0 or REG\_NOSUB was set in the *cflags* argument to *regcomp()*, then *regexec()* shall  
 37446 ignore the *pmatch* argument. Otherwise, the application shall ensure that the *pmatch* argument  
 37447 points to an array with at least *nmatch* elements, and *regexec()* shall fill in the elements of that  
 37448 array with offsets of the substrings of *string* that correspond to the parenthesized subexpressions  
 37449 of *pattern*: *pmatch[i].rm\_so* shall be the byte offset of the beginning and *pmatch[i].rm\_eo* shall be  
 37450 one greater than the byte offset of the end of substring *i*. (Subexpression *i* begins at the *i*th  
 37451 matched open parenthesis, counting from 1.) Offsets in *pmatch[0]* identify the substring that  
 37452 corresponds to the entire regular expression. Unused elements of *pmatch* up to *pmatch[nmatch-1]*  
 37453 shall be filled with -1. If there are more than *nmatch* subexpressions in *pattern* (*pattern* itself  
 37454 counts as a subexpression), then *regexec()* shall still do the match, but shall record only the first  
 37455 *nmatch* substrings.

37456 When matching a basic or extended regular expression, any given parenthesized subexpression  
 37457 of *pattern* might participate in the match of several different substrings of *string*, or it might not  
 37458 match any substring even though the pattern as a whole did match. The following rules shall be  
 37459 used to determine which substrings to report in *pmatch* when matching regular expressions:

37460 1. If subexpression *i* in a regular expression is not contained within another subexpression,  
 37461 and it participated in the match several times, then the byte offsets in *pmatch[i]* shall  
 37462 delimit the last such match.

37463 2. If subexpression *i* is not contained within another subexpression, and it did not participate  
 37464 in an otherwise successful match, the byte offsets in *pmatch[i]* shall be -1. A subexpression  
 37465 does not participate in the match when:

37466 ' \* ' or "\{\}" appears immediately after the subexpression in a basic regular  
 37467 expression, or ' \* ', ' ? ', or "{ }" appears immediately after the subexpression in an  
 37468 extended regular expression, and the subexpression did not match (matched 0 times)

37469 or:

37470 ' | ' is used in an extended regular expression to select this subexpression or another,  
 37471 and the other subexpression matched.

37472 3. If subexpression *i* is contained within another subexpression *j*, and *i* is not contained  
 37473 within any other subexpression that is contained within *j*, and a match of subexpression *j*  
 37474 is reported in *pmatch[j]*, then the match or non-match of subexpression *i* reported in  
 37475 *pmatch[i]* shall be as described in 1. and 2. above, but within the substring reported in  
 37476 *pmatch[j]* rather than the whole string. The offsets in *pmatch[i]* are still relative to the start  
 37477 of *string*.

37478 4. If subexpression *i* is contained in subexpression *j*, and the byte offsets in *pmatch[j]* are -1,  
 37479 then the pointers in *pmatch[i]* shall also be -1.

37480 5. If subexpression *i* matched a zero-length string, then both byte offsets in *pmatch[i]* shall be  
 37481 the byte offset of the character or null terminator immediately following the zero-length  
 37482 string.

37483 If, when *regexec()* is called, the locale is different from when the regular expression was  
 37484 compiled, the result is undefined.

37485 If REG\_NEWLINE is not set in *cflags*, then a <newline> in *pattern* or *string* shall be treated as an  
 37486 ordinary character. If REG\_NEWLINE is set, then <newline> shall be treated as an ordinary  
 37487 character except as follows:

- 37488 1. A <newline> in *string* shall not be matched by a period outside a bracket expression or by  
 37489 any form of a non-matching list (see the Base Definitions volume of IEEE Std 1003.1-2001,  
 37490 Chapter 9, Regular Expressions).
- 37491 2. A circumflex ('^') in *pattern*, when used to specify expression anchoring (see the Base  
 37492 Definitions volume of IEEE Std 1003.1-2001, Section 9.3.8, BRE Expression Anchoring),  
 37493 shall match the zero-length string immediately after a <newline> in *string*, regardless of  
 37494 the setting of REG\_NOTBOL.
- 37495 3. A dollar sign ('\$') in *pattern*, when used to specify expression anchoring, shall match the  
 37496 zero-length string immediately before a <newline> in *string*, regardless of the setting of  
 37497 REG\_NOTEOL.

37498 The *regfree()* function frees any memory allocated by *regcomp()* associated with *preg*.

37499 The following constants are defined as error return values:

|       |              |                                                                                                                      |
|-------|--------------|----------------------------------------------------------------------------------------------------------------------|
| 37500 | REG_NOMATCH  | <i>regexec()</i> failed to match.                                                                                    |
| 37501 | REG_BADPAT   | Invalid regular expression.                                                                                          |
| 37502 | REG_ECOLLATE | Invalid collating element referenced.                                                                                |
| 37503 | REG_ECTYPE   | Invalid character class type referenced.                                                                             |
| 37504 | REG_EESCAPE  | Trailing '\\' in pattern.                                                                                            |
| 37505 | REG_ESUBREG  | Number in "\digit" invalid or in error.                                                                              |
| 37506 | REG_EBRACK   | "[]" imbalance.                                                                                                      |
| 37507 | REG_EPAREN   | "\(\)" or "()" imbalance.                                                                                            |
| 37508 | REG_EBRACE   | "\{\}" imbalance.                                                                                                    |
| 37509 | REG_BADBR    | Content of "\{\}" invalid: not a number, number too large, more than<br>37510 two numbers, first larger than second. |
| 37511 | REG_ERANGE   | Invalid endpoint in range expression.                                                                                |
| 37512 | REG_ESPACE   | Out of memory.                                                                                                       |
| 37513 | REG_BADRPT   | '?', '*', or '+' not preceded by valid regular expression.                                                           |

37514 The *regerror()* function provides a mapping from error codes returned by *regcomp()* and  
 37515 *regexec()* to unspecified printable strings. It generates a string corresponding to the value of the  
 37516 *errcode* argument, which the application shall ensure is the last non-zero value returned by  
 37517 *regcomp()* or *regexec()* with the given value of *preg*. If *errcode* is not such a value, the content of  
 37518 the generated string is unspecified.

37519 If *preg* is a null pointer, but *errcode* is a value returned by a previous call to *regexec()* or *regcomp()*,  
 37520 the *regerror()* still generates an error string corresponding to the value of *errcode*, but it might not  
 37521 be as detailed under some implementations.

37522 If the *errbuf\_size* argument is not 0, *regerror()* shall place the generated string into the buffer of  
 37523 size *errbuf\_size* bytes pointed to by *errbuf*. If the string (including the terminating null) cannot fit  
 37524 in the buffer, *regerror()* shall truncate the string and null-terminate the result.

37525 If *errbuf\_size* is 0, *regerror()* shall ignore the *errbuf* argument, and return the size of the buffer  
 37526 needed to hold the generated string.

37527 If the *preg* argument to *regexec()* or *regfree()* is not a compiled regular expression returned by  
 37528 *regcomp()*, the result is undefined. A *preg* is no longer treated as a compiled regular expression  
 37529 after it is given to *regfree()*.

#### 37530 RETURN VALUE

37531 Upon successful completion, the *regcomp()* function shall return 0. Otherwise, it shall return an  
 37532 integer value indicating an error as described in <**regex.h**>, and the content of *preg* is undefined.  
 37533 If a code is returned, the interpretation shall be as given in <**regex.h**>.

37534 If *regcomp()* detects an invalid RE, it may return REG\_BADPAT, or it may return one of the error  
 37535 codes that more precisely describes the error.

37536 Upon successful completion, the *regexec()* function shall return 0. Otherwise, it shall return  
 37537 REG\_NOMATCH to indicate no match.

37538 Upon successful completion, the *regerror()* function shall return the number of bytes needed to  
 37539 hold the entire generated string, including the null termination. If the return value is greater than  
 37540 *errbuf\_size*, the string returned in the buffer pointed to by *errbuf* has been truncated.

37541 The *regfree()* function shall not return a value.

#### 37542 ERRORS

37543 No errors are defined.

#### 37544 EXAMPLES

```

37545 #include <regex.h>
37546 /*
37547 * Match string against the extended regular expression in
37548 * pattern, treating errors as no match.
37549 *
37550 * Return 1 for match, 0 for no match.
37551 */
37552 int
37553 match(const char *string, char *pattern)
37554 {
37555 int status;
37556 regex_t re;
37557
37558 if (regcomp(&re, pattern, REG_EXTENDED|REG_NOSUB) != 0) {
37559 return(0); /* Report error. */
37560 }
37561 status = regexec(&re, string, (size_t) 0, NULL, 0);
37562 regfree(&re);
37563 if (status != 0) {
37564 return(0); /* Report error. */
37565 }
37566 return(1);
37567 }
```

37567 The following demonstrates how the REG\_NOTBOL flag could be used with *regexec()* to find all  
 37568 substrings in a line that match a pattern supplied by a user. (For simplicity of the example, very  
 37569 little error checking is done.)

```

37570 (void) regcomp (&re, pattern, 0);
37571 /* This call to regexec() finds the first match on the line. */
37572 error = regexec (&re, &buffer[0], 1, &pm, 0);
37573 while (error == 0) { /* While matches found. */
37574 /* Substring found between pm.rm_so and pm.rm_eo. */
37575 /* This call to regexec() finds the next match. */
37576 error = regexec (&re, buffer + pm.rm_eo, 1, &pm, REG_NOTBOL);
37577 }

```

#### 37578 APPLICATION USAGE

37579 An application could use:

```

37580 regerror(code, preg, (char *)NULL, (size_t)0)

```

37581 to find out how big a buffer is needed for the generated string, *malloc()* a buffer to hold the  
37582 string, and then call *regerror()* again to get the string. Alternatively, it could allocate a fixed,  
37583 static buffer that is big enough to hold most strings, and then use *malloc()* to allocate a larger  
37584 buffer if it finds that this is too small.

37585 To match a pattern as described in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section  
37586 2.13, Pattern Matching Notation, use the *fnmatch()* function.

#### 37587 RATIONALE

37588 The *regexec()* function must fill in all *nmatch* elements of *pmatch*, where *nmatch* and *pmatch* are  
37589 supplied by the application, even if some elements of *pmatch* do not correspond to  
37590 subexpressions in *pattern*. The application writer should note that there is probably no reason  
37591 for using a value of *nmatch* that is larger than *preg->re\_nsub+1*.

37592 The REG\_NEWLINE flag supports a use of RE matching that is needed in some applications like  
37593 text editors. In such applications, the user supplies an RE asking the application to find a line  
37594 that matches the given expression. An anchor in such an RE anchors at the beginning or end of  
37595 any line. Such an application can pass a sequence of <newline>-separated lines to *regexec()* as a  
37596 single long string and specify REG\_NEWLINE to *regcomp()* to get the desired behavior. The  
37597 application must ensure that there are no explicit <newline>s in *pattern* if it wants to ensure that  
37598 any match occurs entirely within a single line.

37599 The REG\_NEWLINE flag affects the behavior of *regexec()*, but it is in the *cflags* parameter to  
37600 *regcomp()* to allow flexibility of implementation. Some implementations will want to generate  
37601 the same compiled RE in *regcomp()* regardless of the setting of REG\_NEWLINE and have  
37602 *regexec()* handle anchors differently based on the setting of the flag. Other implementations will  
37603 generate different compiled REs based on the REG\_NEWLINE.

37604 The REG\_ICASE flag supports the operations taken by the *grep -i* option and the historical  
37605 implementations of *ex* and *vi*. Including this flag will make it easier for application code to be  
37606 written that does the same thing as these utilities.

37607 The substrings reported in *pmatch[]* are defined using offsets from the start of the string rather  
37608 than pointers. Since this is a new interface, there should be no impact on historical  
37609 implementations or applications, and offsets should be just as easy to use as pointers. The  
37610 change to offsets was made to facilitate future extensions in which the string to be searched is  
37611 presented to *regexec()* in blocks, allowing a string to be searched that is not all in memory at  
37612 once.

37613 The type **regoff\_t** is used for the elements of *pmatch[]* to ensure that the application can  
37614 represent either the largest possible array in memory (important for an application conforming  
37615 to the Shell and Utilities volume of IEEE Std 1003.1-2001) or the largest possible file (important  
37616 for an application using the extension where a file is searched in chunks).

37617 The standard developers rejected the inclusion of a *regsub()* function that would be used to do  
37618 substitutions for a matched RE. While such a routine would be useful to some applications, its  
37619 utility would be much more limited than the matching function described here. Both RE parsing  
37620 and substitution are possible to implement without support other than that required by the  
37621 ISO C standard, but matching is much more complex than substituting. The only difficult part of  
37622 substitution, given the information supplied by *regexec()*, is finding the next character in a string  
37623 when there can be multi-byte characters. That is a much larger issue, and one that needs a more  
37624 general solution.

37625 The *errno* variable has not been used for error returns to avoid filling the *errno* name space for  
37626 this feature.

37627 The interface is defined so that the matched substrings *rm\_sp* and *rm\_ep* are in a separate  
37628 **regmatch\_t** structure instead of in **regex\_t**. This allows a single compiled RE to be used  
37629 simultaneously in several contexts; in *main()* and a signal handler, perhaps, or in multiple  
37630 threads of lightweight processes. (The *preg* argument to *regexec()* is declared with type **const**, so  
37631 the implementation is not permitted to use the structure to store intermediate results.) It also  
37632 allows an application to request an arbitrary number of substrings from an RE. The number of  
37633 subexpressions in the RE is reported in *re\_nsub* in *preg*. With this change to *regexec()*,  
37634 consideration was given to dropping the REG\_NOSUB flag since the user can now specify this  
37635 with a zero *nmatch* argument to *regexec()*. However, keeping REG\_NOSUB allows an  
37636 implementation to use a different (perhaps more efficient) algorithm if it knows in *regcomp()*  
37637 that no subexpressions need be reported. The implementation is only required to fill in *pmatch* if  
37638 *nmatch* is not zero and if REG\_NOSUB is not specified. Note that the **size\_t** type, as defined in  
37639 the ISO C standard, is unsigned, so the description of *regexec()* does not need to address  
37640 negative values of *nmatch*.

37641 REG\_NOTBOL was added to allow an application to do repeated searches for the same pattern  
37642 in a line. If the pattern contains a circumflex character that should match the beginning of a line,  
37643 then the pattern should only match when matched against the beginning of the line. Without  
37644 the REG\_NOTBOL flag, the application could rewrite the expression for subsequent matches,  
37645 but in the general case this would require parsing the expression. The need for REG\_NOTEOL is  
37646 not as clear; it was added for symmetry.

37647 The addition of the *regerror()* function addresses the historical need for conforming application  
37648 programs to have access to error information more than “Function failed to compile/match your  
37649 RE for unknown reasons”.

37650 This interface provides for two different methods of dealing with error conditions. The specific  
37651 error codes (REG\_EBRACE, for example), defined in **<regex.h>**, allow an application to recover  
37652 from an error if it is so able. Many applications, especially those that use patterns supplied by a  
37653 user, will not try to deal with specific error cases, but will just use *regerror()* to obtain a human-  
37654 readable error message to present to the user.

37655 The *regerror()* function uses a scheme similar to *confstr()* to deal with the problem of allocating  
37656 memory to hold the generated string. The scheme used by *strerror()* in the ISO C standard was  
37657 considered unacceptable since it creates difficulties for multi-threaded applications.

37658 The *preg* argument is provided to *regerror()* to allow an implementation to generate a more  
37659 descriptive message than would be possible with *errcode* alone. An implementation might, for  
37660 example, save the character offset of the offending character of the pattern in a field of *preg*, and  
37661 then include that in the generated message string. The implementation may also ignore *preg*.

37662 A REG\_FILENAME flag was considered, but omitted. This flag caused *regexec()* to match  
37663 patterns as described in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.13,  
37664 Pattern Matching Notation instead of REs. This service is now provided by the *fnmatch()*

37665 function.

37666 Notice that there is a difference in philosophy between the ISO POSIX-2:1993 standard and  
 37667 IEEE Std 1003.1-2001 in how to handle a “bad” regular expression. The ISO POSIX-2:1993  
 37668 standard says that many bad constructs “produce undefined results”, or that “the interpretation  
 37669 is undefined”. IEEE Std 1003.1-2001, however, says that the interpretation of such REs is  
 37670 unspecified. The term “undefined” means that the action by the application is an error, of  
 37671 similar severity to passing a bad pointer to a function.

37672 The *regcomp()* and *regexexec()* functions are required to accept any null-terminated string as the  
 37673 *pattern* argument. If the meaning of the string is “undefined”, the behavior of the function is  
 37674 “unspecified”. IEEE Std 1003.1-2001 does not specify how the functions will interpret the  
 37675 pattern; they might return error codes, or they might do pattern matching in some completely  
 37676 unexpected way, but they should not do something like abort the process.

37677 **FUTURE DIRECTIONS**

37678 None.

37679 **SEE ALSO**

37680 *fnmatch()*, *glob()*, Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.13, Pattern  
 37681 Matching Notation, Base Definitions volume of IEEE Std 1003.1-2001, Chapter 9, Regular  
 37682 Expressions, `<regex.h>`, `<sys/types.h>`

37683 **CHANGE HISTORY**

37684 First released in Issue 4. Derived from the ISO POSIX-2 standard.

37685 **Issue 5**

37686 Moved from POSIX2 C-language Binding to BASE.

37687 **Issue 6**

37688 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

37689 The following new requirements on POSIX implementations derive from alignment with the  
 37690 Single UNIX Specification:

- 37691 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
 37692 required for conforming implementations of previous POSIX specifications, it was not  
 37693 required for UNIX applications.

37694 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

37695 The REG\_ENOSYS constant is removed.

37696 The **restrict** keyword is added to the *regcomp()*, *regerror()*, and *regexexec()* prototypes for  
 37697 alignment with the ISO/IEC 9899:1999 standard.

37698 **NAME**

37699 remainder, remainderf, remainderl — remainder function

37700 **SYNOPSIS**

37701 #include &lt;math.h&gt;

37702 double remainder(double x, double y);

37703 float remainderf(float x, float y);

37704 long double remainderl(long double x, long double y);

37705 **DESCRIPTION**

37706 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 37707 conflict between the requirements described here and the ISO C standard is unintentional. This  
 37708 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

37709 These functions shall return the floating-point remainder  $r=x-ny$  when  $y$  is non-zero. The value  
 37710  $n$  is the integral value nearest the exact value  $x/y$ . When  $|n-x/y| = 1/2$ , the value  $n$  is chosen to  
 37711 be even.

37712 The behavior of *remainder()* shall be independent of the rounding mode.37713 **RETURN VALUE**37714 Upon successful completion, these functions shall return the floating-point remainder  $r=x-ny$   
37715 when  $y$  is non-zero.37716 **MX** If  $x$  or  $y$  is NaN, a NaN shall be returned.37717 If  $x$  is infinite or  $y$  is 0 and the other is non-NaN, a domain error shall occur, and either a NaN (if  
37718 supported), or an implementation-defined value shall be returned.37719 **ERRORS**

37720 These functions shall fail if:

|                 |                     |                                                                                                         |
|-----------------|---------------------|---------------------------------------------------------------------------------------------------------|
| 37721 <b>MX</b> | <b>Domain Error</b> | The $x$ argument is $\pm\text{Inf}$ , or the $y$ argument is $\pm 0$ and the other argument is non-NaN. |
|-----------------|---------------------|---------------------------------------------------------------------------------------------------------|

|       |       |       |       |                                                                                                                                                                                                                                                        |
|-------|-------|-------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 37723 | 37724 | 37725 | 37726 | If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then <i>errno</i> shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised. |
|-------|-------|-------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

37727 **EXAMPLES**

37728 None.

37729 **APPLICATION USAGE**37730 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
37731 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.37732 **RATIONALE**

37733 None.

37734 **FUTURE DIRECTIONS**

37735 None.

37736 **SEE ALSO**37737 *abs()*, *div()*, *feclearexcept()*, *fetestexcept()*, *ldiv()*, the Base Definitions volume of  
37738 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,  
37739 <math.h>

37740 **CHANGE HISTORY**

37741 First released in Issue 4, Version 2.

37742 **Issue 5**

37743 Moved from X/OPEN UNIX extension to BASE.

37744 **Issue 6**

37745 The *remainder()* function is no longer marked as an extension.

37746 The *remainderf()* and *remainderl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

37748 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

37750 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

37751

37752 **NAME**

37753 remove — remove a file

37754 **SYNOPSIS**

37755 #include &lt;stdio.h&gt;

37756 int remove(const char \*path);

37757 **DESCRIPTION**

37758 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
37759 conflict between the requirements described here and the ISO C standard is unintentional. This  
37760 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

37761 The *remove()* function shall cause the file named by the pathname pointed to by *path* to be no  
37762 longer accessible by that name. A subsequent attempt to open that file using that name shall fail,  
37763 unless it is created anew.

37764 CX If *path* does not name a directory, *remove(path)* shall be equivalent to *unlink(path)*.

37765 If *path* names a directory, *remove(path)* shall be equivalent to *rmdir(path)*.

37766 **RETURN VALUE**37767 CX Refer to *rmdir()* or *unlink()*.37768 **ERRORS**37769 CX Refer to *rmdir()* or *unlink()*.37770 **EXAMPLES**37771 **Removing Access to a File**37772 The following example shows how to remove access to a file named `/home/cnd/old_mods`.

37773 #include &lt;stdio.h&gt;

37774 int status;

37775 ...

37776 status = remove("/home/cnd/old\_mods");

37777 **APPLICATION USAGE**

37778 None.

37779 **RATIONALE**

37780 None.

37781 **FUTURE DIRECTIONS**

37782 None.

37783 **SEE ALSO**37784 *rmdir()*, *unlink()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>37785 **CHANGE HISTORY**

37786 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ISO C  
37787 standard.

37788 **Issue 6**

37789 Extensions beyond the ISO C standard are marked.

37790 The following new requirements on POSIX implementations derive from alignment with the  
37791 Single UNIX Specification:

37792  
37793  
37794

- The DESCRIPTION, RETURN VALUE, and ERRORS sections are updated so that if *path* is not a directory, *remove()* is equivalent to *unlink()*, and if it is a directory, it is equivalent to *rmdir()*.

37795 **NAME**

37796           remque — remove an element from a queue

37797 **SYNOPSIS**

37798 xSI       #include <search.h>

37799           void remque(void \*element);

37800

37801 **DESCRIPTION**

37802           Refer to *insque()*.

37803 **NAME**

37804 remquo, remquof, remquol — remainder functions

37805 **SYNOPSIS**

37806 #include &lt;math.h&gt;

37807 double remquo(double x, double y, int \*quo);

37808 float remquof(float x, float y, int \*quo);

37809 long double remquol(long double x, long double y, int \*quo);

37810 **DESCRIPTION**

37811 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 37812 conflict between the requirements described here and the ISO C standard is unintentional. This  
 37813 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

37814 The *remquo()*, *remquof()*, and *remquol()* functions shall compute the same remainder as the  
 37815 *remainder()*, *remainderf()*, and *remainderl()* functions, respectively. In the object pointed to by  
 37816 *quo*, they store a value whose sign is the sign of  $x/y$  and whose magnitude is congruent modulo  
 37817  $2^n$  to the magnitude of the integral quotient of  $x/y$ , where  $n$  is an implementation-defined  
 37818 integer greater than or equal to 3.

37819 An application wishing to check for error situations should set *errno* to zero and call  
 37820 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 37821 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 37822 zero, an error has occurred.

37823 **RETURN VALUE**37824 These functions shall return  $x \text{ REM } y$ .37825 **MX** If  $x$  or  $y$  is NaN, a NaN shall be returned.

37826 If  $x$  is  $\pm\text{Inf}$  or  $y$  is zero and the other argument is non-NaN, a domain error shall occur, and either  
 37827 a NaN (if supported), or an implementation-defined value shall be returned.

37828 **ERRORS**

37829 These functions shall fail if:

37830 **MX** **Domain Error** The  $x$  argument is  $\pm\text{Inf}$ , or the  $y$  argument is  $\pm 0$  and the other argument is  
 37831 non-NaN.

37832 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 37833 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 37834 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 37835 shall be raised.

37836 **EXAMPLES**

37837 None.

37838 **APPLICATION USAGE**

37839 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 37840 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

37841 **RATIONALE**

37842 These functions are intended for implementing argument reductions which can exploit a few  
 37843 low-order bits of the quotient. Note that  $x$  may be so large in magnitude relative to  $y$  that an  
 37844 exact representation of the quotient is not practical.

37845 **FUTURE DIRECTIONS**

37846 None.

37847 **SEE ALSO**37848 *feclearexcept()*, *fetetestexcept()*, *remainder()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
37849 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>37850 **CHANGE HISTORY**

37851 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

## 37852 NAME

37853 rename — rename a file

## 37854 SYNOPSIS

37855 #include &lt;stdio.h&gt;

37856 int rename(const char \*old, const char \*new);

## 37857 DESCRIPTION

37858 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 37859 conflict between the requirements described here and the ISO C standard is unintentional. This  
 37860 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

37861 The *rename()* function shall change the name of a file. The *old* argument points to the pathname  
 37862 of the file to be renamed. The *new* argument points to the new pathname of the file.

37863 cx If either the *old* or *new* argument names a symbolic link, *rename()* shall operate on the symbolic  
 37864 link itself, and shall not resolve the last component of the argument. If the *old* argument and the  
 37865 *new* argument resolve to the same existing file, *rename()* shall return successfully and perform no  
 37866 other action.

37867 If the *old* argument points to the pathname of a file that is not a directory, the *new* argument shall  
 37868 not point to the pathname of a directory. If the link named by the *new* argument exists, it shall be  
 37869 removed and *old* renamed to *new*. In this case, a link named *new* shall remain visible to other  
 37870 processes throughout the renaming operation and refer either to the file referred to by *new* or *old*  
 37871 before the operation began. Write access permission is required for both the directory containing  
 37872 *old* and the directory containing *new*.

37873 If the *old* argument points to the pathname of a directory, the *new* argument shall not point to the  
 37874 pathname of a file that is not a directory. If the directory named by the *new* argument exists, it  
 37875 shall be removed and *old* renamed to *new*. In this case, a link named *new* shall exist throughout  
 37876 the renaming operation and shall refer either to the directory referred to by *new* or *old* before the  
 37877 operation began. If *new* names an existing directory, it shall be required to be an empty directory.

37878 If the *old* argument points to a pathname of a symbolic link, the symbolic link shall be renamed.  
 37879 If the *new* argument points to a pathname of a symbolic link, the symbolic link shall be removed.

37880 The *new* pathname shall not contain a path prefix that names *old*. Write access permission is  
 37881 required for the directory containing *old* and the directory containing *new*. If the *old* argument  
 37882 points to the pathname of a directory, write access permission may be required for the directory  
 37883 named by *old*, and, if it exists, the directory named by *new*.

37884 If the link named by the *new* argument exists and the file's link count becomes 0 when it is  
 37885 removed and no process has the file open, the space occupied by the file shall be freed and the  
 37886 file shall no longer be accessible. If one or more processes have the file open when the last link is  
 37887 removed, the link shall be removed before *rename()* returns, but the removal of the file contents  
 37888 shall be postponed until all references to the file are closed.

37889 Upon successful completion, *rename()* shall mark for update the *st\_ctime* and *st\_mtime* fields of  
 37890 the parent directory of each file.

37891 If the *rename()* function fails for any reason other than [EIO], any file named by *new* shall be  
 37892 unaffected.

## 37893 RETURN VALUE

37894 cx Upon successful completion, *rename()* shall return 0; otherwise,  $-1$  shall be returned, *errno* shall  
 37895 be set to indicate the error, and neither the file named by *old* nor the file named by *new* shall be  
 37896 changed or created.

37897 **ERRORS**37898 The *rename()* function shall fail if:

37899 CX [EACCES] A component of either path prefix denies search permission; or one of the  
37900 directories containing *old* or *new* denies write permissions; or, write  
37901 permission is required and is denied for a directory pointed to by the *old* or  
37902 *new* arguments.

37903 CX [EBUSY] The directory named by *old* or *new* is currently in use by the system or another  
37904 process, and the implementation considers this an error.

37905 CX [EEXIST] or [ENOTEMPTY]  
37906 The link named by *new* is a directory that is not an empty directory.

37907 CX [EINVAL] The *new* directory pathname contains a path prefix that names the *old*  
37908 directory.

37909 CX [EIO] A physical I/O error has occurred.

37910 CX [EISDIR] The *new* argument points to a directory and the *old* argument points to a file  
37911 that is not a directory.

37912 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
37913 argument.

37914 CX [EMLINK] The file named by *old* is a directory, and the link count of the parent directory  
37915 of *new* would exceed {LINK\_MAX}.

37916 CX [ENAMETOOLONG]  
37917 The length of the *old* or *new* argument exceeds {PATH\_MAX} or a pathname  
37918 component is longer than {NAME\_MAX}.

37919 CX [ENOENT] The link named by *old* does not name an existing file, or either *old* or *new*  
37920 points to an empty string.

37921 CX [ENOSPC] The directory that would contain *new* cannot be extended.

37922 CX [ENOTDIR] A component of either path prefix is not a directory; or the *old* argument  
37923 names a directory and *new* argument names a non-directory file.

37924 XSI [EPERM] or [EACCES]  
37925 The S\_ISVTX flag is set on the directory containing the file referred to by *old*  
37926 and the caller is not the file owner, nor is the caller the directory owner, nor  
37927 does the caller have appropriate privileges; or *new* refers to an existing file, the  
37928 S\_ISVTX flag is set on the directory containing this file, and the caller is not  
37929 the file owner, nor is the caller the directory owner, nor does the caller have  
37930 appropriate privileges.

37931 CX [EROFS] The requested operation requires writing in a directory on a read-only file  
37932 system.

37933 CX [EXDEV] The links named by *new* and *old* are on different file systems and the  
37934 implementation does not support links between file systems.

37935 The *rename()* function may fail if:

37936 XSI [EBUSY] The file named by the *old* or *new* arguments is a named STREAM.

37937 CX [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
37938 resolution of the *path* argument.

37939 CX [ENAMETOOLONG]  
 37940 As a result of encountering a symbolic link in resolution of the *path* argument,  
 37941 the length of the substituted pathname string exceeded {PATH\_MAX}.

37942 CX [ETXTBSY] The file to be renamed is a pure procedure (shared text) file that is being  
 37943 executed.

37944 **EXAMPLES**37945 **Renaming a File**

37946 The following example shows how to rename a file named `/home/cnd/mod1` to  
 37947 `/home/cnd/mod2`.

```
37948 #include <stdio.h>
37949 int status;
37950 ...
37951 status = rename("/home/cnd/mod1", "/home/cnd/mod2");
```

37952 **APPLICATION USAGE**

37953 Some implementations mark for update the *st\_ctime* field of renamed files and some do not.  
 37954 Applications which make use of the *st\_ctime* field may behave differently with respect to  
 37955 renamed files unless they are designed to allow for either behavior.

37956 **RATIONALE**

37957 This *rename()* function is equivalent for regular files to that defined by the ISO C standard. Its  
 37958 inclusion here expands that definition to include actions on directories and specifies behavior  
 37959 when the *new* parameter names a file that already exists. That specification requires that the  
 37960 action of the function be atomic.

37961 One of the reasons for introducing this function was to have a means of renaming directories  
 37962 while permitting implementations to prohibit the use of *link()* and *unlink()* with directories,  
 37963 thus constraining links to directories to those made by *mkdir()*.

37964 The specification that if *old* and *new* refer to the same file is intended to guarantee that:

```
37965 rename("x", "x");
```

37966 does not remove the file.

37967 Renaming dot or dot-dot is prohibited in order to prevent cyclical file system paths.

37968 See also the descriptions of [ENOTEMPTY] and [ENAMETOOLONG] in *rmdir()* and [EBUSY] in  
 37969 *unlink()*. For a discussion of [EXDEV], see *link()*.

37970 **FUTURE DIRECTIONS**

37971 None.

37972 **SEE ALSO**

37973 *link()*, *rmdir()*, *symlink()*, *unlink()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
 37974 `<stdio.h>`

37975 **CHANGE HISTORY**

37976 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

37977 **Issue 5**

37978 The [EBUSY] error is added to the “may fail” part of the ERRORS section.

37979 **Issue 6**

37980 Extensions beyond the ISO C standard are marked.

37981 The following new requirements on POSIX implementations derive from alignment with the  
37982 Single UNIX Specification:

37983 • The [EIO] mandatory error condition is added.

37984 • The [ELOOP] mandatory error condition is added.

37985 • A second [ENAMETOOLONG] is added as an optional error condition.

37986 • The [ETXTBSY] optional error condition is added.

37987 The following changes were made to align with the IEEE P1003.1a draft standard:

37988 • Details are added regarding the treatment of symbolic links.

37989 • The [ELOOP] optional error condition is added.

37990 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

37991 **NAME**

37992           rewind — reset the file position indicator in a stream

37993 **SYNOPSIS**

37994           #include <stdio.h>

37995           void rewind(FILE \*stream);

37996 **DESCRIPTION**

37997 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
37998           conflict between the requirements described here and the ISO C standard is unintentional. This  
37999           volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

38000           The call:

38001           rewind(stream)

38002           shall be equivalent to:

38003           (void) fseek(stream, 0L, SEEK\_SET)

38004           except that *rewind()* shall also clear the error indicator.

38005 **CX**       Since *rewind()* does not return a value, an application wishing to detect errors should clear *errno*,  
38006           then call *rewind()*, and if *errno* is non-zero, assume an error has occurred.

38007 **RETURN VALUE**

38008           The *rewind()* function shall not return a value.

38009 **ERRORS**

38010 **CX**       Refer to *fseek()* with the exception of [EINVAL] which does not apply.

38011 **EXAMPLES**

38012           None.

38013 **APPLICATION USAGE**

38014           None.

38015 **RATIONALE**

38016           None.

38017 **FUTURE DIRECTIONS**

38018           None.

38019 **SEE ALSO**

38020           *fseek()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

38021 **CHANGE HISTORY**

38022           First released in Issue 1. Derived from Issue 1 of the SVID.

38023 **Issue 6**

38024           Extensions beyond the ISO C standard are marked.

38025 **NAME**

38026        rewinddir — reset the position of a directory stream to the beginning of a directory

38027 **SYNOPSIS**

38028        #include <dirent.h>

38029        void rewinddir(DIR \*dirp);

38030 **DESCRIPTION**

38031        The *rewinddir()* function shall reset the position of the directory stream to which *dirp* refers to the beginning of the directory. It shall also cause the directory stream to refer to the current state of the corresponding directory, as a call to *opendir()* would have done. If *dirp* does not refer to a directory stream, the effect is undefined.

38035        After a call to the *fork()* function, either the parent or child (but not both) may continue processing the directory stream using *readdir()*, *rewinddir()*, or *seekdir()*. If both the parent and child processes use these functions, the result is undefined.

38038 **RETURN VALUE**

38039        The *rewinddir()* function shall not return a value.

38040 **ERRORS**

38041        No errors are defined.

38042 **EXAMPLES**

38043        None.

38044 **APPLICATION USAGE**

38045        The *rewinddir()* function should be used in conjunction with *opendir()*, *readdir()*, and *closedir()* to examine the contents of the directory. This method is recommended for portability.

38047 **RATIONALE**

38048        None.

38049 **FUTURE DIRECTIONS**

38050        None.

38051 **SEE ALSO**

38052        *closedir()*, *opendir()*, *readdir()*, the Base Definitions volume of IEEE Std 1003.1-2001, <dirent.h>  
38053        <sys/types.h>

38054 **CHANGE HISTORY**

38055        First released in Issue 2.

38056 **Issue 6**

38057        In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

38058        The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 38060        • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

38063 **NAME**38064 rindex — character string operations (**LEGACY**)38065 **SYNOPSIS**38066 XSI 

```
#include <strings.h>
```

38067 

```
char *rindex(const char *s, int c);
```

38068

38069 **DESCRIPTION**38070 The *rindex()* function shall be equivalent to *strchr()*.38071 **RETURN VALUE**38072 Refer to *strchr()*.38073 **ERRORS**38074 Refer to *strchr()*.38075 **EXAMPLES**

38076 None.

38077 **APPLICATION USAGE**38078 The *strchr()* function is preferred over this function.38079 For maximum portability, it is recommended to replace the function call to *rindex()* as follows:38080 

```
#define rindex(a,b) strchr((a),(b))
```

38081 **RATIONALE**

38082 None.

38083 **FUTURE DIRECTIONS**

38084 This function may be withdrawn in a future version.

38085 **SEE ALSO**38086 *strchr()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<strings.h>**38087 **CHANGE HISTORY**

38088 First released in Issue 4, Version 2.

38089 **Issue 5**

38090 Moved from X/OPEN UNIX extension to BASE.

38091 **Issue 6**

38092 This function is marked LEGACY.

## 38093 NAME

38094 rint, rintf, rintl — round-to-nearest integral value

## 38095 SYNOPSIS

38096 #include &lt;math.h&gt;

38097 double rint(double x);

38098 float rintf(float x);

38099 long double rintl(long double x);

## 38100 DESCRIPTION

38101 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 38102 conflict between the requirements described here and the ISO C standard is unintentional. This  
 38103 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

38104 These functions shall return the integral value (represented as a **double**) nearest  $x$  in the  
 38105 direction of the current rounding mode. The current rounding mode is implementation-defined.

38106 If the current rounding mode rounds toward negative infinity, then *rint()* shall be equivalent to  
 38107 *floor()*. If the current rounding mode rounds toward positive infinity, then *rint()* shall be  
 38108 equivalent to *ceil()*.

38109 These functions differ from the *nearbyint()*, *nearbyintf()*, and *nearbyintl()* functions only in that  
 38110 they may raise the inexact floating-point exception if the result differs in value from the  
 38111 argument.

38112 An application wishing to check for error situations should set *errno* to zero and call  
 38113 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 38114 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 38115 zero, an error has occurred.

## 38116 RETURN VALUE

38117 Upon successful completion, these functions shall return the integer (represented as a double  
 38118 precision number) nearest  $x$  in the direction of the current rounding mode.

38119 MX If  $x$  is NaN, a NaN shall be returned.

38120 If  $x$  is  $\pm 0$  or  $\pm \text{Inf}$ ,  $x$  shall be returned.

38121 XSI If the correct value would cause overflow, a range error shall occur and *rint()*, *rintf()*, and *rintl()*  
 38122 shall return the value of the macro  $\pm \text{HUGE\_VAL}$ ,  $\pm \text{HUGE\_VALF}$ , and  $\pm \text{HUGE\_VALL}$  (with the  
 38123 same sign as  $x$ ), respectively.

## 38124 ERRORS

38125 These functions shall fail if:

38126 XSI **Range Error** The result would cause an overflow.

38127 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 38128 then *errno* shall be set to [ERANGE]. If the integer expression  
 38129 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 38130 floating-point exception shall be raised.

38131 **EXAMPLES**

38132 None.

38133 **APPLICATION USAGE**

38134 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`  
38135 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

38136 **RATIONALE**

38137 None.

38138 **FUTURE DIRECTIONS**

38139 None.

38140 **SEE ALSO**

38141 *abs()*, *ceil()*, *feclearexcept()*, *fetestexcept()*, *floor()*, *isnan()*, *nearbyint()*, the Base Definitions volume  
38142 of IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,  
38143 <math.h>

38144 **CHANGE HISTORY**

38145 First released in Issue 4, Version 2.

38146 **Issue 5**

38147 Moved from X/OPEN UNIX extension to BASE.

38148 **Issue 6**

38149 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 38150 • The *rintf()* and *rintl()* functions are added.
- 38151 • The *rint()* function is no longer marked as an extension.
- 38152 • The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
38153 revised to align with the ISO/IEC 9899:1999 standard.
- 38154 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
38155 marked.

38156 **NAME**

38157 rmdir — remove a directory

38158 **SYNOPSIS**

38159 #include &lt;unistd.h&gt;

38160 int rmdir(const char \*path);

38161 **DESCRIPTION**38162 The *rmdir()* function shall remove a directory whose name is given by *path*. The directory shall  
38163 be removed only if it is an empty directory.38164 If the directory is the root directory or the current working directory of any process, it is  
38165 unspecified whether the function succeeds, or whether it shall fail and set *errno* to [EBUSY].38166 If *path* names a symbolic link, then *rmdir()* shall fail and set *errno* to [ENOTDIR].38167 If the *path* argument refers to a path whose final component is either dot or dot-dot, *rmdir()* shall  
38168 fail.38169 If the directory's link count becomes 0 and no process has the directory open, the space occupied  
38170 by the directory shall be freed and the directory shall no longer be accessible. If one or more  
38171 processes have the directory open when the last link is removed, the dot and dot-dot entries, if  
38172 present, shall be removed before *rmdir()* returns and no new entries may be created in the  
38173 directory, but the directory shall not be removed until all references to the directory are closed.38174 If the directory is not an empty directory, *rmdir()* shall fail and set *errno* to [EEXIST] or  
38175 [ENOTEMPTY].38176 Upon successful completion, the *rmdir()* function shall mark for update the *st\_ctime* and  
38177 *st\_mtime* fields of the parent directory.38178 **RETURN VALUE**38179 Upon successful completion, the function *rmdir()* shall return 0. Otherwise, -1 shall be returned,  
38180 and *errno* set to indicate the error. If -1 is returned, the named directory shall not be changed.38181 **ERRORS**38182 The *rmdir()* function shall fail if:38183 [EACCES] Search permission is denied on a component of the path prefix, or write  
38184 permission is denied on the parent directory of the directory to be removed.38185 [EBUSY] The directory to be removed is currently in use by the system or some process  
38186 and the implementation considers this to be an error.38187 [EEXIST] or [ENOTEMPTY]  
38188 The *path* argument names a directory that is not an empty directory, or there  
38189 are hard links to the directory other than dot or a single entry in dot-dot.38190 [EINVAL] The *path* argument contains a last component that is dot.

38191 [EIO] A physical I/O error has occurred.

38192 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
38193 argument.38194 [ENAMETOOLONG]  
38195 The length of the *path* argument exceeds {PATH\_MAX} or a pathname  
38196 component is longer than {NAME\_MAX}.38197 [ENOENT] A component of *path* does not name an existing file, or the *path* argument  
38198 names a nonexistent directory or points to an empty string.

- 38199 [ENOTDIR] A component of *path* is not a directory.
- 38200 XSI [EPERM] or [EACCES]  
 38201 The S\_ISVTX flag is set on the parent directory of the directory to be removed  
 38202 and the caller is not the owner of the directory to be removed, nor is the caller  
 38203 the owner of the parent directory, nor does the caller have the appropriate  
 38204 privileges.
- 38205 [EROFS] The directory entry to be removed resides on a read-only file system.
- 38206 The *rmdir()* function may fail if:
- 38207 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 38208 resolution of the *path* argument.
- 38209 [ENAMETOOLONG]  
 38210 As a result of encountering a symbolic link in resolution of the *path* argument,  
 38211 the length of the substituted pathname string exceeded {PATH\_MAX}.

38212 **EXAMPLES**38213 **Removing a Directory**

38214 The following example shows how to remove a directory named `/home/cnd/mod1`.

```
38215 #include <unistd.h>
38216 int status;
38217 ...
38218 status = rmdir("/home/cnd/mod1");
```

38219 **APPLICATION USAGE**

38220 None.

38221 **RATIONALE**

38222 The *rmdir()* and *rename()* functions originated in 4.2 BSD, and they used [ENOTEMPTY] for the  
 38223 condition when the directory to be removed does not exist or *new* already exists. When the 1984  
 38224 /usr/group standard was published, it contained [EEXIST] instead. When these functions were  
 38225 adopted into System V, the 1984 /usr/group standard was used as a reference. Therefore, several  
 38226 existing applications and implementations support/use both forms, and no agreement could be  
 38227 reached on either value. All implementations are required to supply both [EEXIST] and  
 38228 [ENOTEMPTY] in `<errno.h>` with distinct values, so that applications can use both values in C-  
 38229 language **case** statements.

38230 The meaning of deleting *pathname/dot* is unclear, because the name of the file (directory) in the  
 38231 parent directory to be removed is not clear, particularly in the presence of multiple links to a  
 38232 directory.

38233 The POSIX.1-1990 standard was silent with regard to the behavior of *rmdir()* when there are  
 38234 multiple hard links to the directory being removed. The requirement to set *errno* to [EEXIST] or  
 38235 [ENOTEMPTY] clarifies the behavior in this case.

38236 If the process' current working directory is being removed, that should be an allowed error.

38237 Virtually all existing implementations detect [ENOTEMPTY] or the case of dot-dot. The text in  
 38238 Section 2.3 (on page 21) about returning any one of the possible errors permits that behavior to  
 38239 continue. The [ELOOP] error may be returned if more than {SYMLOOP\_MAX} symbolic links  
 38240 are encountered during resolution of the *path* argument.

38241 **FUTURE DIRECTIONS**

38242 None.

38243 **SEE ALSO**38244 Section 2.3 (on page 21), *mkdir()*, *remove()*, *unlink()*, the Base Definitions volume of  
38245 IEEE Std 1003.1-2001, <**unistd.h**>38246 **CHANGE HISTORY**

38247 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

38248 **Issue 6**38249 The following new requirements on POSIX implementations derive from alignment with the  
38250 Single UNIX Specification:

- 38251 • The DESCRIPTION is updated to indicate the results of naming a symbolic link in *path*.
- 38252 • The [EIO] mandatory error condition is added.
- 38253 • The [ELOOP] mandatory error condition is added.
- 38254 • A second [ENAMETOOLONG] is added as an optional error condition.

38255 The following changes were made to align with the IEEE P1003.1a draft standard:

- 38256 • The [ELOOP] optional error condition is added.

38257 **NAME**

38258 round, roundf, roundl — round to the nearest integer value in a floating-point format

38259 **SYNOPSIS**

38260 #include &lt;math.h&gt;

38261 double round(double x);

38262 float roundf(float x);

38263 long double roundl(long double x);

38264 **DESCRIPTION**

38265 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 38266 conflict between the requirements described here and the ISO C standard is unintentional. This  
 38267 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

38268 These functions shall round their argument to the nearest integer value in floating-point format,  
 38269 rounding halfway cases away from zero, regardless of the current rounding direction.

38270 An application wishing to check for error situations should set *errno* to zero and call  
 38271 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 38272 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 38273 zero, an error has occurred.

38274 **RETURN VALUE**

38275 Upon successful completion, these functions shall return the rounded integer value.

38276 **MX** If *x* is NaN, a NaN shall be returned.38277 If *x* is  $\pm 0$  or  $\pm \text{Inf}$ , *x* shall be returned.

38278 **XSI** If the correct value would cause overflow, a range error shall occur and *round()*, *roundf()*, and  
 38279 *roundl()* shall return the value of the macro  $\pm \text{HUGE\_VAL}$ ,  $\pm \text{HUGE\_VALF}$ , and  $\pm \text{HUGE\_VALL}$   
 38280 (with the same sign as *x*), respectively.

38281 **ERRORS**

38282 These functions may fail if:

38283 **XSI** **Range Error** The result overflows.

38284 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 38285 then *errno* shall be set to [ERANGE]. If the integer expression  
 38286 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow  
 38287 floating-point exception shall be raised.

38288 **EXAMPLES**

38289 None.

38290 **APPLICATION USAGE**

38291 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 38292 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

38293 **RATIONALE**

38294 None.

38295 **FUTURE DIRECTIONS**

38296 None.

38297 **SEE ALSO**

38298 *feclearexcept()*, *fetetestexcept()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.18,  
38299 Treatment of Error Conditions for Mathematical Functions, <math.h>

38300 **CHANGE HISTORY**

38301 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

## 38302 NAME

38303 scalb — load exponent of a radix-independent floating-point number

## 38304 SYNOPSIS

38305 OB XSI `#include <math.h>`38306 `double scalb(double x, double n);`

38307

## 38308 DESCRIPTION

38309 The *scalb()* function shall compute  $x \cdot r^n$ , where  $r$  is the radix of the machine's floating-point  
 38310 arithmetic. When  $r$  is 2, *scalb()* shall be equivalent to *ldexp()*. The value of  $r$  is FLT\_RADIX  
 38311 which is defined in `<float.h>`.

38312 An application wishing to check for error situations should set *errno* to zero and call  
 38313 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 38314 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 38315 zero, an error has occurred.

## 38316 RETURN VALUE

38317 Upon successful completion, the *scalb()* function shall return  $x \cdot r^n$ .38318 If  $x$  or  $n$  is NaN, a NaN shall be returned.38319 If  $n$  is zero,  $x$  shall be returned.38320 If  $x$  is  $\pm\text{Inf}$  and  $n$  is not  $-\text{Inf}$ ,  $x$  shall be returned.38321 If  $x$  is  $\pm 0$  and  $n$  is not  $+\text{Inf}$ ,  $x$  shall be returned.

38322 If  $x$  is  $\pm 0$  and  $n$  is  $+\text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an  
 38323 implementation-defined value shall be returned.

38324 If  $x$  is  $\pm\text{Inf}$  and  $n$  is  $-\text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an  
 38325 implementation-defined value shall be returned.

38326 If the result would cause an overflow, a range error shall occur and  $\pm\text{HUGE\_VAL}$  (according to  
 38327 the sign of  $x$ ) shall be returned.

38328 If the correct value would cause underflow, and is representable, a range error may occur and  
 38329 the correct value shall be returned.

38330 If the correct value would cause underflow, and is not representable, a range error may occur,  
 38331 and 0.0 shall be returned.

## 38332 ERRORS

38333 The *scalb()* function shall fail if:38334 Domain Error If  $x$  is zero and  $n$  is  $+\text{Inf}$ , or  $x$  is  $\text{Inf}$  and  $n$  is  $-\text{Inf}$ .

38335 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 38336 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 38337 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 38338 shall be raised.

38339 Range Error The result would overflow.

38340 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 38341 then *errno* shall be set to [ERANGE]. If the integer expression  
 38342 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 38343 floating-point exception shall be raised.

38344 The *scalb()* function may fail if:

38345 Range Error The result underflows.

38346 If the integer expression (*math\_errhandling* & *MATH\_ERRNO*) is non-zero,  
 38347 then *errno* shall be set to [ERANGE]. If the integer expression  
 38348 (*math\_errhandling* & *MATH\_ERREXCEPT*) is non-zero, then the underflow  
 38349 floating-point exception shall be raised.

#### 38350 EXAMPLES

38351 None.

#### 38352 APPLICATION USAGE

38353 Applications should use either *scalbln()*, *scalblnf()*, or *scalblnl()* in preference to this function.

38354 IEEE Std 1003.1-2001 only defines the behavior for the *scalb()* function when the *n* argument is  
 38355 an integer, a NaN, or Inf. The behavior of other values for the *n* argument is unspecified.

38356 On error, the expressions (*math\_errhandling* & *MATH\_ERRNO*) and (*math\_errhandling* &  
 38357 *MATH\_ERREXCEPT*) are independent of each other, but at least one of them must be non-zero.

#### 38358 RATIONALE

38359 None.

#### 38360 FUTURE DIRECTIONS

38361 None.

#### 38362 SEE ALSO

38363 *feclearexcept()*, *fetetestexcept()*, *ilogb()*, *ldexp()*, *logb()*, *scalbln()*, the Base Definitions volume of  
 38364 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,  
 38365 <float.h>, <math.h>

#### 38366 CHANGE HISTORY

38367 First released in Issue 4, Version 2.

#### 38368 Issue 5

38369 Moved from X/OPEN UNIX extension to BASE.

38370 The DESCRIPTION is updated to indicate how an application should check for an error. This  
 38371 text was previously published in the APPLICATION USAGE section.

#### 38372 Issue 6

38373 This function is marked obsolescent.

38374 Although this function is not part of the ISO/IEC 9899:1999 standard, the RETURN VALUE and  
 38375 ERRORS sections are updated to align with the error handling in the ISO/IEC 9899:1999  
 38376 standard.

## 38377 NAME

38378 scalbn, scalblnf, scalblnl, scalbn, scalbnf, scalbnl — compute exponent using FLT\_RADIX

## 38379 SYNOPSIS

38380 #include <math.h>

```
38381 double scalbn(double x, long n);
38382 float scalblnf(float x, long n);
38383 long double scalblnl(long double x, long n);
38384 double scalbn(double x, int n);
38385 float scalbnf(float x, int n);
38386 long double scalbnl(long double x, int n);
```

## 38387 DESCRIPTION

38388 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
38389 conflict between the requirements described here and the ISO C standard is unintentional. This  
38390 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

38391 These functions shall compute  $x * FLT\_RADIX^n$  efficiently, not normally by computing  
38392  $FLT\_RADIX^n$  explicitly.

38393 An application wishing to check for error situations should set *errno* to zero and call  
38394 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
38395 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
38396 zero, an error has occurred.

## 38397 RETURN VALUE

38398 Upon successful completion, these functions shall return  $x * FLT\_RADIX^n$ .

38399 If the result would cause overflow, a range error shall occur and these functions shall return  
38400  $\pm HUGUE\_VAL$ ,  $\pm HUGUE\_VALF$ , and  $\pm HUGUE\_VALL$  (according to the sign of *x*) as appropriate for  
38401 the return type of the function.

38402 If the correct value would cause underflow, and is not representable, a range error may occur,  
38403 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

38404 MX If *x* is NaN, a NaN shall be returned.

38405 If *x* is  $\pm 0$  or  $\pm Inf$ , *x* shall be returned.

38406 If *n* is 0, *x* shall be returned.

38407 If the correct value would cause underflow, and is representable, a range error may occur and  
38408 the correct value shall be returned.

## 38409 ERRORS

38410 These functions shall fail if:

38411 Range Error The result overflows.

38412 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
38413 then *errno* shall be set to [ERANGE]. If the integer expression  
38414 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow  
38415 floating-point exception shall be raised.

38416 These functions may fail if:

38417 Range Error The result underflows.

38418 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
38419 then *errno* shall be set to [ERANGE]. If the integer expression

38420 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow  
38421 floating-point exception shall be raised.

38422 **EXAMPLES**

38423 None.

38424 **APPLICATION USAGE**

38425 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
38426 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

38427 **RATIONALE**

38428 These functions are named so as to avoid conflicting with the historical definition of the *scalb()*  
38429 function from the Single UNIX Specification. The difference is that the *scalb()* function has a  
38430 second argument of **double** instead of **int**. The *scalb()* function is not part of the ISO C standard.  
38431 The three functions whose second type is **long** are provided because the factor required to scale  
38432 from the smallest positive floating-point value to the largest finite one, on many  
38433 implementations, is too large to represent in the minimum-width **int** format.

38434 **FUTURE DIRECTIONS**

38435 None.

38436 **SEE ALSO**

38437 *feclearexcept()*, *fetestexcept()*, *scalb()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section  
38438 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

38439 **CHANGE HISTORY**

38440 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

38441 **NAME**

38442           scanf — convert formatted input

38443 **SYNOPSIS**

38444           #include <stdio.h>

38445           int scanf(const char \*restrict *format*, ... );

38446 **DESCRIPTION**

38447           Refer to *fscanf()*.

38448 **NAME**38449 sched\_get\_priority\_max, sched\_get\_priority\_min — get priority limits (**REALTIME**)38450 **SYNOPSIS**

38451 PS|TPS #include &lt;sched.h&gt;

38452 int sched\_get\_priority\_max(int *policy*);38453 int sched\_get\_priority\_min(int *policy*);

38454

38455 **DESCRIPTION**38456 The *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()* functions shall return the appropriate  
38457 maximum or minimum, respectively, for the scheduling policy specified by *policy*.38458 The value of *policy* shall be one of the scheduling policy values defined in <sched.h>.38459 **RETURN VALUE**38460 If successful, the *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()* functions shall return the  
38461 appropriate maximum or minimum values, respectively. If unsuccessful, they shall return a  
38462 value of  $-1$  and set *errno* to indicate the error.38463 **ERRORS**38464 The *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()* functions shall fail if:38465 [EINVAL] The value of the *policy* parameter does not represent a defined scheduling  
38466 policy.38467 **EXAMPLES**

38468 None.

38469 **APPLICATION USAGE**

38470 None.

38471 **RATIONALE**

38472 None.

38473 **FUTURE DIRECTIONS**

38474 None.

38475 **SEE ALSO**38476 *sched\_getparam()*, *sched\_setparam()*, *sched\_getscheduler()*, *sched\_rr\_get\_interval()*,  
38477 *sched\_setscheduler()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sched.h>38478 **CHANGE HISTORY**

38479 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38480 **Issue 6**

38481 These functions are marked as part of the Process Scheduling option.

38482 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
38483 implementation does not support the Process Scheduling option.38484 The [ESRCH] error condition has been removed since these functions do not take a *pid*  
38485 argument.38486 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/52 is applied, changing the PS margin  
38487 code in the SYNOPSIS to PS|TPS.

38488 **NAME**

38489 sched\_getparam — get scheduling parameters (**REALTIME**)

38490 **SYNOPSIS**

38491 PS #include <sched.h>

38492 int sched\_getparam(pid\_t pid, struct sched\_param \*param);

38493

38494 **DESCRIPTION**

38495 The *sched\_getparam()* function shall return the scheduling parameters of a process specified by  
38496 *pid* in the **sched\_param** structure pointed to by *param*.

38497 If a process specified by *pid* exists, and if the calling process has permission, the scheduling  
38498 parameters for the process whose process ID is equal to *pid* shall be returned.

38499 If *pid* is zero, the scheduling parameters for the calling process shall be returned. The behavior of  
38500 the *sched\_getparam()* function is unspecified if the value of *pid* is negative.

38501 **RETURN VALUE**

38502 Upon successful completion, the *sched\_getparam()* function shall return zero. If the call to  
38503 *sched\_getparam()* is unsuccessful, the function shall return a value of -1 and set *errno* to indicate  
38504 the error.

38505 **ERRORS**

38506 The *sched\_getparam()* function shall fail if:

38507 [EPERM] The requesting process does not have permission to obtain the scheduling  
38508 parameters of the specified process.

38509 [ESRCH] No process can be found corresponding to that specified by *pid*.

38510 **EXAMPLES**

38511 None.

38512 **APPLICATION USAGE**

38513 None.

38514 **RATIONALE**

38515 None.

38516 **FUTURE DIRECTIONS**

38517 None.

38518 **SEE ALSO**

38519 *sched\_getscheduler()*, *sched\_setparam()*, *sched\_setscheduler()*, the Base Definitions volume of  
38520 IEEE Std 1003.1-2001, <**sched.h**>

38521 **CHANGE HISTORY**

38522 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38523 **Issue 6**

38524 The *sched\_getparam()* function is marked as part of the Process Scheduling option.

38525 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
38526 implementation does not support the Process Scheduling option.

38527 **NAME**38528 sched\_getscheduler — get scheduling policy (**REALTIME**)38529 **SYNOPSIS**

38530 PS #include &lt;sched.h&gt;

38531 int sched\_getscheduler(pid\_t pid);

38532

38533 **DESCRIPTION**

38534 The *sched\_getscheduler()* function shall return the scheduling policy of the process specified by  
 38535 *pid*. If the value of *pid* is negative, the behavior of the *sched\_getscheduler()* function is  
 38536 unspecified.

38537 The values that can be returned by *sched\_getscheduler()* are defined in the <**sched.h**> header.

38538 If a process specified by *pid* exists, and if the calling process has permission, the scheduling  
 38539 policy shall be returned for the process whose process ID is equal to *pid*.

38540 If *pid* is zero, the scheduling policy shall be returned for the calling process.

38541 **RETURN VALUE**

38542 Upon successful completion, the *sched\_getscheduler()* function shall return the scheduling policy  
 38543 of the specified process. If unsuccessful, the function shall return  $-1$  and set *errno* to indicate the  
 38544 error.

38545 **ERRORS**

38546 The *sched\_getscheduler()* function shall fail if:

38547 [EPERM] The requesting process does not have permission to determine the scheduling  
 38548 policy of the specified process.

38549 [ESRCH] No process can be found corresponding to that specified by *pid*.

38550 **EXAMPLES**

38551 None.

38552 **APPLICATION USAGE**

38553 None.

38554 **RATIONALE**

38555 None.

38556 **FUTURE DIRECTIONS**

38557 None.

38558 **SEE ALSO**

38559 *sched\_getparam()*, *sched\_setparam()*, *sched\_setscheduler()*, the Base Definitions volume of  
 38560 IEEE Std 1003.1-2001, <**sched.h**>

38561 **CHANGE HISTORY**

38562 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38563 **Issue 6**

38564 The *sched\_getscheduler()* function is marked as part of the Process Scheduling option.

38565 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 38566 implementation does not support the Process Scheduling option.

38567 **NAME**

38568 sched\_rr\_get\_interval — get execution time limits (**REALTIME**)

38569 **SYNOPSIS**

38570 PS|TPS #include <sched.h>

38571 int sched\_rr\_get\_interval(pid\_t pid, struct timespec \*interval);

38572

38573 **DESCRIPTION**

38574 The *sched\_rr\_get\_interval()* function shall update the **timespec** structure referenced by the  
 38575 *interval* argument to contain the current execution time limit (that is, time quantum) for the  
 38576 process specified by *pid*. If *pid* is zero, the current execution time limit for the calling process  
 38577 shall be returned.

38578 **RETURN VALUE**

38579 If successful, the *sched\_rr\_get\_interval()* function shall return zero. Otherwise, it shall return a  
 38580 value of  $-1$  and set *errno* to indicate the error.

38581 **ERRORS**

38582 The *sched\_rr\_get\_interval()* function shall fail if:

38583 [ESRCH] No process can be found corresponding to that specified by *pid*.

38584 **EXAMPLES**

38585 None.

38586 **APPLICATION USAGE**

38587 None.

38588 **RATIONALE**

38589 None.

38590 **FUTURE DIRECTIONS**

38591 None.

38592 **SEE ALSO**

38593 *sched\_getparam()*, *sched\_get\_priority\_max()*, *sched\_getscheduler()*, *sched\_setparam()*,  
 38594 *sched\_setscheduler()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sched.h>

38595 **CHANGE HISTORY**

38596 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38597 **Issue 6**

38598 The *sched\_rr\_get\_interval()* function is marked as part of the Process Scheduling option.

38599 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 38600 implementation does not support the Process Scheduling option.

38601 IEEE Std 1003.1-2001/Cor 1-2002, XSH/TC1/D6/53 is applied, changing the PS margin code in  
 38602 the SYNOPSIS to PS|TPS.

## 38603 NAME

38604 sched\_setparam — set scheduling parameters (**REALTIME**)

## 38605 SYNOPSIS

38606 PS #include &lt;sched.h&gt;

38607 int sched\_setparam(pid\_t pid, const struct sched\_param \*param);

38608

## 38609 DESCRIPTION

38610 The *sched\_setparam()* function shall set the scheduling parameters of the process specified by *pid*  
 38611 to the values specified by the **sched\_param** structure pointed to by *param*. The value of the  
 38612 *sched\_priority* member in the **sched\_param** structure shall be any integer within the inclusive  
 38613 priority range for the current scheduling policy of the process specified by *pid*. Higher  
 38614 numerical values for the priority represent higher priorities. If the value of *pid* is negative, the  
 38615 behavior of the *sched\_setparam()* function is unspecified.

38616 If a process specified by *pid* exists, and if the calling process has permission, the scheduling  
 38617 parameters shall be set for the process whose process ID is equal to *pid*.

38618 If *pid* is zero, the scheduling parameters shall be set for the calling process.

38619 The conditions under which one process has permission to change the scheduling parameters of  
 38620 another process are implementation-defined.

38621 Implementations may require the requesting process to have the appropriate privilege to set its  
 38622 own scheduling parameters or those of another process.

38623 The target process, whether it is running or not running, shall be moved to the tail of the thread  
 38624 list for its priority.

38625 If the priority of the process specified by the *pid* argument is set higher than that of the lowest  
 38626 priority running process and if the specified process is ready to run, the process specified by the  
 38627 *pid* argument shall preempt a lowest priority running process. Similarly, if the process calling  
 38628 *sched\_setparam()* sets its own priority lower than that of one or more other non-empty process  
 38629 lists, then the process that is the head of the highest priority list shall also preempt the calling  
 38630 process. Thus, in either case, the originating process might not receive notification of the  
 38631 completion of the requested priority change until the higher priority process has executed.

38632 ss If the scheduling policy of the target process is SCHED\_SPORADIC, the value specified by the  
 38633 *sched\_ss\_low\_priority* member of the *param* argument shall be any integer within the inclusive  
 38634 priority range for the sporadic server policy. The *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget*  
 38635 members of the *param* argument shall represent the time parameters to be used by the sporadic  
 38636 server scheduling policy for the target process. The *sched\_ss\_max\_repl* member of the *param*  
 38637 argument shall represent the maximum number of replenishments that are allowed to be  
 38638 pending simultaneously for the process scheduled under this scheduling policy.

38639 The specified *sched\_ss\_repl\_period* shall be greater than or equal to the specified  
 38640 *sched\_ss\_init\_budget* for the function to succeed; if it is not, then the function shall fail.

38641 The value of *sched\_ss\_max\_repl* shall be within the inclusive range [1,{SS\_REPL\_MAX}] for the  
 38642 function to succeed; if not, the function shall fail.

38643 If the scheduling policy of the target process is either SCHED\_FIFO or SCHED\_RR, the  
 38644 *sched\_ss\_low\_priority*, *sched\_ss\_repl\_period*, and *sched\_ss\_init\_budget* members of the *param*  
 38645 argument shall have no effect on the scheduling behavior. If the scheduling policy of this process  
 38646 is not SCHED\_FIFO, SCHED\_RR, or SCHED\_SPORADIC, the effects of these members are  
 38647 implementation-defined; this case includes the SCHED\_OTHER policy.

38648 If the current scheduling policy for the process specified by *pid* is not SCHED\_FIFO,  
 38649 ss SCHED\_RR, or SCHED\_SPORADIC, the result is implementation-defined; this case includes the  
 38650 SCHED\_OTHER policy.

38651 The effect of this function on individual threads is dependent on the scheduling contention  
 38652 scope of the threads:

- 38653 • For threads with system scheduling contention scope, these functions shall have no effect on  
 38654 their scheduling.

- 38655 • For threads with process scheduling contention scope, the threads' scheduling parameters  
 38656 shall not be affected. However, the scheduling of these threads with respect to threads in  
 38657 other processes may be dependent on the scheduling parameters of their process, which are  
 38658 governed using these functions.

38659 If an implementation supports a two-level scheduling model in which library threads are  
 38660 multiplexed on top of several kernel-scheduled entities, then the underlying kernel-scheduled  
 38661 entities for the system contention scope threads shall not be affected by these functions.

38662 The underlying kernel-scheduled entities for the process contention scope threads shall have  
 38663 their scheduling parameters changed to the value specified in *param*. Kernel-scheduled entities  
 38664 for use by process contention scope threads that are created after this call completes shall inherit  
 38665 their scheduling policy and associated scheduling parameters from the process.

38666 This function is not atomic with respect to other threads in the process. Threads may continue to  
 38667 execute while this function call is in the process of changing the scheduling policy for the  
 38668 underlying kernel-scheduled entities used by the process contention scope threads.

38669 **RETURN VALUE**

38670 If successful, the *sched\_setparam()* function shall return zero.

38671 If the call to *sched\_setparam()* is unsuccessful, the priority shall remain unchanged, and the  
 38672 function shall return a value of -1 and set *errno* to indicate the error.

38673 **ERRORS**

38674 The *sched\_setparam()* function shall fail if:

38675 [EINVAL] One or more of the requested scheduling parameters is outside the range  
 38676 defined for the scheduling policy of the specified *pid*.

38677 [EPERM] The requesting process does not have permission to set the scheduling  
 38678 parameters for the specified process, or does not have the appropriate  
 38679 privilege to invoke *sched\_setparam()*.

38680 [ESRCH] No process can be found corresponding to that specified by *pid*.

38681 **EXAMPLES**

38682 None.

38683 **APPLICATION USAGE**

38684 None.

38685 **RATIONALE**

38686 None.

38687 **FUTURE DIRECTIONS**

38688 None.

38689 **SEE ALSO**

38690 *sched\_getparam()*, *sched\_getscheduler()*, *sched\_setscheduler()*, the Base Definitions volume of  
38691 IEEE Std 1003.1-2001, <**sched.h**>

38692 **CHANGE HISTORY**

38693 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38694 **Issue 6**

38695 The *sched\_setparam()* function is marked as part of the Process Scheduling option.

38696 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
38697 implementation does not support the Process Scheduling option.

38698 The following new requirements on POSIX implementations derive from alignment with the  
38699 Single UNIX Specification:

38700 • In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is  
38701 added.

38702 • Sections describing two-level scheduling and atomicity of the function are added.

38703 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

38704 IEEE PASC Interpretation 1003.1 #100 is applied.

## 38705 NAME

38706 sched\_setscheduler — set scheduling policy and parameters (**REALTIME**)

## 38707 SYNOPSIS

38708 PS #include &lt;sched.h&gt;

38709 int sched\_setscheduler(pid\_t pid, int policy,  
38710 const struct sched\_param \*param);

38711

## 38712 DESCRIPTION

38713 The *sched\_setscheduler()* function shall set the scheduling policy and scheduling parameters of  
 38714 the process specified by *pid* to *policy* and the parameters specified in the **sched\_param** structure  
 38715 pointed to by *param*, respectively. The value of the *sched\_priority* member in the **sched\_param**  
 38716 structure shall be any integer within the inclusive priority range for the scheduling policy  
 38717 specified by *policy*. If the value of *pid* is negative, the behavior of the *sched\_setscheduler()*  
 38718 function is unspecified.

38719 The possible values for the *policy* parameter are defined in the <**sched.h**> header.

38720 If a process specified by *pid* exists, and if the calling process has permission, the scheduling  
 38721 policy and scheduling parameters shall be set for the process whose process ID is equal to *pid*.

38722 If *pid* is zero, the scheduling policy and scheduling parameters shall be set for the calling  
 38723 process.

38724 The conditions under which one process has the appropriate privilege to change the scheduling  
 38725 parameters of another process are implementation-defined.

38726 Implementations may require that the requesting process have permission to set its own  
 38727 scheduling parameters or those of another process. Additionally, implementation-defined  
 38728 restrictions may apply as to the appropriate privileges required to set a process' own scheduling  
 38729 policy, or another process' scheduling policy, to a particular value.

38730 The *sched\_setscheduler()* function shall be considered successful if it succeeds in setting the  
 38731 scheduling policy and scheduling parameters of the process specified by *pid* to the values  
 38732 specified by *policy* and the structure pointed to by *param*, respectively.

38733 ss If the scheduling policy specified by *policy* is SCHED\_SPORADIC, the value specified by the  
 38734 *sched\_ss\_low\_priority* member of the *param* argument shall be any integer within the inclusive  
 38735 priority range for the sporadic server policy. The *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget*  
 38736 members of the *param* argument shall represent the time parameters used by the sporadic server  
 38737 scheduling policy for the target process. The *sched\_ss\_max\_repl* member of the *param* argument  
 38738 shall represent the maximum number of replenishments that are allowed to be pending  
 38739 simultaneously for the process scheduled under this scheduling policy.

38740 The specified *sched\_ss\_repl\_period* shall be greater than or equal to the specified  
 38741 *sched\_ss\_init\_budget* for the function to succeed; if it is not, then the function shall fail.

38742 The value of *sched\_ss\_max\_repl* shall be within the inclusive range [1,{SS\_REPL\_MAX}] for the  
 38743 function to succeed; if not, the function shall fail.

38744 If the scheduling policy specified by *policy* is either SCHED\_FIFO or SCHED\_RR, the  
 38745 *sched\_ss\_low\_priority*, *sched\_ss\_repl\_period*, and *sched\_ss\_init\_budget* members of the *param*  
 38746 argument shall have no effect on the scheduling behavior.

38747 The effect of this function on individual threads is dependent on the scheduling contention  
 38748 scope of the threads:

38749           • For threads with system scheduling contention scope, these functions shall have no effect on  
38750 their scheduling.

38751           • For threads with process scheduling contention scope, the threads' scheduling policy and  
38752 associated parameters shall not be affected. However, the scheduling of these threads with  
38753 respect to threads in other processes may be dependent on the scheduling parameters of their  
38754 process, which are governed using these functions.

38755           If an implementation supports a two-level scheduling model in which library threads are  
38756 multiplexed on top of several kernel-scheduled entities, then the underlying kernel-scheduled  
38757 entities for the system contention scope threads shall not be affected by these functions.

38758           The underlying kernel-scheduled entities for the process contention scope threads shall have  
38759 their scheduling policy and associated scheduling parameters changed to the values specified in  
38760 *policy* and *param*, respectively. Kernel-scheduled entities for use by process contention scope  
38761 threads that are created after this call completes shall inherit their scheduling policy and  
38762 associated scheduling parameters from the process.

38763           This function is not atomic with respect to other threads in the process. Threads may continue to  
38764 execute while this function call is in the process of changing the scheduling policy and  
38765 associated scheduling parameters for the underlying kernel-scheduled entities used by the  
38766 process contention scope threads.

#### 38767 RETURN VALUE

38768           Upon successful completion, the function shall return the former scheduling policy of the  
38769 specified process. If the *sched\_setscheduler()* function fails to complete successfully, the policy  
38770 and scheduling parameters shall remain unchanged, and the function shall return a value of  $-1$   
38771 and set *errno* to indicate the error.

#### 38772 ERRORS

38773           The *sched\_setscheduler()* function shall fail if:

38774           [EINVAL]           The value of the *policy* parameter is invalid, or one or more of the parameters  
38775 contained in *param* is outside the valid range for the specified scheduling  
38776 policy.

38777           [EPERM]           The requesting process does not have permission to set either or both of the  
38778 scheduling parameters or the scheduling policy of the specified process.

38779           [ESRCH]           No process can be found corresponding to that specified by *pid*.

#### 38780 EXAMPLES

38781           None.

#### 38782 APPLICATION USAGE

38783           None.

#### 38784 RATIONALE

38785           None.

#### 38786 FUTURE DIRECTIONS

38787           None.

#### 38788 SEE ALSO

38789           *sched\_getparam()*, *sched\_getscheduler()*, *sched\_setparam()*, the Base Definitions volume of  
38790 IEEE Std 1003.1-2001, <*sched.h*>

38791 **CHANGE HISTORY**

38792 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38793 **Issue 6**

38794 The *sched\_setscheduler()* function is marked as part of the Process Scheduling option.

38795 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
38796 implementation does not support the Process Scheduling option.

38797 The following new requirements on POSIX implementations derive from alignment with the  
38798 Single UNIX Specification:

38799 • In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is  
38800 added.

38801 • Sections describing two-level scheduling and atomicity of the function are added.

38802 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

38803 **NAME**

38804 sched\_yield — yield the processor

38805 **SYNOPSIS**

38806 PS|THR #include &lt;sched.h&gt;

38807 int sched\_yield(void);

38808

38809 **DESCRIPTION**38810 The *sched\_yield()* function shall force the running thread to relinquish the processor until it again  
38811 becomes the head of its thread list. It takes no arguments.38812 **RETURN VALUE**38813 The *sched\_yield()* function shall return 0 if it completes successfully; otherwise, it shall return a  
38814 value of -1 and set *errno* to indicate the error.38815 **ERRORS**

38816 No errors are defined.

38817 **EXAMPLES**

38818 None.

38819 **APPLICATION USAGE**

38820 None.

38821 **RATIONALE**

38822 None.

38823 **FUTURE DIRECTIONS**

38824 None.

38825 **SEE ALSO**

38826 The Base Definitions volume of IEEE Std 1003.1-2001, &lt;sched.h&gt;

38827 **CHANGE HISTORY**38828 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
38829 POSIX Threads Extension.38830 **Issue 6**38831 The *sched\_yield()* function is now marked as part of the Process Scheduling and Threads options.

38832 **NAME**38833        **seed48** — seed a uniformly distributed pseudo-random non-negative long integer generator38834 **SYNOPSIS**38835 XSI        `#include <stdlib.h>`38836        `unsigned short *seed48(unsigned short seed16v[3]);`

38837

38838 **DESCRIPTION**38839        Refer to *drand48()*.

38840 **NAME**

38841 seekdir — set the position of a directory stream

38842 **SYNOPSIS**

38843 XSI #include &lt;dirent.h&gt;

38844 void seekdir(DIR \*dirp, long loc);

38845

38846 **DESCRIPTION**

38847 The *seekdir()* function shall set the position of the next *readdir()* operation on the directory  
38848 stream specified by *dirp* to the position specified by *loc*. The value of *loc* should have been  
38849 returned from an earlier call to *telldir()*. The new position reverts to the one associated with the  
38850 directory stream when *telldir()* was performed.

38851 If the value of *loc* was not obtained from an earlier call to *telldir()*, or if a call to *rewinddir()*  
38852 occurred between the call to *telldir()* and the call to *seekdir()*, the results of subsequent calls to  
38853 *readdir()* are unspecified.

38854 **RETURN VALUE**38855 The *seekdir()* function shall not return a value.38856 **ERRORS**

38857 No errors are defined.

38858 **EXAMPLES**

38859 None.

38860 **APPLICATION USAGE**

38861 None.

38862 **RATIONALE**

38863 The original standard developers perceived that there were restrictions on the use of the  
38864 *seekdir()* and *telldir()* functions related to implementation details, and for that reason these  
38865 functions need not be supported on all POSIX-conforming systems. They are required on  
38866 implementations supporting the XSI extension.

38867 One of the perceived problems of implementation is that returning to a given point in a directory  
38868 is quite difficult to describe formally, in spite of its intuitive appeal, when systems that use B-  
38869 trees, hashing functions, or other similar mechanisms to order their directories are considered.  
38870 The definition of *seekdir()* and *telldir()* does not specify whether, when using these interfaces, a  
38871 given directory entry will be seen at all, or more than once.

38872 On systems not supporting these functions, their capability can sometimes be accomplished by  
38873 saving a filename found by *readdir()* and later using *rewinddir()* and a loop on *readdir()* to  
38874 relocate the position from which the filename was saved.

38875 **FUTURE DIRECTIONS**

38876 None.

38877 **SEE ALSO**

38878 *opendir()*, *readdir()*, *telldir()*, the Base Definitions volume of IEEE Std 1003.1-2001, <dirent.h>,  
38879 <stdio.h>, <sys/types.h>

38880 **CHANGE HISTORY**

38881 First released in Issue 2.

38882 **Issue 6**

38883

In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

38884 **NAME**

38885       select — synchronous I/O multiplexing

38886 **SYNOPSIS**

38887       #include <sys/time.h>

```
38888 int select(int nfds, fd_set *restrict readfds,
38889 fd_set *restrict writefds, fd_set *restrict errorfds,
38890 struct timeval *restrict timeout);
```

38891

38892 **DESCRIPTION**

38893       Refer to *pselect()*.

38894 **NAME**

38895 sem\_close — close a named semaphore (**REALTIME**)

38896 **SYNOPSIS**

```
38897 SEM #include <semaphore.h>
```

```
38898 int sem_close(sem_t *sem);
```

38899

38900 **DESCRIPTION**

38901 The *sem\_close()* function shall indicate that the calling process is finished using the named  
38902 semaphore indicated by *sem*. The effects of calling *sem\_close()* for an unnamed semaphore (one  
38903 created by *sem\_init()*) are undefined. The *sem\_close()* function shall deallocate (that is, make  
38904 available for reuse by a subsequent *sem\_open()* by this process) any system resources allocated  
38905 by the system for use by this process for this semaphore. The effect of subsequent use of the  
38906 semaphore indicated by *sem* by this process is undefined. If the semaphore has not been  
38907 removed with a successful call to *sem\_unlink()*, then *sem\_close()* has no effect on the state of the  
38908 semaphore. If the *sem\_unlink()* function has been successfully invoked for *name* after the most  
38909 recent call to *sem\_open()* with *O\_CREAT* for this semaphore, then when all processes that have  
38910 opened the semaphore close it, the semaphore is no longer accessible.

38911 **RETURN VALUE**

38912 Upon successful completion, a value of zero shall be returned. Otherwise, a value of -1 shall be  
38913 returned and *errno* set to indicate the error.

38914 **ERRORS**

38915 The *sem\_close()* function shall fail if:

38916 [EINVAL] The *sem* argument is not a valid semaphore descriptor.

38917 **EXAMPLES**

38918 None.

38919 **APPLICATION USAGE**

38920 The *sem\_close()* function is part of the Semaphores option and need not be available on all  
38921 implementations.

38922 **RATIONALE**

38923 None.

38924 **FUTURE DIRECTIONS**

38925 None.

38926 **SEE ALSO**

38927 *semctl()*, *semget()*, *semop()*, *sem\_init()*, *sem\_open()*, *sem\_unlink()*, the Base Definitions volume of  
38928 IEEE Std 1003.1-2001, <**semaphore.h**>

38929 **CHANGE HISTORY**

38930 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38931 **Issue 6**

38932 The *sem\_close()* function is marked as part of the Semaphores option.

38933 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
38934 implementation does not support the Semaphores option.

38935 **NAME**

38936 sem\_destroy — destroy an unnamed semaphore (**REALTIME**)

38937 **SYNOPSIS**

```
38938 SEM #include <semaphore.h>
```

```
38939 int sem_destroy(sem_t *sem);
```

38940

38941 **DESCRIPTION**

38942 The *sem\_destroy()* function shall destroy the unnamed semaphore indicated by *sem*. Only a  
38943 semaphore that was created using *sem\_init()* may be destroyed using *sem\_destroy()*; the effect of  
38944 calling *sem\_destroy()* with a named semaphore is undefined. The effect of subsequent use of the  
38945 semaphore *sem* is undefined until *sem* is reinitialized by another call to *sem\_init()*.

38946 It is safe to destroy an initialized semaphore upon which no threads are currently blocked. The  
38947 effect of destroying a semaphore upon which other threads are currently blocked is undefined.

38948 **RETURN VALUE**

38949 Upon successful completion, a value of zero shall be returned. Otherwise, a value of -1 shall be  
38950 returned and *errno* set to indicate the error.

38951 **ERRORS**

38952 The *sem\_destroy()* function shall fail if:

38953 [EINVAL] The *sem* argument is not a valid semaphore.

38954 The *sem\_destroy()* function may fail if:

38955 [EBUSY] There are currently processes blocked on the semaphore.

38956 **EXAMPLES**

38957 None.

38958 **APPLICATION USAGE**

38959 The *sem\_destroy()* function is part of the Semaphores option and need not be available on all  
38960 implementations.

38961 **RATIONALE**

38962 None.

38963 **FUTURE DIRECTIONS**

38964 None.

38965 **SEE ALSO**

38966 *semctl()*, *semget()*, *semop()*, *sem\_init()*, *sem\_open()*, the Base Definitions volume of  
38967 IEEE Std 1003.1-2001, <**semaphore.h**>

38968 **CHANGE HISTORY**

38969 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38970 **Issue 6**

38971 The *sem\_destroy()* function is marked as part of the Semaphores option.

38972 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
38973 implementation does not support the Semaphores option.

38974 **NAME**38975 sem\_getvalue — get the value of a semaphore (**REALTIME**)38976 **SYNOPSIS**

38977 SEM #include &lt;semaphore.h&gt;

38978 int sem\_getvalue(sem\_t \*restrict sem, int \*restrict sval);

38979

38980 **DESCRIPTION**

38981 The *sem\_getvalue()* function shall update the location referenced by the *sval* argument to have  
 38982 the value of the semaphore referenced by *sem* without affecting the state of the semaphore. The  
 38983 updated value represents an actual semaphore value that occurred at some unspecified time  
 38984 during the call, but it need not be the actual value of the semaphore when it is returned to the  
 38985 calling process.

38986 If *sem* is locked, then the object to which *sval* points shall either be set to zero or to a negative  
 38987 number whose absolute value represents the number of processes waiting for the semaphore at  
 38988 some unspecified time during the call.

38989 **RETURN VALUE**

38990 Upon successful completion, the *sem\_getvalue()* function shall return a value of zero. Otherwise,  
 38991 it shall return a value of -1 and set *errno* to indicate the error.

38992 **ERRORS**38993 The *sem\_getvalue()* function shall fail if:38994 [EINVAL] The *sem* argument does not refer to a valid semaphore.38995 **EXAMPLES**

38996 None.

38997 **APPLICATION USAGE**

38998 The *sem\_getvalue()* function is part of the Semaphores option and need not be available on all  
 38999 implementations.

39000 **RATIONALE**

39001 None.

39002 **FUTURE DIRECTIONS**

39003 None.

39004 **SEE ALSO**

39005 *semctl()*, *semget()*, *semop()*, *sem\_post()*, *sem\_timedwait()*, *sem\_trywait()*, *sem\_wait()*, the Base  
 39006 Definitions volume of IEEE Std 1003.1-2001, <**semaphore.h**>

39007 **CHANGE HISTORY**

39008 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39009 **Issue 6**39010 The *sem\_getvalue()* function is marked as part of the Semaphores option.

39011 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 39012 implementation does not support the Semaphores option.

39013 The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with  
 39014 IEEE Std 1003.1d-1999.

39015 The **restrict** keyword is added to the *sem\_getvalue()* prototype for alignment with the  
 39016 ISO/IEC 9899:1999 standard.

39017

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/54 is applied.

39018 **NAME**39019 sem\_init — initialize an unnamed semaphore (**REALTIME**)39020 **SYNOPSIS**

39021 SEM #include &lt;semaphore.h&gt;

39022 int sem\_init(sem\_t \*sem, int pshared, unsigned value);

39023

39024 **DESCRIPTION**

39025 The *sem\_init()* function shall initialize the unnamed semaphore referred to by *sem*. The value of  
 39026 the initialized semaphore shall be *value*. Following a successful call to *sem\_init()*, the semaphore  
 39027 may be used in subsequent calls to *sem\_wait()*, *sem\_trywait()*, *sem\_post()*, and *sem\_destroy()*.  
 39028 This semaphore shall remain usable until the semaphore is destroyed.

39029 If the *pshared* argument has a non-zero value, then the semaphore is shared between processes;  
 39030 in this case, any process that can access the semaphore *sem* can use *sem* for performing  
 39031 *sem\_wait()*, *sem\_trywait()*, *sem\_post()*, and *sem\_destroy()* operations.

39032 Only *sem* itself may be used for performing synchronization. The result of referring to copies of  
 39033 *sem* in calls to *sem\_wait()*, *sem\_trywait()*, *sem\_post()*, and *sem\_destroy()* is undefined.

39034 If the *pshared* argument is zero, then the semaphore is shared between threads of the process; any  
 39035 thread in this process can use *sem* for performing *sem\_wait()*, *sem\_trywait()*, *sem\_post()*, and  
 39036 *sem\_destroy()* operations. The use of the semaphore by threads other than those created in the  
 39037 same process is undefined.

39038 Attempting to initialize an already initialized semaphore results in undefined behavior.

39039 **RETURN VALUE**

39040 Upon successful completion, the *sem\_init()* function shall initialize the semaphore in *sem*.  
 39041 Otherwise, it shall return  $-1$  and set *errno* to indicate the error.

39042 **ERRORS**39043 The *sem\_init()* function shall fail if:

39044 [EINVAL] The *value* argument exceeds {SEM\_VALUE\_MAX}.

39045 [ENOSPC] A resource required to initialize the semaphore has been exhausted, or the  
 39046 limit on semaphores ({SEM\_NSEMS\_MAX}) has been reached.

39047 [EPERM] The process lacks the appropriate privileges to initialize the semaphore.

39048 **EXAMPLES**

39049 None.

39050 **APPLICATION USAGE**

39051 The *sem\_init()* function is part of the Semaphores option and need not be available on all  
 39052 implementations.

39053 **RATIONALE**

39054 Although this volume of IEEE Std 1003.1-2001 fails to specify a successful return value, it is  
 39055 likely that a later version may require the implementation to return a value of zero if the call to  
 39056 *sem\_init()* is successful.

39057 **FUTURE DIRECTIONS**

39058 None.

39059 **SEE ALSO**

39060 *sem\_destroy()*, *sem\_post()*, *sem\_timedwait()*, *sem\_trywait()*, *sem\_wait()*, the Base Definitions  
39061 volume of IEEE Std 1003.1-2001, <**semaphore.h**>

39062 **CHANGE HISTORY**

39063 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39064 **Issue 6**

39065 The *sem\_init()* function is marked as part of the Semaphores option.

39066 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
39067 implementation does not support the Semaphores option.

39068 The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with  
39069 IEEE Std 1003.1d-1999.

## 39070 NAME

39071 sem\_open — initialize and open a named semaphore (**REALTIME**)

## 39072 SYNOPSIS

39073 SEM #include &lt;semaphore.h&gt;

39074 sem\_t \*sem\_open(const char \*name, int oflag, ...);

39075

## 39076 DESCRIPTION

39077 The *sem\_open()* function shall establish a connection between a named semaphore and a process.  
 39078 Following a call to *sem\_open()* with semaphore name *name*, the process may reference the  
 39079 semaphore associated with *name* using the address returned from the call. This semaphore may  
 39080 be used in subsequent calls to *sem\_wait()*, *sem\_trywait()*, *sem\_post()*, and *sem\_close()*. The  
 39081 semaphore remains usable by this process until the semaphore is closed by a successful call to  
 39082 *sem\_close()*, *\_exit()*, or one of the *exec* functions.

39083 The *oflag* argument controls whether the semaphore is created or merely accessed by the call to  
 39084 *sem\_open()*. The following flag bits may be set in *oflag*:

39085 **O\_CREAT** This flag is used to create a semaphore if it does not already exist. If **O\_CREAT** is  
 39086 set and the semaphore already exists, then **O\_CREAT** has no effect, except as noted  
 39087 under **O\_EXCL**. Otherwise, *sem\_open()* creates a named semaphore. The **O\_CREAT**  
 39088 flag requires a third and a fourth argument: *mode*, which is of type **mode\_t**, and  
 39089 *value*, which is of type **unsigned**. The semaphore is created with an initial value of  
 39090 *value*. Valid initial values for semaphores are less than or equal to  
 39091 {SEM\_VALUE\_MAX}.

39092 The user ID of the semaphore is set to the effective user ID of the process; the  
 39093 group ID of the semaphore is set to a system default group ID or to the effective  
 39094 group ID of the process. The permission bits of the semaphore are set to the value  
 39095 of the *mode* argument except those set in the file mode creation mask of the  
 39096 process. When bits in *mode* other than the file permission bits are specified, the  
 39097 effect is unspecified.

39098 After the semaphore named *name* has been created by *sem\_open()* with the  
 39099 **O\_CREAT** flag, other processes can connect to the semaphore by calling  
 39100 *sem\_open()* with the same value of *name*.

39101 **O\_EXCL** If **O\_EXCL** and **O\_CREAT** are set, *sem\_open()* fails if the semaphore *name* exists.  
 39102 The check for the existence of the semaphore and the creation of the semaphore if  
 39103 it does not exist are atomic with respect to other processes executing *sem\_open()*  
 39104 with **O\_EXCL** and **O\_CREAT** set. If **O\_EXCL** is set and **O\_CREAT** is not set, the  
 39105 effect is undefined.

39106 If flags other than **O\_CREAT** and **O\_EXCL** are specified in the *oflag* parameter, the  
 39107 effect is unspecified.

39108 The *name* argument points to a string naming a semaphore object. It is unspecified whether the  
 39109 name appears in the file system and is visible to functions that take pathnames as arguments.  
 39110 The *name* argument conforms to the construction rules for a pathname. If *name* begins with the  
 39111 slash character, then processes calling *sem\_open()* with the same value of *name* shall refer to the  
 39112 same semaphore object, as long as that name has not been removed. If *name* does not begin with  
 39113 the slash character, the effect is implementation-defined. The interpretation of slash characters  
 39114 other than the leading slash character in *name* is implementation-defined.

39115 If a process makes multiple successful calls to *sem\_open()* with the same value for *name*, the  
 39116 same semaphore address shall be returned for each such successful call, provided that there

39117 have been no calls to *sem\_unlink()* for this semaphore.

39118 References to copies of the semaphore produce undefined results.

#### 39119 RETURN VALUE

39120 Upon successful completion, the *sem\_open()* function shall return the address of the semaphore.

39121 Otherwise, it shall return a value of SEM\_FAILED and set *errno* to indicate the error. The symbol

39122 SEM\_FAILED is defined in the <semaphore.h> header. No successful return from *sem\_open()*

39123 shall return the value SEM\_FAILED.

#### 39124 ERRORS

39125 If any of the following conditions occur, the *sem\_open()* function shall return SEM\_FAILED and  
39126 set *errno* to the corresponding value:

39127 [EACCES] The named semaphore exists and the permissions specified by *oflag* are  
39128 denied, or the named semaphore does not exist and permission to create the  
39129 named semaphore is denied.

39130 [EEXIST] O\_CREAT and O\_EXCL are set and the named semaphore already exists.

39131 [EINTR] The *sem\_open()* operation was interrupted by a signal.

39132 [EINVAL] The *sem\_open()* operation is not supported for the given name, or O\_CREAT  
39133 was specified in *oflag* and *value* was greater than {SEM\_VALUE\_MAX}.

39134 [EMFILE] Too many semaphore descriptors or file descriptors are currently in use by  
39135 this process.

39136 [ENAMETOOLONG]

39137 The length of the *name* argument exceeds {PATH\_MAX} or a pathname  
39138 component is longer than {NAME\_MAX}.

39139 [ENFILE] Too many semaphores are currently open in the system.

39140 [ENOENT] O\_CREAT is not set and the named semaphore does not exist.

39141 [ENOSPC] There is insufficient space for the creation of the new named semaphore.

#### 39142 EXAMPLES

39143 None.

#### 39144 APPLICATION USAGE

39145 The *sem\_open()* function is part of the Semaphores option and need not be available on all  
39146 implementations.

#### 39147 RATIONALE

39148 Early drafts required an error return value of -1 with the type **sem\_t \*** for the *sem\_open()*  
39149 function, which is not guaranteed to be portable across implementations. The revised text  
39150 provides the symbolic error code SEM\_FAILED to eliminate the type conflict.

#### 39151 FUTURE DIRECTIONS

39152 None.

#### 39153 SEE ALSO

39154 *semctl()*, *semget()*, *semop()*, *sem\_close()*, *sem\_post()*, *sem\_timedwait()*, *sem\_trywait()*, *sem\_unlink()*,

39155 *sem\_wait()*, the Base Definitions volume of IEEE Std 1003.1-2001, <semaphore.h>

#### 39156 CHANGE HISTORY

39157 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39158 **Issue 6**

39159 The *sem\_open()* function is marked as part of the Semaphores option.

39160 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
39161 implementation does not support the Semaphores option.

39162 The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with  
39163 IEEE Std 1003.1d-1999.

39164 **NAME**39165 sem\_post — unlock a semaphore (**REALTIME**)39166 **SYNOPSIS**

39167 SEM #include &lt;semaphore.h&gt;

39168 int sem\_post(sem\_t \*sem);

39169

39170 **DESCRIPTION**39171 The *sem\_post()* function shall unlock the semaphore referenced by *sem* by performing a  
39172 semaphore unlock operation on that semaphore.39173 If the semaphore value resulting from this operation is positive, then no threads were blocked  
39174 waiting for the semaphore to become unlocked; the semaphore value is simply incremented.39175 If the value of the semaphore resulting from this operation is zero, then one of the threads  
39176 blocked waiting for the semaphore shall be allowed to return successfully from its call to  
39177 *sem\_wait()*. If the Process Scheduling option is supported, the thread to be unblocked shall be  
39178 chosen in a manner appropriate to the scheduling policies and parameters in effect for the  
39179 blocked threads. In the case of the schedulers SCHED\_FIFO and SCHED\_RR, the highest  
39180 priority waiting thread shall be unblocked, and if there is more than one highest priority thread  
39181 blocked waiting for the semaphore, then the highest priority thread that has been waiting the  
39182 longest shall be unblocked. If the Process Scheduling option is not defined, the choice of a thread  
39183 to unblock is unspecified.39184 SS If the Process Sporadic Server option is supported, and the scheduling policy is  
39185 SCHED\_SPORADIC, the semantics are as per SCHED\_FIFO above.39186 The *sem\_post()* function shall be reentrant with respect to signals and may be invoked from a  
39187 signal-catching function.39188 **RETURN VALUE**39189 If successful, the *sem\_post()* function shall return zero; otherwise, the function shall return  $-1$   
39190 and set *errno* to indicate the error.39191 **ERRORS**39192 The *sem\_post()* function shall fail if:39193 [EINVAL] The *sem* argument does not refer to a valid semaphore.39194 **EXAMPLES**

39195 None.

39196 **APPLICATION USAGE**39197 The *sem\_post()* function is part of the Semaphores option and need not be available on all  
39198 implementations.39199 **RATIONALE**

39200 None.

39201 **FUTURE DIRECTIONS**

39202 None.

39203 **SEE ALSO**39204 *semctl()*, *semget()*, *semop()*, *sem\_timedwait()*, *sem\_trywait()*, *sem\_wait()*, the Base Definitions  
39205 volume of IEEE Std 1003.1-2001, <semaphore.h>

39206 **CHANGE HISTORY**

39207 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39208 **Issue 6**

39209 The *sem\_post()* function is marked as part of the Semaphores option.

39210 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
39211 implementation does not support the Semaphores option.

39212 The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with  
39213 IEEE Std 1003.1d-1999.

39214 SCHED\_SPORADIC is added to the list of scheduling policies for which the thread that is to be  
39215 unblocked is specified for alignment with IEEE Std 1003.1d-1999.

39216 **NAME**39217 sem\_timedwait — lock a semaphore (**ADVANCED REALTIME**)39218 **SYNOPSIS**

39219 SEM TMO #include &lt;semaphore.h&gt;

39220 #include &lt;time.h&gt;

```
39221 int sem_timedwait(sem_t *restrict sem,
39222 const struct timespec *restrict abs_timeout);
39223
```

39224 **DESCRIPTION**

39225 The *sem\_timedwait()* function shall lock the semaphore referenced by *sem* as in the *sem\_wait()*  
 39226 function. However, if the semaphore cannot be locked without waiting for another process or  
 39227 thread to unlock the semaphore by performing a *sem\_post()* function, this wait shall be  
 39228 terminated when the specified timeout expires.

39229 The timeout shall expire when the absolute time specified by *abs\_timeout* passes, as measured by  
 39230 the clock on which timeouts are based (that is, when the value of that clock equals or exceeds  
 39231 *abs\_timeout*), or if the absolute time specified by *abs\_timeout* has already been passed at the time  
 39232 of the call.

39233 TMR If the Timers option is supported, the timeout shall be based on the `CLOCK_REALTIME` clock. If  
 39234 the Timers option is not supported, the timeout shall be based on the system clock as returned  
 39235 by the *time()* function. The resolution of the timeout shall be the resolution of the clock on which  
 39236 it is based. The **timespec** data type is defined as a structure in the `<time.h>` header.

39237 Under no circumstance shall the function fail with a timeout if the semaphore can be locked  
 39238 immediately. The validity of the *abs\_timeout* need not be checked if the semaphore can be locked  
 39239 immediately.

39240 **RETURN VALUE**

39241 The *sem\_timedwait()* function shall return zero if the calling process successfully performed the  
 39242 semaphore lock operation on the semaphore designated by *sem*. If the call was unsuccessful, the  
 39243 state of the semaphore shall be unchanged, and the function shall return a value of `-1` and set  
 39244 *errno* to indicate the error.

39245 **ERRORS**39246 The *sem\_timedwait()* function shall fail if:39247 [EINVAL] The *sem* argument does not refer to a valid semaphore.

39248 [EINVAL] The process or thread would have blocked, and the *abs\_timeout* parameter  
 39249 specified a nanoseconds field value less than zero or greater than or equal to  
 39250 1 000 million.

39251 [ETIMEDOUT] The semaphore could not be locked before the specified timeout expired.

39252 The *sem\_timedwait()* function may fail if:

39253 [EDEADLK] A deadlock condition was detected.

39254 [EINTR] A signal interrupted this function.

**39255 EXAMPLES**

39256 None.

**39257 APPLICATION USAGE**

39258 Applications using these functions may be subject to priority inversion, as discussed in the Base  
39259 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

39260 The *sem\_timedwait()* function is part of the Semaphores and Timeouts options and need not be  
39261 provided on all implementations.

**39262 RATIONALE**

39263 None.

**39264 FUTURE DIRECTIONS**

39265 None.

**39266 SEE ALSO**

39267 *sem\_post()*, *sem\_trywait()*, *sem\_wait()*, *semctl()*, *semget()*, *semop()*, *time()*, the Base Definitions  
39268 volume of IEEE Std 1003.1-2001, <**semaphore.h**>, <**time.h**>

**39269 CHANGE HISTORY**

39270 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

39271 **NAME**39272 sem\_trywait, sem\_wait — lock a semaphore (**REALTIME**)39273 **SYNOPSIS**

39274 SEM #include &lt;semaphore.h&gt;

39275 int sem\_trywait(sem\_t \*sem);

39276 int sem\_wait(sem\_t \*sem);

39277

39278 **DESCRIPTION**

39279 The *sem\_trywait()* function shall lock the semaphore referenced by *sem* only if the semaphore is  
 39280 currently not locked; that is, if the semaphore value is currently positive. Otherwise, it shall not  
 39281 lock the semaphore.

39282 The *sem\_wait()* function shall lock the semaphore referenced by *sem* by performing a semaphore  
 39283 lock operation on that semaphore. If the semaphore value is currently zero, then the calling  
 39284 thread shall not return from the call to *sem\_wait()* until it either locks the semaphore or the call is  
 39285 interrupted by a signal.

39286 Upon successful return, the state of the semaphore shall be locked and shall remain locked until  
 39287 the *sem\_post()* function is executed and returns successfully.

39288 The *sem\_wait()* function is interruptible by the delivery of a signal.

39289 **RETURN VALUE**

39290 The *sem\_trywait()* and *sem\_wait()* functions shall return zero if the calling process successfully  
 39291 performed the semaphore lock operation on the semaphore designated by *sem*. If the call was  
 39292 unsuccessful, the state of the semaphore shall be unchanged, and the function shall return a  
 39293 value of  $-1$  and set *errno* to indicate the error.

39294 **ERRORS**

39295 The *sem\_trywait()* and *sem\_wait()* functions shall fail if:

39296 [EAGAIN] The semaphore was already locked, so it cannot be immediately locked by the  
 39297 *sem\_trywait()* operation (*sem\_trywait()* only).

39298 [EINVAL] The *sem* argument does not refer to a valid semaphore.

39299 The *sem\_trywait()* and *sem\_wait()* functions may fail if:

39300 [EDEADLK] A deadlock condition was detected.

39301 [EINTR] A signal interrupted this function.

39302 **EXAMPLES**

39303 None.

39304 **APPLICATION USAGE**

39305 Applications using these functions may be subject to priority inversion, as discussed in the Base  
 39306 Definitions volume of IEEE Std 1003.1-2001, Section 3.285, Priority Inversion.

39307 The *sem\_trywait()* and *sem\_wait()* functions are part of the Semaphores option and need not be  
 39308 provided on all implementations.

39309 **RATIONALE**

39310 None.

39311 **FUTURE DIRECTIONS**

39312       None.

39313 **SEE ALSO**39314       *semctl()*, *semget()*, *semop()*, *sem\_post()*, *sem\_timedwait()*, the Base Definitions volume of  
39315       IEEE Std 1003.1-2001, <**semaphore.h**>39316 **CHANGE HISTORY**

39317       First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39318 **Issue 6**39319       The *sem\_trywait()* and *sem\_wait()* functions are marked as part of the Semaphores option.39320       The [ENOSYS] error condition has been removed as stubs need not be provided if an  
39321       implementation does not support the Semaphores option.39322       The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with  
39323       IEEE Std 1003.1d-1999.

39324 **NAME**39325 sem\_unlink — remove a named semaphore (**REALTIME**)39326 **SYNOPSIS**

39327 SEM #include &lt;semaphore.h&gt;

39328 int sem\_unlink(const char \*name);

39329

39330 **DESCRIPTION**

39331 The *sem\_unlink()* function shall remove the semaphore named by the string *name*. If the  
 39332 semaphore named by *name* is currently referenced by other processes, then *sem\_unlink()* shall  
 39333 have no effect on the state of the semaphore. If one or more processes have the semaphore open  
 39334 when *sem\_unlink()* is called, destruction of the semaphore is postponed until all references to the  
 39335 semaphore have been destroyed by calls to *sem\_close()*, *\_exit()*, or *exec*. Calls to *sem\_open()* to  
 39336 recreate or reconnect to the semaphore refer to a new semaphore after *sem\_unlink()* is called. The  
 39337 *sem\_unlink()* call shall not block until all references have been destroyed; it shall return  
 39338 immediately.

39339 **RETURN VALUE**

39340 Upon successful completion, the *sem\_unlink()* function shall return a value of 0. Otherwise, the  
 39341 semaphore shall not be changed and the function shall return a value of -1 and set *errno* to  
 39342 indicate the error.

39343 **ERRORS**39344 The *sem\_unlink()* function shall fail if:

39345 [EACCES] Permission is denied to unlink the named semaphore.

39346 [ENAMETOOLONG]

39347 The length of the *name* argument exceeds {PATH\_MAX} or a pathname  
 39348 component is longer than {NAME\_MAX}.

39349 [ENOENT] The named semaphore does not exist.

39350 **EXAMPLES**

39351 None.

39352 **APPLICATION USAGE**

39353 The *sem\_unlink()* function is part of the Semaphores option and need not be available on all  
 39354 implementations.

39355 **RATIONALE**

39356 None.

39357 **FUTURE DIRECTIONS**

39358 None.

39359 **SEE ALSO**

39360 *semctl()*, *semget()*, *semop()*, *sem\_close()*, *sem\_open()*, the Base Definitions volume of  
 39361 IEEE Std 1003.1-2001, <**semaphore.h**>

39362 **CHANGE HISTORY**

39363 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39364 **Issue 6**39365 The *sem\_unlink()* function is marked as part of the Semaphores option.

39366  
39367

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

39368 **NAME**

39369 sem\_wait — lock a semaphore (**REALTIME**)

39370 **SYNOPSIS**

39371 SEM #include <semaphore.h>

39372 int sem\_wait(sem\_t \*sem);

39373

39374 **DESCRIPTION**

39375 Refer to *sem\_trywait()*.

## 39376 NAME

39377 semctl — XSI semaphore control operations

## 39378 SYNOPSIS

39379 XSI 

```
#include <sys/sem.h>
```

39380 

```
int semctl(int semid, int semnum, int cmd, ...);
```

39381

## 39382 DESCRIPTION

39383 The *semctl()* function operates on XSI semaphores (see the Base Definitions volume of  
 39384 IEEE Std 1003.1-2001, Section 4.15, Semaphore). It is unspecified whether this function  
 39385 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on  
 39386 page 41).

39387 The *semctl()* function provides a variety of semaphore control operations as specified by *cmd*.  
 39388 The fourth argument is optional and depends upon the operation requested. If required, it is of  
 39389 type **union semun**, which the application shall explicitly declare:

```
39390 union semun {
39391 int val;
39392 struct semid_ds *buf;
39393 unsigned short *array;
39394 } arg;
```

39395 The following semaphore control operations as specified by *cmd* are executed with respect to the  
 39396 semaphore specified by *semid* and *semnum*. The level of permission required for each operation  
 39397 is shown with each command; see Section 2.7 (on page 39). The symbolic names for the values  
 39398 of *cmd* are defined in the `<sys/sem.h>` header:

|       |         |                                                                                                                                                                                                                                                                                                                                 |
|-------|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 39399 | GETVAL  | Return the value of <i>semval</i> ; see <code>&lt;sys/sem.h&gt;</code> . Requires read permission.                                                                                                                                                                                                                              |
| 39400 | SETVAL  | Set the value of <i>semval</i> to <i>arg.val</i> , where <i>arg</i> is the value of the fourth argument to <i>semctl()</i> . When this command is successfully executed, the <i>semadj</i> value corresponding to the specified semaphore in all processes is cleared. Requires alter permission; see Section 2.7 (on page 39). |
| 39401 |         |                                                                                                                                                                                                                                                                                                                                 |
| 39402 |         |                                                                                                                                                                                                                                                                                                                                 |
| 39403 |         |                                                                                                                                                                                                                                                                                                                                 |
| 39404 | GETPID  | Return the value of <i>sempid</i> . Requires read permission.                                                                                                                                                                                                                                                                   |
| 39405 | GETNCNT | Return the value of <i>semmcnt</i> . Requires read permission.                                                                                                                                                                                                                                                                  |
| 39406 | GETZCNT | Return the value of <i>semzcnt</i> . Requires read permission.                                                                                                                                                                                                                                                                  |

39407 The following values of *cmd* operate on each *semval* in the set of semaphores:

|       |        |                                                                                                                                                                                                                                                                                                                                                                     |
|-------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 39408 | GETALL | Return the value of <i>semval</i> for each semaphore in the semaphore set and place into the array pointed to by <i>arg.array</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> . Requires read permission.                                                                                                                                          |
| 39409 |        |                                                                                                                                                                                                                                                                                                                                                                     |
| 39410 |        |                                                                                                                                                                                                                                                                                                                                                                     |
| 39411 | SETALL | Set the value of <i>semval</i> for each semaphore in the semaphore set according to the array pointed to by <i>arg.array</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> . When this command is successfully executed, the <i>semadj</i> values corresponding to each specified semaphore in all processes are cleared. Requires alter permission. |
| 39412 |        |                                                                                                                                                                                                                                                                                                                                                                     |
| 39413 |        |                                                                                                                                                                                                                                                                                                                                                                     |
| 39414 |        |                                                                                                                                                                                                                                                                                                                                                                     |
| 39415 |        |                                                                                                                                                                                                                                                                                                                                                                     |

39416 The following values of *cmd* are also available:

|       |          |                                                                                                                                                                                                                                                                        |
|-------|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 39417 | IPC_STAT | Place the current value of each member of the <b>semid_ds</b> data structure associated with <i>semid</i> into the structure pointed to by <i>arg.buf</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> . The contents of this structure are defined in |
| 39418 |          |                                                                                                                                                                                                                                                                        |
| 39419 |          |                                                                                                                                                                                                                                                                        |

|       |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 39420 |                     | <sys/sem.h>. Requires read permission.                                                                                                                                                                                                                                                                                                                                                                                                            |
| 39421 | IPC_SET             | Set the value of the following members of the <b>semid_ds</b> data structure associated with <i>semid</i> to the corresponding value found in the structure pointed to by <i>arg.buf</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> :                                                                                                                                                                                           |
| 39422 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39423 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39424 |                     | <i>sem_perm.uid</i>                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 39425 |                     | <i>sem_perm.gid</i>                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 39426 |                     | <i>sem_perm.mode</i>                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 39427 |                     | The mode bits specified in Section 2.7.1 (on page 40) are copied into the corresponding bits of the <i>sem_perm.mode</i> associated with <i>semid</i> . The stored values of any other bits are unspecified.                                                                                                                                                                                                                                      |
| 39428 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39429 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39430 |                     | This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the <b>semid_ds</b> data structure associated with <i>semid</i> .                                                                                                                                                                    |
| 39431 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39432 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39433 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39434 | IPC_RMID            | Remove the semaphore identifier specified by <i>semid</i> from the system and destroy the set of semaphores and <b>semid_ds</b> data structure associated with it. This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the <b>semid_ds</b> data structure associated with <i>semid</i> . |
| 39435 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39436 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39437 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39438 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39439 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39440 | <b>RETURN VALUE</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39441 |                     | If successful, the value returned by <i>semctl()</i> depends on <i>cmd</i> as follows:                                                                                                                                                                                                                                                                                                                                                            |
| 39442 | GETVAL              | The value of <i>semval</i> .                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 39443 | GETPID              | The value of <i>sempid</i> .                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 39444 | GETNCNT             | The value of <i>semmcnt</i> .                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 39445 | GETZCNT             | The value of <i>semzcnt</i> .                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 39446 | All others          | 0.                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 39447 |                     | Otherwise, <i>semctl()</i> shall return $-1$ and set <i>errno</i> to indicate the error.                                                                                                                                                                                                                                                                                                                                                          |
| 39448 | <b>ERRORS</b>       |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39449 |                     | The <i>semctl()</i> function shall fail if:                                                                                                                                                                                                                                                                                                                                                                                                       |
| 39450 | [EACCES]            | Operation permission is denied to the calling process; see Section 2.7 (on page 39).                                                                                                                                                                                                                                                                                                                                                              |
| 39451 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39452 | [EINVAL]            | The value of <i>semid</i> is not a valid semaphore identifier, or the value of <i>semnum</i> is less than 0 or greater than or equal to <i>sem_nsems</i> , or the value of <i>cmd</i> is not a valid command.                                                                                                                                                                                                                                     |
| 39453 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39454 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39455 | [EPERM]             | The argument <i>cmd</i> is equal to IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the data structure associated with <i>semid</i> .                                                                                                                                       |
| 39456 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39457 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39458 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 39459 | [ERANGE]            | The argument <i>cmd</i> is equal to SETVAL or SETALL and the value to which <i>semval</i> is to be set is greater than the system-imposed maximum.                                                                                                                                                                                                                                                                                                |
| 39460 |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

39461 **EXAMPLES**

39462 None.

39463 **APPLICATION USAGE**

39464 The fourth parameter in the SYNOPSIS section is now specified as ". . ." in order to avoid a  
39465 clash with the ISO C standard when referring to the union *semun* (as defined in Issue 3) and for  
39466 backwards-compatibility.

39467 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
39468 Application developers who need to use IPC should design their applications so that modules  
39469 using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the  
39470 alternative interfaces.

39471 **RATIONALE**

39472 None.

39473 **FUTURE DIRECTIONS**

39474 None.

39475 **SEE ALSO**

39476 Section 2.7 (on page 39), Section 2.8 (on page 41), *semget()*, *semop()*, *sem\_close()*, *sem\_destroy()*,  
39477 *sem\_getvalue()*, *sem\_init()*, *sem\_open()*, *sem\_post()*, *sem\_unlink()*, *sem\_wait()*, the Base Definitions  
39478 volume of IEEE Std 1003.1-2001, <sys/sem.h>

39479 **CHANGE HISTORY**

39480 First released in Issue 2. Derived from Issue 2 of the SVID.

39481 **Issue 5**

39482 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
39483 DIRECTIONS to the APPLICATION USAGE section.

39484 **NAME**39485 `semget` — get set of XSI semaphores39486 **SYNOPSIS**39487 XSI 

```
#include <sys/sem.h>
```

39488 

```
int semget(key_t key, int nsems, int semflg);
```

39489

39490 **DESCRIPTION**

39491 The `semget()` function operates on XSI semaphores (see the Base Definitions volume of  
 39492 IEEE Std 1003.1-2001, Section 4.15, Semaphore). It is unspecified whether this function  
 39493 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on  
 39494 page 41).

39495 The `semget()` function shall return the semaphore identifier associated with *key*.

39496 A semaphore identifier with its associated **semid\_ds** data structure and its associated set of  
 39497 *nsems* semaphores (see <sys/sem.h>) is created for *key* if one of the following is true:

- 39498 • The argument *key* is equal to `IPC_PRIVATE`.
- 39499 • The argument *key* does not already have a semaphore identifier associated with it and (*semflg*  
 39500 & `IPC_CREAT`) is non-zero.

39501 Upon creation, the **semid\_ds** data structure associated with the new semaphore identifier is  
 39502 initialized as follows:

- 39503 • In the operation permissions structure *sem\_perm.cuid*, *sem\_perm.uid*, *sem\_perm.cgid*, and  
 39504 *sem\_perm.gid* shall be set equal to the effective user ID and effective group ID, respectively, of  
 39505 the calling process.
- 39506 • The low-order 9 bits of *sem\_perm.mode* shall be set equal to the low-order 9 bits of *semflg*.
- 39507 • The variable *sem\_nsems* shall be set equal to the value of *nsems*.
- 39508 • The variable *sem\_otime* shall be set equal to 0 and *sem\_ctime* shall be set equal to the current  
 39509 time.
- 39510 • The data structure associated with each semaphore in the set shall not be initialized. The  
 39511 *semctl()* function with the command `SETVAL` or `SETALL` can be used to initialize each  
 39512 semaphore.

39513 **RETURN VALUE**

39514 Upon successful completion, `semget()` shall return a non-negative integer, namely a semaphore  
 39515 identifier; otherwise, it shall return `-1` and set *errno* to indicate the error.

39516 **ERRORS**

39517 The `semget()` function shall fail if:

- |       |          |                                                                                                                                                                                                                                                                                               |
|-------|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 39518 | [EACCES] | A semaphore identifier exists for <i>key</i> , but operation permission as specified by the low-order 9 bits of <i>semflg</i> would not be granted; see Section 2.7 (on page 39).                                                                                                             |
| 39519 |          |                                                                                                                                                                                                                                                                                               |
| 39520 |          |                                                                                                                                                                                                                                                                                               |
| 39521 | [EEXIST] | A semaphore identifier exists for the argument <i>key</i> but (( <i>semflg</i> & <code>IPC_CREAT</code> ) && ( <i>semflg</i> & <code>IPC_EXCL</code> )) is non-zero.                                                                                                                          |
| 39522 |          |                                                                                                                                                                                                                                                                                               |
| 39523 | [EINVAL] | The value of <i>nsems</i> is either less than or equal to 0 or greater than the system-imposed limit, or a semaphore identifier exists for the argument <i>key</i> , but the number of semaphores in the set associated with it is less than <i>nsems</i> and <i>nsems</i> is not equal to 0. |
| 39524 |          |                                                                                                                                                                                                                                                                                               |
| 39525 |          |                                                                                                                                                                                                                                                                                               |
| 39526 |          |                                                                                                                                                                                                                                                                                               |

39527 [ENOENT] A semaphore identifier does not exist for the argument *key* and (*semflg*  
 39528 &IPC\_CREAT) is equal to 0.

39529 [ENOSPC] A semaphore identifier is to be created but the system-imposed limit on the  
 39530 maximum number of allowed semaphores system-wide would be exceeded.

39531 **EXAMPLES**39532 **Creating a Semaphore Identifier**

39533 The following example gets a unique semaphore key using the *ftok()* function, then gets a  
 39534 semaphore ID associated with that key using the *semget()* function (the first call also tests to  
 39535 make sure the semaphore exists). If the semaphore does not exist, the program creates it, as  
 39536 shown by the second call to *semget()*. In creating the semaphore for the queuing process, the  
 39537 program attempts to create one semaphore with read/write permission for all. It also uses the  
 39538 IPC\_EXCL flag, which forces *semget()* to fail if the semaphore already exists.

39539 After creating the semaphore, the program uses a call to *semop()* to initialize it to the values in  
 39540 the *sbuf* array. The number of processes that can execute concurrently without queuing is  
 39541 initially set to 2. The final call to *semget()* creates a semaphore identifier that can be used later in  
 39542 the program.

```

39543 #include <sys/types.h>
39544 #include <stdio.h>
39545 #include <sys/ipc.h>
39546 #include <sys/sem.h>
39547 #include <sys/stat.h>
39548 #include <errno.h>
39549 #include <unistd.h>
39550 #include <stdlib.h>
39551 #include <pwd.h>
39552 #include <fcntl.h>
39553 #include <limits.h>
39554 ...
39555 key_t semkey;
39556 int semid, pfd, fv;
39557 struct sembuf sbuf;
39558 char *lgn;
39559 char filename[PATH_MAX+1];
39560 struct stat outstat;
39561 struct passwd *pw;
39562 ...
39563 /* Get unique key for semaphore. */
39564 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
39565 perror("IPC error: ftok"); exit(1);
39566 }
39567 /* Get semaphore ID associated with this key. */
39568 if ((semid = semget(semkey, 0, 0)) == -1) {
39569 /* Semaphore does not exist - Create. */
39570 if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
39571 S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1)
39572 {
39573 /* Initialize the semaphore. */
39574 sbuf.sem_num = 0;

```

```

39575 sbuf.sem_op = 2; /* This is the number of runs
39576 without queuing. */
39577 sbuf.sem_flg = 0;
39578 if (semop(semid, &sbuf, 1) == -1) {
39579 perror("IPC error: semop"); exit(1);
39580 }
39581 }
39582 else if (errno == EEXIST) {
39583 if ((semid = semget(semkey, 0, 0)) == -1) {
39584 perror("IPC error 1: semget"); exit(1);
39585 }
39586 }
39587 else {
39588 perror("IPC error 2: semget"); exit(1);
39589 }
39590 }
39591 ...

```

**39592 APPLICATION USAGE**

39593 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
 39594 Application developers who need to use IPC should design their applications so that modules  
 39595 using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the  
 39596 alternative interfaces.

**39597 RATIONALE**

39598 None.

**39599 FUTURE DIRECTIONS**

39600 None.

**39601 SEE ALSO**

39602 Section 2.7 (on page 39), Section 2.8 (on page 41), *semctl()*, *semop()*, *sem\_close()*, *sem\_destroy()*,  
 39603 *sem\_getvalue()*, *sem\_init()*, *sem\_open()*, *sem\_post()*, *sem\_unlink()*, *sem\_wait()*, the Base Definitions  
 39604 volume of IEEE Std 1003.1-2001, <sys/sem.h>

**39605 CHANGE HISTORY**

39606 First released in Issue 2. Derived from Issue 2 of the SVID.

**39607 Issue 5**

39608 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
 39609 DIRECTIONS to a new APPLICATION USAGE section.

## 39610 NAME

39611 semop — XSI semaphore operations

## 39612 SYNOPSIS

39613 XSI #include &lt;sys/sem.h&gt;

39614 int semop(int *semid*, struct sembuf \**sops*, size\_t *nsops*);

39615

## 39616 DESCRIPTION

39617 The *semop()* function operates on XSI semaphores (see the Base Definitions volume of  
 39618 IEEE Std 1003.1-2001, Section 4.15, Semaphore). It is unspecified whether this function  
 39619 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on  
 39620 page 41).

39621 The *semop()* function shall perform atomically a user-defined array of semaphore operations on  
 39622 the set of semaphores associated with the semaphore identifier specified by the argument *semid*.

39623 The argument *sops* is a pointer to a user-defined array of semaphore operation structures. The  
 39624 implementation shall not modify elements of this array unless the application uses  
 39625 implementation-defined extensions.

39626 The argument *nsops* is the number of such structures in the array.

39627 Each structure, **sembuf**, includes the following members:

39628

39629

| Member Type | Member Name    | Description          |
|-------------|----------------|----------------------|
| short       | <i>sem_num</i> | Semaphore number.    |
| short       | <i>sem_op</i>  | Semaphore operation. |
| short       | <i>sem_flg</i> | Operation flags.     |

39630

39631

39632

39633 Each semaphore operation specified by *sem\_op* is performed on the corresponding semaphore  
 39634 specified by *semid* and *sem\_num*.

39635 The variable *sem\_op* specifies one of three semaphore operations:

39636 1. If *sem\_op* is a negative integer and the calling process has alter permission, one of the  
 39637 following shall occur:

39638 • If *semval* (see <sys/sem.h>) is greater than or equal to the absolute value of *sem\_op*, the  
 39639 absolute value of *sem\_op* is subtracted from *semval*. Also, if (*sem\_flg* &SEM\_UNDO) is  
 39640 non-zero, the absolute value of *sem\_op* shall be added to the calling process' *semadj*  
 39641 value for the specified semaphore.

39642 • If *semval* is less than the absolute value of *sem\_op* and (*sem\_flg* &IPC\_NOWAIT) is non-  
 39643 zero, *semop()* shall return immediately.

39644 • If *semval* is less than the absolute value of *sem\_op* and (*sem\_flg* &IPC\_NOWAIT) is 0,  
 39645 *semop()* shall increment the *semncnt* associated with the specified semaphore and  
 39646 suspend execution of the calling thread until one of the following conditions occurs:

39647 — The value of *semval* becomes greater than or equal to the absolute value of *sem\_op*.  
 39648 When this occurs, the value of *semncnt* associated with the specified semaphore  
 39649 shall be decremented, the absolute value of *sem\_op* shall be subtracted from *semval*  
 39650 and, if (*sem\_flg* &SEM\_UNDO) is non-zero, the absolute value of *sem\_op* shall be  
 39651 added to the calling process' *semadj* value for the specified semaphore.

39652 — The *semid* for which the calling thread is awaiting action is removed from the  
 39653 system. When this occurs, *errno* shall be set equal to [EIDRM] and *-1* shall be

- 39654 returned.
- 39655 — The calling thread receives a signal that is to be caught. When this occurs, the value  
39656 of *semncnt* associated with the specified semaphore shall be decremented, and the  
39657 calling thread shall resume execution in the manner prescribed in *sigaction()*.
- 39658 2. If *sem\_op* is a positive integer and the calling process has alter permission, the value of  
39659 *sem\_op* shall be added to *semval* and, if (*sem\_flg* &SEM\_UNDO) is non-zero, the value of  
39660 *sem\_op* shall be subtracted from the calling process' *semadj* value for the specified  
39661 semaphore.
- 39662 3. If *sem\_op* is 0 and the calling process has read permission, one of the following shall occur:
- 39663 • If *semval* is 0, *semop()* shall return immediately.
- 39664 • If *semval* is non-zero and (*sem\_flg* &IPC\_NOWAIT) is non-zero, *semop()* shall return  
39665 immediately.
- 39666 • If *semval* is non-zero and (*sem\_flg* &IPC\_NOWAIT) is 0, *semop()* shall increment the  
39667 *semzcnt* associated with the specified semaphore and suspend execution of the calling  
39668 thread until one of the following occurs:
- 39669 — The value of *semval* becomes 0, at which time the value of *semzcnt* associated with  
39670 the specified semaphore shall be decremented.
- 39671 — The *semid* for which the calling thread is awaiting action is removed from the  
39672 system. When this occurs, *errno* shall be set equal to [EIDRM] and -1 shall be  
39673 returned.
- 39674 — The calling thread receives a signal that is to be caught. When this occurs, the value  
39675 of *semzcnt* associated with the specified semaphore shall be decremented, and the  
39676 calling thread shall resume execution in the manner prescribed in *sigaction()*.
- 39677 Upon successful completion, the value of *sempid* for each semaphore specified in the array  
39678 pointed to by *sops* shall be set equal to the process ID of the calling process.

39679 **RETURN VALUE**

39680 Upon successful completion, *semop()* shall return 0; otherwise, it shall return -1 and set *errno* to  
39681 indicate the error.

39682 **ERRORS**

39683 The *semop()* function shall fail if:

- 39684 [E2BIG] The value of *nsops* is greater than the system-imposed maximum.
- 39685 [EACCES] Operation permission is denied to the calling process; see Section 2.7 (on page  
39686 39).
- 39687 [EAGAIN] The operation would result in suspension of the calling process but (*sem\_flg*  
39688 &IPC\_NOWAIT) is non-zero.
- 39689 [EFBIG] The value of *sem\_num* is less than 0 or greater than or equal to the number of  
39690 semaphores in the set associated with *semid*.
- 39691 [EIDRM] The semaphore identifier *semid* is removed from the system.
- 39692 [EINTR] The *semop()* function was interrupted by a signal.
- 39693 [EINVAL] The value of *semid* is not a valid semaphore identifier, or the number of  
39694 individual semaphores for which the calling process requests a SEM\_UNDO  
39695 would exceed the system-imposed limit.

39696 [ENOSPC] The limit on the number of individual processes requesting a SEM\_UNDO  
 39697 would be exceeded.

39698 [ERANGE] An operation would cause a *semval* to overflow the system-imposed limit, or  
 39699 an operation would cause a *semadj* value to overflow the system-imposed  
 39700 limit.

## 39701 EXAMPLES

### 39702 Setting Values in Semaphores

39703 The following example sets the values of the two semaphores associated with the *semid*  
 39704 identifier to the values contained in the *sb* array.

```
39705 #include <sys/sem.h>
39706 ...
39707 int semid;
39708 struct sembuf sb[2];
39709 int nsops = 2;
39710 int result;

39711 /* Adjust value of semaphore in the semaphore array semid. */
39712 sb[0].sem_num = 0;
39713 sb[0].sem_op = -1;
39714 sb[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
39715 sb[1].sem_num = 1;
39716 sb[1].sem_op = 1;
39717 sb[1].sem_flg = 0;

39718 result = semop(semid, sb, nsops);
```

### 39719 Creating a Semaphore Identifier

39720 The following example gets a unique semaphore key using the *ftok()* function, then gets a  
 39721 semaphore ID associated with that key using the *semget()* function (the first call also tests to  
 39722 make sure the semaphore exists). If the semaphore does not exist, the program creates it, as  
 39723 shown by the second call to *semget()*. In creating the semaphore for the queuing process, the  
 39724 program attempts to create one semaphore with read/write permission for all. It also uses the  
 39725 IPC\_EXCL flag, which forces *semget()* to fail if the semaphore already exists.

39726 After creating the semaphore, the program uses a call to *semop()* to initialize it to the values in  
 39727 the *sbuf* array. The number of processes that can execute concurrently without queuing is  
 39728 initially set to 2. The final call to *semget()* creates a semaphore identifier that can be used later in  
 39729 the program.

39730 The final call to *semop()* acquires the semaphore and waits until it is free; the SEM\_UNDO  
 39731 option releases the semaphore when the process exits, waiting until there are less than two  
 39732 processes running concurrently.

```
39733 #include <sys/types.h>
39734 #include <stdio.h>
39735 #include <sys/ipc.h>
39736 #include <sys/sem.h>
39737 #include <sys/stat.h>
39738 #include <errno.h>
39739 #include <unistd.h>
39740 #include <stdlib.h>
```

```

39741 #include <pwd.h>
39742 #include <fcntl.h>
39743 #include <limits.h>
39744 ...
39745 key_t semkey;
39746 int semid, pfd, fv;
39747 struct sembuf sbuf;
39748 char *lgn;
39749 char filename[PATH_MAX+1];
39750 struct stat outstat;
39751 struct passwd *pw;
39752 ...
39753 /* Get unique key for semaphore. */
39754 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
39755 perror("IPC error: ftok"); exit(1);
39756 }
39757 /* Get semaphore ID associated with this key. */
39758 if ((semid = semget(semkey, 0, 0)) == -1) {
39759 /* Semaphore does not exist - Create. */
39760 if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
39761 S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1)
39762 {
39763 /* Initialize the semaphore. */
39764 sbuf.sem_num = 0;
39765 sbuf.sem_op = 2; /* This is the number of runs without queuing. */
39766 sbuf.sem_flg = 0;
39767 if (semop(semid, &sbuf, 1) == -1) {
39768 perror("IPC error: semop"); exit(1);
39769 }
39770 }
39771 else if (errno == EEXIST) {
39772 if ((semid = semget(semkey, 0, 0)) == -1) {
39773 perror("IPC error 1: semget"); exit(1);
39774 }
39775 }
39776 else {
39777 perror("IPC error 2: semget"); exit(1);
39778 }
39779 }
39780 ...
39781 sbuf.sem_num = 0;
39782 sbuf.sem_op = -1;
39783 sbuf.sem_flg = SEM_UNDO;
39784 if (semop(semid, &sbuf, 1) == -1) {
39785 perror("IPC Error: semop"); exit(1);
39786 }

```

### 39787 APPLICATION USAGE

39788 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
39789 Application developers who need to use IPC should design their applications so that modules  
39790 using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the  
39791 alternative interfaces.

39792 **RATIONALE**

39793           None.

39794 **FUTURE DIRECTIONS**

39795           None.

39796 **SEE ALSO**

39797           Section 2.7 (on page 39), Section 2.8 (on page 41), *exec*, *exit()*, *fork()*, *semctl()*, *semget()*,  
39798           *sem\_close()*, *sem\_destroy()*, *sem\_getvalue()*, *sem\_init()*, *sem\_open()*, *sem\_post()*, *sem\_unlink()*,  
39799           *sem\_wait()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<sys/ipc.h>`, `<sys/sem.h>`,  
39800           `<sys/types.h>`

39801 **CHANGE HISTORY**

39802           First released in Issue 2. Derived from Issue 2 of the SVID.

39803 **Issue 5**

39804           The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
39805           DIRECTIONS to a new APPLICATION USAGE section.

39806 **NAME**

39807        send — send a message on a socket

39808 **SYNOPSIS**

39809        #include &lt;sys/socket.h&gt;

39810        ssize\_t send(int *socket*, const void \**buffer*, size\_t *length*, int *flags*);39811 **DESCRIPTION**39812        The *send()* function shall initiate transmission of a message from the specified socket to its peer.39813        The *send()* function shall send a message only when the socket is connected (including when the peer of a connectionless socket has been set via *connect()*).39815        The *send()* function takes the following arguments:39816        *socket*                Specifies the socket file descriptor.39817        *buffer*                Points to the buffer containing the message to send.39818        *length*                Specifies the length of the message in bytes.39819        *flags*                 Specifies the type of message transmission. Values of this argument are formed by logically OR'ing zero or more of the following flags:

39821                MSG\_EOR        Terminates a record (if supported by the protocol).

39822                MSG\_OOB        Sends out-of-band data on sockets that support out-of-band communications. The significance and semantics of out-of-band data are protocol-specific.

39825        The length of the message to be sent is specified by the *length* argument. If the message is too long to pass through the underlying protocol, *send()* shall fail and no data shall be transmitted.39827        Successful completion of a call to *send()* does not guarantee delivery of the message. A return value of  $-1$  indicates only locally-detected errors.39829        If space is not available at the sending socket to hold the message to be transmitted, and the socket file descriptor does not have O\_NONBLOCK set, *send()* shall block until space is available. If space is not available at the sending socket to hold the message to be transmitted, and the socket file descriptor does have O\_NONBLOCK set, *send()* shall fail. The *select()* and *poll()* functions can be used to determine when it is possible to send more data.39834        The socket in use may require the process to have appropriate privileges to use the *send()* function.39836 **RETURN VALUE**39837        Upon successful completion, *send()* shall return the number of bytes sent. Otherwise,  $-1$  shall be returned and *errno* set to indicate the error.39839 **ERRORS**39840        The *send()* function shall fail if:

39841        [EAGAIN] or [EWOULDBLOCK]

39842                The socket's file descriptor is marked O\_NONBLOCK and the requested operation would block.

39844        [EBADF]               The *socket* argument is not a valid file descriptor.

39845        [ECONNRESET]        A connection was forcibly closed by a peer.

39846        [EDESTADDRREQ]

39847                The socket is not connection-mode and no peer address is set.

- 39848 [EINTR] A signal interrupted *send()* before any data was transmitted.
- 39849 [EMSGSIZE] The message is too large to be sent all at once, as the socket requires.
- 39850 [ENOTCONN] The socket is not connected or otherwise has not had the peer pre-specified.
- 39851 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 39852 [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or  
39853 more of the values set in *flags*.
- 39854 [EPIPE] The socket is shut down for writing, or the socket is connection-mode and is  
39855 no longer connected. In the latter case, and if the socket is of type  
39856 SOCK\_STREAM, the SIGPIPE signal is generated to the calling thread.
- 39857 The *send()* function may fail if:
- 39858 [EACCES] The calling process does not have the appropriate privileges.
- 39859 [EIO] An I/O error occurred while reading from or writing to the file system.
- 39860 [ENETDOWN] The local network interface used to reach the destination is down.
- 39861 [ENETUNREACH]  
39862 No route to the network is present.
- 39863 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 39864 **EXAMPLES**
- 39865 None.
- 39866 **APPLICATION USAGE**
- 39867 The *send()* function is equivalent to *sendto()* with a null pointer *dest\_len* argument, and to *write()*  
39868 if no flags are used.
- 39869 **RATIONALE**
- 39870 None.
- 39871 **FUTURE DIRECTIONS**
- 39872 None.
- 39873 **SEE ALSO**
- 39874 *connect()*, *getsockopt()*, *poll()*, *recv()*, *recvfrom()*, *recvmsg()*, *select()*, *sendmsg()*, *sendto()*,  
39875 *setsockopt()*, *shutdown()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
39876 <sys/socket.h>
- 39877 **CHANGE HISTORY**
- 39878 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

39879 **NAME**

39880 sendmsg — send a message on a socket using a message structure

39881 **SYNOPSIS**

39882 #include &lt;sys/socket.h&gt;

39883 ssize\_t sendmsg(int *socket*, const struct msghdr \**message*, int *flags*);39884 **DESCRIPTION**

39885 The *sendmsg()* function shall send a message through a connection-mode or connectionless-  
 39886 mode socket. If the socket is connectionless-mode, the message shall be sent to the address  
 39887 specified by **msghdr**. If the socket is connection-mode, the destination address in **msghdr** shall  
 39888 be ignored.

39889 The *sendmsg()* function takes the following arguments:

|       |                |                                                                                                                                                                                                                                                 |
|-------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 39890 | <i>socket</i>  | Specifies the socket file descriptor.                                                                                                                                                                                                           |
| 39891 | <i>message</i> | Points to a <b>msghdr</b> structure, containing both the destination address and the buffers for the outgoing message. The length and format of the address depend on the address family of the socket. The <i>msg_flags</i> member is ignored. |
| 39892 |                |                                                                                                                                                                                                                                                 |
| 39893 |                |                                                                                                                                                                                                                                                 |
| 39894 | <i>flags</i>   | Specifies the type of message transmission. The application may specify 0 or the following flag:                                                                                                                                                |
| 39895 |                |                                                                                                                                                                                                                                                 |
| 39896 | MSG_EOR        | Terminates a record (if supported by the protocol).                                                                                                                                                                                             |
| 39897 | MSG_OOB        | Sends out-of-band data on sockets that support out-of-band data. The significance and semantics of out-of-band data are protocol-specific.                                                                                                      |
| 39898 |                |                                                                                                                                                                                                                                                 |
| 39899 |                |                                                                                                                                                                                                                                                 |

39900 The *msg\_iov* and *msg\_iovlen* fields of *message* specify zero or more buffers containing the data to  
 39901 be sent. *msg\_iov* points to an array of **iovec** structures; *msg\_iovlen* shall be set to the dimension of  
 39902 this array. In each **iovec** structure, the *iov\_base* field specifies a storage area and the *iov\_len* field  
 39903 gives its size in bytes. Some of these sizes can be zero. The data from each storage area indicated  
 39904 by *msg\_iov* is sent in turn.

39905 Successful completion of a call to *sendmsg()* does not guarantee delivery of the message. A  
 39906 return value of  $-1$  indicates only locally-detected errors.

39907 If space is not available at the sending socket to hold the message to be transmitted and the  
 39908 socket file descriptor does not have **O\_NONBLOCK** set, the *sendmsg()* function shall block until  
 39909 space is available. If space is not available at the sending socket to hold the message to be  
 39910 transmitted and the socket file descriptor does have **O\_NONBLOCK** set, the *sendmsg()* function  
 39911 shall fail.

39912 If the socket protocol supports broadcast and the specified address is a broadcast address for the  
 39913 socket protocol, *sendmsg()* shall fail if the **SO\_BROADCAST** option is not set for the socket.

39914 The socket in use may require the process to have appropriate privileges to use the *sendmsg()*  
 39915 function.

39916 **RETURN VALUE**

39917 Upon successful completion, *sendmsg()* shall return the number of bytes sent. Otherwise,  $-1$   
 39918 shall be returned and *errno* set to indicate the error.

39919 **ERRORS**39920 The *sendmsg()* function shall fail if:

39921 [EAGAIN] or [EWOULDBLOCK]

39922 The socket's file descriptor is marked **O\_NONBLOCK** and the requested

|       |                |                                                                                                                                                                                                                                |
|-------|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 39923 |                | operation would block.                                                                                                                                                                                                         |
| 39924 | [EAFNOSUPPORT] |                                                                                                                                                                                                                                |
| 39925 |                | Addresses in the specified address family cannot be used with this socket.                                                                                                                                                     |
| 39926 | [EBADF]        | The <i>socket</i> argument is not a valid file descriptor.                                                                                                                                                                     |
| 39927 | [ECONNRESET]   | A connection was forcibly closed by a peer.                                                                                                                                                                                    |
| 39928 | [EINTR]        | A signal interrupted <i>sendmsg()</i> before any data was transmitted.                                                                                                                                                         |
| 39929 | [EINVAL]       | The sum of the <i>iov_len</i> values overflows an <i>ssize_t</i> .                                                                                                                                                             |
| 39930 | [EMSGSIZE]     | The message is too large to be sent all at once (as the socket requires), or the <i>msg_iovlen</i> member of the <i>msghdr</i> structure pointed to by <i>message</i> is less than or equal to 0 or is greater than {IOV_MAX}. |
| 39931 |                |                                                                                                                                                                                                                                |
| 39932 |                |                                                                                                                                                                                                                                |
| 39933 | [ENOTCONN]     | The socket is connection-mode but is not connected.                                                                                                                                                                            |
| 39934 | [ENOTSOCK]     | The <i>socket</i> argument does not refer to a socket.                                                                                                                                                                         |
| 39935 | [EOPNOTSUPP]   | The <i>socket</i> argument is associated with a socket that does not support one or more of the values set in <i>flags</i> .                                                                                                   |
| 39936 |                |                                                                                                                                                                                                                                |
| 39937 | [EPIPE]        | The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type SOCK_STREAM, the SIGPIPE signal is generated to the calling thread.         |
| 39938 |                |                                                                                                                                                                                                                                |
| 39939 |                |                                                                                                                                                                                                                                |
| 39940 |                | If the address family of the socket is AF_UNIX, then <i>sendmsg()</i> shall fail if:                                                                                                                                           |
| 39941 | [EIO]          | An I/O error occurred while reading from or writing to the file system.                                                                                                                                                        |
| 39942 | [ELOOP]        | A loop exists in symbolic links encountered during resolution of the pathname in the socket address.                                                                                                                           |
| 39943 |                |                                                                                                                                                                                                                                |
| 39944 | [ENAMETOOLONG] |                                                                                                                                                                                                                                |
| 39945 |                | A component of a pathname exceeded {NAME_MAX} characters, or an entire pathname exceeded {PATH_MAX} characters.                                                                                                                |
| 39946 |                |                                                                                                                                                                                                                                |
| 39947 | [ENOENT]       | A component of the pathname does not name an existing file or the path name is an empty string.                                                                                                                                |
| 39948 |                |                                                                                                                                                                                                                                |
| 39949 | [ENOTDIR]      | A component of the path prefix of the pathname in the socket address is not a directory.                                                                                                                                       |
| 39950 |                |                                                                                                                                                                                                                                |
| 39951 |                | The <i>sendmsg()</i> function may fail if:                                                                                                                                                                                     |
| 39952 | [EACCES]       | Search permission is denied for a component of the path prefix; or write access to the named socket is denied.                                                                                                                 |
| 39953 |                |                                                                                                                                                                                                                                |
| 39954 | [EDESTADDRREQ] |                                                                                                                                                                                                                                |
| 39955 |                | The socket is not connection-mode and does not have its peer address set, and no destination address was specified.                                                                                                            |
| 39956 |                |                                                                                                                                                                                                                                |
| 39957 | [EHOSTUNREACH] |                                                                                                                                                                                                                                |
| 39958 |                | The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).                                                                                                                 |
| 39959 |                |                                                                                                                                                                                                                                |
| 39960 | [EIO]          | An I/O error occurred while reading from or writing to the file system.                                                                                                                                                        |
| 39961 | [EISCONN]      | A destination address was specified and the socket is already connected.                                                                                                                                                       |
| 39962 | [ENETDOWN]     | The local network interface used to reach the destination is down.                                                                                                                                                             |

- 39963 [ENETUNREACH]  
39964 No route to the network is present.
- 39965 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 39966 [ENOMEM] Insufficient memory was available to fulfill the request.
- 39967 If the address family of the socket is AF\_UNIX, then *sendmsg()* may fail if:
- 39968 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
39969 resolution of the pathname in the socket address.
- 39970 [ENAMETOOLONG]  
39971 Pathname resolution of a symbolic link produced an intermediate result  
39972 whose length exceeds {PATH\_MAX}.
- 39973 **EXAMPLES**  
39974 Done.
- 39975 **APPLICATION USAGE**  
39976 The *select()* and *poll()* functions can be used to determine when it is possible to send more data.
- 39977 **RATIONALE**  
39978 None.
- 39979 **FUTURE DIRECTIONS**  
39980 None.
- 39981 **SEE ALSO**  
39982 *getsockopt()*, *poll()*, *recv()*, *recvfrom()*, *recvmsg()*, *select()*, *send()*, *sendto()*, *setsockopt()*,  
39983 *shutdown()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/socket.h>
- 39984 **CHANGE HISTORY**  
39985 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.  
39986 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
39987 [ELOOP] error condition is added.

## 39988 NAME

39989 sendto — send a message on a socket

## 39990 SYNOPSIS

39991 #include &lt;sys/socket.h&gt;

39992 ssize\_t sendto(int *socket*, const void \**message*, size\_t *length*,39993 int *flags*, const struct sockaddr \**dest\_addr*,39994 socklen\_t *dest\_len*);

## 39995 DESCRIPTION

39996 The *sendto()* function shall send a message through a connection-mode or connectionless-mode  
 39997 socket. If the socket is connectionless-mode, the message shall be sent to the address specified by  
 39998 *dest\_addr*. If the socket is connection-mode, *dest\_addr* shall be ignored.

39999 The *sendto()* function takes the following arguments:40000 *socket* Specifies the socket file descriptor.40001 *message* Points to a buffer containing the message to be sent.40002 *length* Specifies the size of the message in bytes.

40003 *flags* Specifies the type of message transmission. Values of this argument are  
 40004 formed by logically OR'ing zero or more of the following flags:

40005 MSG\_EOR Terminates a record (if supported by the protocol).

40006 MSG\_OOB Sends out-of-band data on sockets that support out-of-band  
 40007 data. The significance and semantics of out-of-band data are  
 40008 protocol-specific.

40009 *dest\_addr* Points to a **sockaddr** structure containing the destination address. The length  
 40010 and format of the address depend on the address family of the socket.

40011 *dest\_len* Specifies the length of the **sockaddr** structure pointed to by the *dest\_addr*  
 40012 argument.

40013 If the socket protocol supports broadcast and the specified address is a broadcast address for the  
 40014 socket protocol, *sendto()* shall fail if the SO\_BROADCAST option is not set for the socket.

40015 The *dest\_addr* argument specifies the address of the target. The *length* argument specifies the  
 40016 length of the message.

40017 Successful completion of a call to *sendto()* does not guarantee delivery of the message. A return  
 40018 value of  $-1$  indicates only locally-detected errors.

40019 If space is not available at the sending socket to hold the message to be transmitted and the  
 40020 socket file descriptor does not have O\_NONBLOCK set, *sendto()* shall block until space is  
 40021 available. If space is not available at the sending socket to hold the message to be transmitted  
 40022 and the socket file descriptor does have O\_NONBLOCK set, *sendto()* shall fail.

40023 The socket in use may require the process to have appropriate privileges to use the *sendto()*  
 40024 function.

## 40025 RETURN VALUE

40026 Upon successful completion, *sendto()* shall return the number of bytes sent. Otherwise,  $-1$  shall  
 40027 be returned and *errno* set to indicate the error.

40028 **ERRORS**

- 40029 The *sendto()* function shall fail if:
- 40030 [EAFNOSUPPORT]  
40031 Addresses in the specified address family cannot be used with this socket.
- 40032 [EAGAIN] or [EWOULDBLOCK]  
40033 The socket's file descriptor is marked O\_NONBLOCK and the requested  
40034 operation would block.
- 40035 [EBADF] The *socket* argument is not a valid file descriptor.
- 40036 [ECONNRESET] A connection was forcibly closed by a peer.
- 40037 [EINTR] A signal interrupted *sendto()* before any data was transmitted.
- 40038 [EMSGSIZE] The message is too large to be sent all at once, as the socket requires.
- 40039 [ENOTCONN] The socket is connection-mode but is not connected.
- 40040 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 40041 [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or  
40042 more of the values set in *flags*.
- 40043 [EPIPE] The socket is shut down for writing, or the socket is connection-mode and is  
40044 no longer connected. In the latter case, and if the socket is of type  
40045 SOCK\_STREAM, the SIGPIPE signal is generated to the calling thread.
- 40046 If the address family of the socket is AF\_UNIX, then *sendto()* shall fail if:
- 40047 [EIO] An I/O error occurred while reading from or writing to the file system.
- 40048 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname  
40049 in the socket address.
- 40050 [ENAMETOOLONG]  
40051 A component of a pathname exceeded {NAME\_MAX} characters, or an entire  
40052 pathname exceeded {PATH\_MAX} characters.
- 40053 [ENOENT] A component of the pathname does not name an existing file or the pathname  
40054 is an empty string.
- 40055 [ENOTDIR] A component of the path prefix of the pathname in the socket address is not a  
40056 directory.
- 40057 The *sendto()* function may fail if:
- 40058 [EACCES] Search permission is denied for a component of the path prefix; or write  
40059 access to the named socket is denied.
- 40060 [EDESTADDRREQ]  
40061 The socket is not connection-mode and does not have its peer address set, and  
40062 no destination address was specified.
- 40063 [EHOSTUNREACH]  
40064 The destination host cannot be reached (probably because the host is down or  
40065 a remote router cannot reach it).
- 40066 [EINVAL] The *dest\_len* argument is not a valid length for the address family.
- 40067 [EIO] An I/O error occurred while reading from or writing to the file system.

- 40068 [EISCONN] A destination address was specified and the socket is already connected. This  
40069 error may or may not be returned for connection mode sockets.
- 40070 [ENETDOWN] The local network interface used to reach the destination is down.
- 40071 [ENETUNREACH]  
40072 No route to the network is present.
- 40073 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 40074 [ENOMEM] Insufficient memory was available to fulfill the request.
- 40075 If the address family of the socket is AF\_UNIX, then *sendto()* may fail if:
- 40076 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
40077 resolution of the pathname in the socket address.
- 40078 [ENAMETOOLONG]  
40079 Pathname resolution of a symbolic link produced an intermediate result  
40080 whose length exceeds {PATH\_MAX}.
- 40081 **EXAMPLES**  
40082 None.
- 40083 **APPLICATION USAGE**  
40084 The *select()* and *poll()* functions can be used to determine when it is possible to send more data.
- 40085 **RATIONALE**  
40086 None.
- 40087 **FUTURE DIRECTIONS**  
40088 None.
- 40089 **SEE ALSO**  
40090 *getsockopt()*, *poll()*, *recv()*, *recvfrom()*, *recvmsg()*, *select()*, *send()*, *sendmsg()*, *setsockopt()*,  
40091 *shutdown()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/socket.h>
- 40092 **CHANGE HISTORY**  
40093 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.
- 40094 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
40095 [ELOOP] error condition is added.

40096 **NAME**

40097        setbuf — assign buffering to a stream

40098 **SYNOPSIS**

40099        #include &lt;stdio.h&gt;

40100        void setbuf(FILE \*restrict stream, char \*restrict buf);

40101 **DESCRIPTION**

40102 cx        The functionality described on this reference page is aligned with the ISO C standard. Any  
40103        conflict between the requirements described here and the ISO C standard is unintentional. This  
40104        volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

40105        Except that it returns no value, the function call:

40106        setbuf(stream, buf)

40107        shall be equivalent to:

40108        setvbuf(stream, buf, \_IOFBF, BUFSIZ)

40109        if *buf* is not a null pointer, or to:

40110        setvbuf(stream, buf, \_IONBF, BUFSIZ)

40111        if *buf* is a null pointer.40112 **RETURN VALUE**40113        The *setbuf()* function shall not return a value.40114 **ERRORS**

40115        No errors are defined.

40116 **EXAMPLES**

40117        None.

40118 **APPLICATION USAGE**

40119        A common source of error is allocating buffer space as an “automatic” variable in a code block,  
40120        and then failing to close the stream in the same block.

40121        With *setbuf()*, allocating a buffer of BUFSIZ bytes does not necessarily imply that all of BUFSIZ  
40122        bytes are used for the buffer area.

40123 **RATIONALE**

40124        None.

40125 **FUTURE DIRECTIONS**

40126        None.

40127 **SEE ALSO**40128        *fopen()*, *setvbuf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>40129 **CHANGE HISTORY**

40130        First released in Issue 1. Derived from Issue 1 of the SVID.

40131 **Issue 6**40132        The prototype for *setbuf()* is updated for alignment with the ISO/IEC 9899:1999 standard.

40133 **NAME**

40134           setcontext — set current user context

40135 **SYNOPSIS**

40136 xSI       #include <ucontext.h>

40137           int setcontext(const ucontext\_t \*ucp);

40138

40139 **DESCRIPTION**

40140           Refer to *getcontext()*.

40141 **NAME**

40142 setegid — set the effective group ID

40143 **SYNOPSIS**

40144 #include &lt;unistd.h&gt;

40145 int setegid(gid\_t gid);

40146 **DESCRIPTION**

40147 If *gid* is equal to the real group ID or the saved set-group-ID, or if the process has appropriate  
40148 privileges, *setegid()* shall set the effective group ID of the calling process to *gid*; the real group  
40149 ID, saved set-group-ID, and any supplementary group IDs shall remain unchanged.

40150 The *setegid()* function shall not affect the supplementary group list in any way.40151 **RETURN VALUE**

40152 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
40153 indicate the error.

40154 **ERRORS**40155 The *setegid()* function shall fail if:

40156 [EINVAL] The value of the *gid* argument is invalid and is not supported by the  
40157 implementation.

40158 [EPERM] The process does not have appropriate privileges and *gid* does not match the  
40159 real group ID or the saved set-group-ID.

40160 **EXAMPLES**

40161 None.

40162 **APPLICATION USAGE**

40163 None.

40164 **RATIONALE**40165 Refer to the RATIONALE section in *setuid()*.40166 **FUTURE DIRECTIONS**

40167 None.

40168 **SEE ALSO**

40169 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the  
40170 Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>

40171 **CHANGE HISTORY**

40172 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

40173 **NAME**

40174 setenv — add or change environment variable

40175 **SYNOPSIS**40176 cx 

```
#include <stdlib.h>
```

40177 

```
int setenv(const char *envname, const char *envval, int overwrite);
```

40178

40179 **DESCRIPTION**

40180 The *setenv()* function shall update or add a variable in the environment of the calling process.  
 40181 The *envname* argument points to a string containing the name of an environment variable to be  
 40182 added or altered. The environment variable shall be set to the value to which *envval* points. The  
 40183 function shall fail if *envname* points to a string which contains an '=' character. If the  
 40184 environment variable named by *envname* already exists and the value of *overwrite* is non-zero,  
 40185 the function shall return success and the environment shall be updated. If the environment  
 40186 variable named by *envname* already exists and the value of *overwrite* is zero, the function shall  
 40187 return success and the environment shall remain unchanged.

40188 If the application modifies *environ* or the pointers to which it points, the behavior of *setenv()* is  
 40189 undefined. The *setenv()* function shall update the list of pointers to which *environ* points.

40190 The strings described by *envname* and *envval* are copied by this function.

40191 The *setenv()* function need not be reentrant. A function that is not required to be reentrant is not  
 40192 required to be thread-safe.

40193 **RETURN VALUE**

40194 Upon successful completion, zero shall be returned. Otherwise, -1 shall be returned, *errno* set to  
 40195 indicate the error, and the environment shall be unchanged.

40196 **ERRORS**

40197 The *setenv()* function shall fail if:

40198 [EINVAL] The *name* argument is a null pointer, points to an empty string, or points to a  
 40199 string containing an '=' character.

40200 [ENOMEM] Insufficient memory was available to add a variable or its value to the  
 40201 environment.

40202 **EXAMPLES**

40203 None.

40204 **APPLICATION USAGE**

40205 See *exec*, for restrictions on changing the environment in multi-threaded applications.

40206 **RATIONALE**

40207 Unanticipated results may occur if *setenv()* changes the external variable *environ*. In particular,  
 40208 if the optional *envp* argument to *main()* is present, it is not changed, and thus may point to an  
 40209 obsolete copy of the environment (as may any other copy of *environ*). However, other than the  
 40210 aforementioned restriction, the developers of IEEE Std 1003.1-2001 intended that the traditional  
 40211 method of walking through the environment by way of the *environ* pointer must be supported.

40212 It was decided that *setenv()* should be required by this revision because it addresses a piece of  
 40213 missing functionality, and does not impose a significant burden on the implementor.

40214 There was considerable debate as to whether the System V *putenv()* function or the BSD *setenv()*  
 40215 function should be required as a mandatory function. The *setenv()* function was chosen because  
 40216 it permitted the implementation of the *unsetenv()* function to delete environmental variables,  
 40217 without specifying an additional interface. The *putenv()* function is available as an XSI

- 40218 extension.
- 40219 The standard developers considered requiring that *setenv()* indicate an error when a call to it  
40220 would result in exceeding {ARG\_MAX}. The requirement was rejected since the condition might  
40221 be temporary, with the application eventually reducing the environment size. The ultimate  
40222 success or failure depends on the size at the time of a call to *exec*, which returns an indication of  
40223 this error condition.
- 40224 **FUTURE DIRECTIONS**
- 40225 None.
- 40226 **SEE ALSO**
- 40227 *exec*, *getenv()*, *unsetenv()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>, |  
40228 <sys/types.h>, <unistd.h>
- 40229 **CHANGE HISTORY**
- 40230 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.
- 40231 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/55 is applied, adding references to *exec* in |  
40232 the APPLICATION USAGE and SEE ALSO sections. |

40233 **NAME**

40234           seteuid — set effective user ID

40235 **SYNOPSIS**

40236           #include &lt;unistd.h&gt;

40237           int seteuid(uid\_t uid);

40238 **DESCRIPTION**

40239           If *uid* is equal to the real user ID or the saved set-user-ID, or if the process has appropriate  
40240           privileges, *seteuid()* shall set the effective user ID of the calling process to *uid*; the real user ID  
40241           and saved set-user-ID shall remain unchanged.

40242           The *seteuid()* function shall not affect the supplementary group list in any way.40243 **RETURN VALUE**

40244           Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
40245           indicate the error.

40246 **ERRORS**40247           The *seteuid()* function shall fail if:

40248           [EINVAL]           The value of the *uid* argument is invalid and is not supported by the  
40249           implementation.

40250           [EPERM]           The process does not have appropriate privileges and *uid* does not match the  
40251           real group ID or the saved set-group-ID.

40252 **EXAMPLES**

40253           None.

40254 **APPLICATION USAGE**

40255           None.

40256 **RATIONALE**40257           Refer to the RATIONALE section in *setuid()*.40258 **FUTURE DIRECTIONS**

40259           None.

40260 **SEE ALSO**

40261           *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the  
40262           Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>

40263 **CHANGE HISTORY**

40264           First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

40265 **NAME**

40266           setgid — set-group-ID

40267 **SYNOPSIS**

40268           #include &lt;unistd.h&gt;

40269           int setgid(gid\_t *gid*);40270 **DESCRIPTION**40271           If the process has appropriate privileges, *setgid()* shall set the real group ID, effective group ID, and the saved set-group-ID of the calling process to *gid*.40273           If the process does not have appropriate privileges, but *gid* is equal to the real group ID or the saved set-group-ID, *setgid()* shall set the effective group ID to *gid*; the real group ID and saved set-group-ID shall remain unchanged.40276           The *setgid()* function shall not affect the supplementary group list in any way.

40277           Any supplementary group IDs of the calling process shall remain unchanged.

40278 **RETURN VALUE**40279           Upon successful completion, 0 is returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.40281 **ERRORS**40282           The *setgid()* function shall fail if:40283           [EINVAL]           The value of the *gid* argument is invalid and is not supported by the  
40284           implementation.40285           [EPERM]           The process does not have appropriate privileges and *gid* does not match the  
40286           real group ID or the saved set-group-ID.40287 **EXAMPLES**

40288           None.

40289 **APPLICATION USAGE**

40290           None.

40291 **RATIONALE**40292           Refer to the RATIONALE section in *setuid()*.40293 **FUTURE DIRECTIONS**

40294           None.

40295 **SEE ALSO**40296           *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setregid()*, *setreuid()*, *setuid()*, the  
40297           Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>40298 **CHANGE HISTORY**

40299           First released in Issue 1. Derived from Issue 1 of the SVID.

40300 **Issue 6**

40301           In the SYNOPSIS, the optional include of the &lt;sys/types.h&gt; header is removed.

40302           The following new requirements on POSIX implementations derive from alignment with the  
40303           Single UNIX Specification:

- 40304           • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
- 
- 40305           required for conforming implementations of previous POSIX specifications, it was not
- 
- 40306           required for UNIX applications.

40307       • Functionality associated with `_POSIX_SAVED_IDS` is now mandated. This is a FIPS  
40308       requirement.

40309       The following changes were made to align with the IEEE P1003.1a draft standard:

- 40310       • The effects of `setgid()` in processes without appropriate privileges are changed.
- 40311       • A requirement that the supplementary group list is not affected is added.

40312 **NAME**

40313           setgrent — reset the group database to the first entry

40314 **SYNOPSIS**

```
40315 xSI #include <grp.h>
```

```
40316 void setgrent(void);
```

40317

40318 **DESCRIPTION**

40319           Refer to *endgrent()*.

40320 **NAME**

40321           sethostent — network host database functions

40322 **SYNOPSIS**

40323           #include <netdb.h>

40324           void sethostent(int *stayopen*);

40325 **DESCRIPTION**

40326           Refer to *endhostent()*.

40327 **NAME**

40328       setitimer — set the value of an interval timer

40329 **SYNOPSIS**

40330 xSI       #include &lt;sys/time.h&gt;

40331       int setitimer(int *which*, const struct itimerval \*restrict *value*,  
40332                    struct itimerval \*restrict *ovalue*);

40333

40334 **DESCRIPTION**40335       Refer to *getitimer()*.

40336 **NAME**

40337 setjmp — set jump point for a non-local goto

40338 **SYNOPSIS**

40339 #include &lt;setjmp.h&gt;

40340 int setjmp(jmp\_buf env);

40341 **DESCRIPTION**

40342 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
40343 conflict between the requirements described here and the ISO C standard is unintentional. This  
40344 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

40345 A call to *setjmp()* shall save the calling environment in its *env* argument for later use by  
40346 *longjmp()*.

40347 It is unspecified whether *setjmp()* is a macro or a function. If a macro definition is suppressed in  
40348 order to access an actual function, or a program defines an external identifier with the name  
40349 *setjmp*, the behavior is undefined.

40350 An application shall ensure that an invocation of *setjmp()* appears in one of the following  
40351 contexts only:

- 40352 • The entire controlling expression of a selection or iteration statement
- 40353 • One operand of a relational or equality operator with the other operand an integral constant  
40354 expression, with the resulting expression being the entire controlling expression of a  
40355 selection or iteration statement
- 40356 • The operand of a unary '!' operator with the resulting expression being the entire  
40357 controlling expression of a selection or iteration
- 40358 • The entire expression of an expression statement (possibly cast to **void**)

40359 If the invocation appears in any other context, the behavior is undefined.

40360 **RETURN VALUE**

40361 If the return is from a direct invocation, *setjmp()* shall return 0. If the return is from a call to  
40362 *longjmp()*, *setjmp()* shall return a non-zero value.

40363 **ERRORS**

40364 No errors are defined.

40365 **EXAMPLES**

40366 None.

40367 **APPLICATION USAGE**

40368 In general, *sigsetjmp()* is more useful in dealing with errors and interrupts encountered in a low-  
40369 level subroutine of a program.

40370 **RATIONALE**

40371 None.

40372 **FUTURE DIRECTIONS**

40373 None.

40374 **SEE ALSO**

40375 *longjmp()*, *sigsetjmp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**setjmp.h**>

40376 **CHANGE HISTORY**

40377 First released in Issue 1. Derived from Issue 1 of the SVID.

40378 **Issue 6**

40379 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

40380 **NAME**40381 setkey — set encoding key (**CRYPT**)40382 **SYNOPSIS**40383 XSI `#include <stdlib.h>`40384 `void setkey(const char *key);`

40385

40386 **DESCRIPTION**

40387 The *setkey()* function provides access to an implementation-defined encoding algorithm. The  
40388 argument of *setkey()* is an array of length 64 bytes containing only the bytes with numerical  
40389 value of 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is  
40390 ignored; this gives a 56-bit key which is used by the algorithm. This is the key that shall be used  
40391 with the algorithm to encode a string *block* passed to *encrypt()*.

40392 The *setkey()* function shall not change the setting of *errno* if successful. An application wishing to  
40393 check for error situations should set *errno* to 0 before calling *setkey()*. If *errno* is non-zero on  
40394 return, an error has occurred.

40395 The *setkey()* function need not be reentrant. A function that is not required to be reentrant is not  
40396 required to be thread-safe.

40397 **RETURN VALUE**

40398 No values are returned.

40399 **ERRORS**40400 The *setkey()* function shall fail if:

40401 [ENOSYS] The functionality is not supported on this implementation.

40402 **EXAMPLES**

40403 None.

40404 **APPLICATION USAGE**

40405 Decoding need not be implemented in all environments. This is related to government  
40406 restrictions in some countries on encryption and decryption routines. Historical practice has  
40407 been to ship a different version of the encryption library without the decryption feature in the  
40408 routines supplied. Thus the exported version of *encrypt()* does encoding but not decoding.

40409 **RATIONALE**

40410 None.

40411 **FUTURE DIRECTIONS**

40412 None.

40413 **SEE ALSO**40414 *crypt()*, *encrypt()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<stdlib.h>`40415 **CHANGE HISTORY**

40416 First released in Issue 1. Derived from Issue 1 of the SVID.

40417 **Issue 5**40418 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

40419 **NAME**

40420 setlocale — set program locale

40421 **SYNOPSIS**

40422 #include &lt;locale.h&gt;

40423 char \*setlocale(int *category*, const char \**locale*);40424 **DESCRIPTION**

40425 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 40426 conflict between the requirements described here and the ISO C standard is unintentional. This  
 40427 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

40428 The *setlocale()* function selects the appropriate piece of the program's locale, as specified by the  
 40429 *category* and *locale* arguments, and may be used to change or query the program's entire locale or  
 40430 portions thereof. The value *LC\_ALL* for *category* names the program's entire locale; other values  
 40431 for *category* name only a part of the program's locale:

40432 *LC\_COLLATE* Affects the behavior of regular expressions and the collation functions.

40433 *LC\_CTYPE* Affects the behavior of regular expressions, character classification, character  
 40434 conversion functions, and wide-character functions.

40435 CX *LC\_MESSAGES* Affects what strings are expected by commands and utilities as affirmative or  
 40436 negative responses.

40437 XSI It also affects what strings are given by commands and utilities as affirmative  
 40438 or negative responses, and the content of messages.

40439 *LC\_MONETARY* Affects the behavior of functions that handle monetary values.

40440 *LC\_NUMERIC* Affects the behavior of functions that handle numeric values.

40441 *LC\_TIME* Affects the behavior of the time conversion functions.

40442 The *locale* argument is a pointer to a character string containing the required setting of *category*.  
 40443 The contents of this string are implementation-defined. In addition, the following preset values  
 40444 of *locale* are defined for all settings of *category*:

40445 CX "POSIX" Specifies the minimal environment for C-language translation called the  
 40446 POSIX locale. If *setlocale()* is not invoked, the POSIX locale is the default at  
 40447 entry to *main()*.

40448 "C" Equivalent to "POSIX".

40449 CX "" Specifies an implementation-defined native environment. This corresponds to  
 40450 the value of the associated environment variables, *LC\_\** and *LANG*; see the  
 40451 Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale and the  
 40452 Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment  
 40453 Variables.

40454 A null pointer Used to direct *setlocale()* to query the current internationalized environment  
 40455 and return the name of the locale.

40456 THR The locale state is common to all threads within a process.

40457 **RETURN VALUE**

40458 Upon successful completion, *setlocale()* shall return the string associated with the specified  
 40459 category for the new locale. Otherwise, *setlocale()* shall return a null pointer and the program's  
 40460 locale is not changed.

40461 A null pointer for *locale* causes *setlocale()* to return a pointer to the string associated with the  
 40462 *category* for the program's current locale. The program's locale shall not be changed.

40463 The string returned by *setlocale()* is such that a subsequent call with that string and its associated  
 40464 *category* shall restore that part of the program's locale. The application shall not modify the string  
 40465 returned which may be overwritten by a subsequent call to *setlocale()*.

#### 40466 ERRORS

40467 No errors are defined.

#### 40468 EXAMPLES

40469 None.

#### 40470 APPLICATION USAGE

40471 The following code illustrates how a program can initialize the international environment for  
 40472 one language, while selectively modifying the program's locale such that regular expressions  
 40473 and string operations can be applied to text recorded in a different language:

```
40474 setlocale(LC_ALL, "De");
40475 setlocale(LC_COLLATE, "Fr@dict");
```

40476 Internationalized programs must call *setlocale()* to initiate a specific language operation. This can  
 40477 be done by calling *setlocale()* as follows:

```
40478 setlocale(LC_ALL, "");
```

40479 Changing the setting of *LC\_MESSAGES* has no effect on catalogs that have already been opened  
 40480 by calls to *catopen()*.

#### 40481 RATIONALE

40482 The ISO C standard defines a collection of functions to support internationalization. One of the  
 40483 most significant aspects of these functions is a facility to set and query the *international*  
 40484 *environment*. The international environment is a repository of information that affects the  
 40485 behavior of certain functionality, namely:

- 40486 1. Character handling
- 40487 2. Collating
- 40488 3. Date/time formatting
- 40489 4. Numeric editing
- 40490 5. Monetary formatting
- 40491 6. Messaging

40492 The *setlocale()* function provides the application developer with the ability to set all or portions,  
 40493 called *categories*, of the international environment. These categories correspond to the areas of  
 40494 functionality mentioned above. The syntax for *setlocale()* is as follows:

```
40495 char *setlocale(int category, const char *locale);
```

40496 where *category* is the name of one of following categories, namely:

```
40497 LC_COLLATE
40498 LC_CTYPE
40499 LC_MESSAGES
40500 LC_MONETARY
40501 LC_NUMERIC
40502 LC_TIME
```

40503 In addition, a special value called *LC\_ALL* directs *setlocale()* to set all categories.

40504 There are two primary uses of *setlocale()*:

- 40505 1. Querying the international environment to find out what it is set to
- 40506 2. Setting the international environment, or *locale*, to a specific value

40507 The behavior of *setlocale()* in these two areas is described below. Since it is difficult to describe  
40508 the behavior in words, examples are used to illustrate the behavior of specific uses.

40509 To query the international environment, *setlocale()* is invoked with a specific category and the  
40510 NULL pointer as the locale. The NULL pointer is a special directive to *setlocale()* that tells it to  
40511 query rather than set the international environment. The following syntax is used to query the  
40512 name of the international environment:

```
40513 setlocale({LC_ALL, LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, \
40514 LC_NUMERIC, LC_TIME}, (char *) NULL);
```

40515 The *setlocale()* function shall return the string corresponding to the current international  
40516 environment. This value may be used by a subsequent call to *setlocale()* to reset the international  
40517 environment to this value. However, it should be noted that the return value from *setlocale()*  
40518 may be a pointer to a static area within the function and is not guaranteed to remain unchanged  
40519 (that is, it may be modified by a subsequent call to *setlocale()*). Therefore, if the purpose of  
40520 calling *setlocale()* is to save the value of the current international environment so it can be  
40521 changed and reset later, the return value should be copied to an array of **char** in the calling  
40522 program.

40523 There are three ways to set the international environment with *setlocale()*:

40524 *setlocale(category, string)*

40525 This usage sets a specific *category* in the international environment to a specific value  
40526 corresponding to the value of the *string*. A specific example is provided below:

```
40527 setlocale(LC_ALL, "fr_FR.ISO-8859-1");
```

40528 In this example, all categories of the international environment are set to the locale  
40529 corresponding to the string "fr\_FR.ISO-8859-1", or to the French language as spoken in  
40530 France using the ISO/IEC 8859-1:1998 standard codeset.

40531 If the string does not correspond to a valid locale, *setlocale()* shall return a NULL pointer  
40532 and the international environment is not changed. Otherwise, *setlocale()* shall return the  
40533 name of the locale just set.

40534 *setlocale(category, "C")*

40535 The ISO C standard states that one locale must exist on all conforming implementations.  
40536 The name of the locale is C and corresponds to a minimal international environment needed  
40537 to support the C programming language.

40538 *setlocale(category, "")*

40539 This sets a specific category to an implementation-defined default. This corresponds to the  
40540 value of the environment variables.

#### 40541 FUTURE DIRECTIONS

40542 None.

#### 40543 SEE ALSO

40544 *exec*, *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*,  
40545 *isspace()*, *isupper()*, *iswalnum()*, *iswalpha()*, *iswblank()*, *iswcntrl()*, *iswctype()*, *iswdigit()*,  
40546 *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *isxdigit()*,

40547 *localeconv()*, *mblen()*, *mbstowcs()*, *mbtowc()*, *nl\_langinfo()*, *printf()*, *scanf()*, *setlocale()*, *strcoll()*,  
40548 *strerror()*, *strfmon()*, *strtod()*, *strxfrm()*, *tolower()*, *toupper()*, *towlower()*, *towupper()*, *wscoll()*,  
40549 *wcstod()*, *wcstombs()*, *wcsxfrm()*, *wctomb()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
40550 **<langinfo.h>**, **<locale.h>**

40551 **CHANGE HISTORY**

40552 First released in Issue 3.

40553 **Issue 5**

40554 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

40555 **Issue 6**

40556 Extensions beyond the ISO C standard are marked.

40557 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

40558 **NAME**

40559 setlogmask — set the log priority mask

40560 **SYNOPSIS**

40561 xSI #include &lt;syslog.h&gt;

40562 int setlogmask(int maskpri);

40563

40564 **DESCRIPTION**40565 Refer to *closelog()*.

40566 **NAME**

40567           setnetent — network database function

40568 **SYNOPSIS**

40569           #include <netdb.h>

40570           void setnetent(int stayopen);

40571 **DESCRIPTION**

40572           Refer to *endnetent()*.

40573 **NAME**

40574 setpgid — set process group ID for job control

40575 **SYNOPSIS**

40576 #include &lt;unistd.h&gt;

40577 int setpgid(pid\_t pid, pid\_t pgid);

40578 **DESCRIPTION**

40579 The *setpgid()* function shall either join an existing process group or create a new process group  
 40580 within the session of the calling process. The process group ID of a session leader shall not  
 40581 change. Upon successful completion, the process group ID of the process with a process ID that  
 40582 matches *pid* shall be set to *pgid*. As a special case, if *pid* is 0, the process ID of the calling process  
 40583 shall be used. Also, if *pgid* is 0, the process ID of the indicated process shall be used.

40584 **RETURN VALUE**

40585 Upon successful completion, *setpgid()* shall return 0; otherwise, -1 shall be returned and *errno*  
 40586 shall be set to indicate the error.

40587 **ERRORS**40588 The *setpgid()* function shall fail if:

40589 [EACCES] The value of the *pid* argument matches the process ID of a child process of the  
 40590 calling process and the child process has successfully executed one of the *exec*  
 40591 functions.

40592 [EINVAL] The value of the *pgid* argument is less than 0, or is not a value supported by  
 40593 the implementation.

40594 [EPERM] The process indicated by the *pid* argument is a session leader.

40595 [EPERM] The value of the *pid* argument matches the process ID of a child process of the  
 40596 calling process and the child process is not in the same session as the calling  
 40597 process.

40598 [EPERM] The value of the *pgid* argument is valid but does not match the process ID of  
 40599 the process indicated by the *pid* argument and there is no process with a  
 40600 process group ID that matches the value of the *pgid* argument in the same  
 40601 session as the calling process.

40602 [ESRCH] The value of the *pid* argument does not match the process ID of the calling  
 40603 process or of a child process of the calling process.

40604 **EXAMPLES**

40605 None.

40606 **APPLICATION USAGE**

40607 None.

40608 **RATIONALE**

40609 The *setpgid()* function shall group processes together for the purpose of signaling, placement in  
 40610 foreground or background, and other job control actions.

40611 The *setpgid()* function is similar to the *setpgrp()* function of 4.2 BSD, except that 4.2 BSD allowed  
 40612 the specified new process group to assume any value. This presents certain security problems  
 40613 and is more flexible than necessary to support job control.

40614 To provide tighter security, *setpgid()* only allows the calling process to join a process group  
 40615 already in use inside its session or create a new process group whose process group ID was  
 40616 equal to its process ID.

40617 When a job control shell spawns a new job, the processes in the job must be placed into a new  
40618 process group via *setpgid()*. There are two timing constraints involved in this action:

- 40619 1. The new process must be placed in the new process group before the appropriate program  
40620 is launched via one of the *exec* functions.
- 40621 2. The new process must be placed in the new process group before the shell can correctly  
40622 send signals to the new process group.

40623 To address these constraints, the following actions are performed. The new processes call  
40624 *setpgid()* to alter their own process groups after *fork()* but before *exec*. This satisfies the first  
40625 constraint. Under 4.3 BSD, the second constraint is satisfied by the synchronization property of  
40626 *vfork()*; that is, the shell is suspended until the child has completed the *exec*, thus ensuring that  
40627 the child has completed the *setpgid()*. A new version of *fork()* with this same synchronization  
40628 property was considered, but it was decided instead to merely allow the parent shell process to  
40629 adjust the process group of its child processes via *setpgid()*. Both timing constraints are now  
40630 satisfied by having both the parent shell and the child attempt to adjust the process group of the  
40631 child process; it does not matter which succeeds first.

40632 Since it would be confusing to an application to have its process group change after it began  
40633 executing (that is, after *exec*), and because the child process would already have adjusted its  
40634 process group before this, the [EACCES] error was added to disallow this.

40635 One non-obvious use of *setpgid()* is to allow a job control shell to return itself to its original  
40636 process group (the one in effect when the job control shell was executed). A job control shell  
40637 does this before returning control back to its parent when it is terminating or suspending itself as  
40638 a way of restoring its job control “state” back to what its parent would expect. (Note that the  
40639 original process group of the job control shell typically matches the process group of its parent,  
40640 but this is not necessarily always the case.)

#### 40641 FUTURE DIRECTIONS

40642 None.

#### 40643 SEE ALSO

40644 *exec*, *getpgrp()*, *setsid()*, *tcsetpgrp()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
40645 <sys/types.h>, <unistd.h>

#### 40646 CHANGE HISTORY

40647 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

#### 40648 Issue 6

40649 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

40650 The following new requirements on POSIX implementations derive from alignment with the  
40651 Single UNIX Specification:

- 40652 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
40653 required for conforming implementations of previous POSIX specifications, it was not  
40654 required for UNIX applications.
- 40655 • The *setpgid()* function is mandatory since `_POSIX_JOB_CONTROL` is required to be defined  
40656 in this issue. This is a FIPS requirement.

40657 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/56 is applied, changing the wording in  
40658 the DESCRIPTION from “the process group ID of the indicated process shall be used” to “the  
40659 process ID of the indicated process shall be used”. This change reverts the wording to as in the  
40660 ISO POSIX-1: 1996 standard; it appeared to be an unintentional change.

40661 **NAME**

40662           setpgrp — set the process group ID

40663 **SYNOPSIS**

40664 XSI       #include &lt;unistd.h&gt;

40665           pid\_t setpgrp(void);

40666

40667 **DESCRIPTION**

40668           If the calling process is not already a session leader, *setpgrp()* sets the process group ID of the  
40669           calling process to the process ID of the calling process. If *setpgrp()* creates a new session, then  
40670           the new session has no controlling terminal.

40671           The *setpgrp()* function has no effect when the calling process is a session leader.

40672 **RETURN VALUE**40673           Upon completion, *setpgrp()* shall return the process group ID.40674 **ERRORS**

40675           No errors are defined.

40676 **EXAMPLES**

40677           None.

40678 **APPLICATION USAGE**

40679           None.

40680 **RATIONALE**

40681           None.

40682 **FUTURE DIRECTIONS**

40683           None.

40684 **SEE ALSO**

40685           *exec*, *fork()*, *getpid()*, *getsid()*, *kill()*, *setpgid()*, *setsid()*, the Base Definitions volume of  
40686           IEEE Std 1003.1-2001, <unistd.h>

40687 **CHANGE HISTORY**

40688           First released in Issue 4, Version 2.

40689 **Issue 5**

40690           Moved from X/OPEN UNIX extension to BASE.

40691 **NAME**

40692           setpriority — set the nice value

40693 **SYNOPSIS**

40694 XSI       #include <sys/resource.h>

40695           int setpriority(int *which*, id\_t *who*, int *nice*);

40696

40697 **DESCRIPTION**

40698           Refer to *getpriority()*.

40699 **NAME**

40700           setprotoent — network protocol database functions

40701 **SYNOPSIS**

40702           #include <netdb.h>

40703           void setprotoent(int *stayopen*);

40704 **DESCRIPTION**

40705           Refer to *endprotoent()*.

40706 **NAME**

40707           setpwent — user database function

40708 **SYNOPSIS**

40709 xSI       #include <pwd.h>

40710           void setpwent(void);

40711

40712 **DESCRIPTION**

40713           Refer to *endpwent()*.

40714 **NAME**

40715 setregid — set real and effective group IDs

40716 **SYNOPSIS**

40717 XSI #include &lt;unistd.h&gt;

40718 int setregid(gid\_t rgid, gid\_t egid);

40719

40720 **DESCRIPTION**40721 The *setregid()* function shall set the real and effective group IDs of the calling process.40722 If *rgid* is  $-1$ , the real group ID shall not be changed; if *egid* is  $-1$ , the effective group ID shall not  
40723 be changed.

40724 The real and effective group IDs may be set to different values in the same call.

40725 Only a process with appropriate privileges can set the real group ID and the effective group ID  
40726 to any valid value.40727 A non-privileged process can set either the real group ID to the saved set-group-ID from one of  
40728 the *exec* family of functions, or the effective group ID to the saved set-group-ID or the real group  
40729 ID.

40730 Any supplementary group IDs of the calling process remain unchanged.

40731 **RETURN VALUE**40732 Upon successful completion, 0 shall be returned. Otherwise,  $-1$  shall be returned and *errno* set to  
40733 indicate the error, and neither of the group IDs are changed.40734 **ERRORS**40735 The *setregid()* function shall fail if:40736 [EINVAL] The value of the *rgid* or *egid* argument is invalid or out-of-range.40737 [EPERM] The process does not have appropriate privileges and a change other than  
40738 changing the real group ID to the saved set-group-ID, or changing the  
40739 effective group ID to the real group ID or the saved set-group-ID, was  
40740 requested.40741 **EXAMPLES**

40742 None.

40743 **APPLICATION USAGE**40744 If a set-group-ID process sets its effective group ID to its real group ID, it can still set its effective  
40745 group ID back to the saved set-group-ID.40746 **RATIONALE**

40747 None.

40748 **FUTURE DIRECTIONS**

40749 None.

40750 **SEE ALSO**40751 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setreuid()*, *setuid()*, the  
40752 Base Definitions volume of IEEE Std 1003.1-2001, <unistd.h>40753 **CHANGE HISTORY**

40754 First released in Issue 4, Version 2.

40755 **Issue 5**

40756 Moved from X/OPEN UNIX extension to BASE.

40757 The DESCRIPTION is updated to indicate that the saved set-group-ID can be set by any of the  
40758 *exec* family of functions, not just *execve()*.

40759 **NAME**

40760 setreuid — set real and effective user IDs

40761 **SYNOPSIS**

40762 XSI #include &lt;unistd.h&gt;

40763 int setreuid(uid\_t ruid, uid\_t euid);

40764

40765 **DESCRIPTION**

40766 The *setreuid()* function shall set the real and effective user IDs of the current process to the  
 40767 values specified by the *ruid* and *euid* arguments. If *ruid* or *euid* is  $-1$ , the corresponding effective  
 40768 or real user ID of the current process shall be left unchanged.

40769 A process with appropriate privileges can set either ID to any value. An unprivileged process  
 40770 can only set the effective user ID if the *euid* argument is equal to either the real, effective, or  
 40771 saved user ID of the process.

40772 It is unspecified whether a process without appropriate privileges is permitted to change the real  
 40773 user ID to match the current real, effective, or saved set-user-ID of the process.

40774 **RETURN VALUE**

40775 Upon successful completion, 0 shall be returned. Otherwise,  $-1$  shall be returned and *errno* set to  
 40776 indicate the error.

40777 **ERRORS**40778 The *setreuid()* function shall fail if:

40779 [EINVAL] The value of the *ruid* or *euid* argument is invalid or out-of-range.

40780 [EPERM] The current process does not have appropriate privileges, and either an  
 40781 attempt was made to change the effective user ID to a value other than the  
 40782 real user ID or the saved set-user-ID or an attempt was made to change the  
 40783 real user ID to a value not permitted by the implementation.

40784 **EXAMPLES**40785 **Setting the Effective User ID to the Real User ID**

40786 The following example sets the effective user ID of the calling process to the real user ID, so that  
 40787 files created later will be owned by the current user.

```
40788 #include <unistd.h>
40789 #include <sys/types.h>
40790 ...
40791 setreuid(getuid(), getuid());
40792 ...
```

40793 **APPLICATION USAGE**

40794 None.

40795 **RATIONALE**

40796 None.

40797 **FUTURE DIRECTIONS**

40798 None.

40799 **SEE ALSO**

40800 *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setuid()*, the Base  
40801 Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

40802 **CHANGE HISTORY**

40803 First released in Issue 4, Version 2.

40804 **Issue 5**

40805 Moved from X/OPEN UNIX extension to BASE.

40806 **NAME**

40807           setrlimit — control maximum resource consumption

40808 **SYNOPSIS**

40809 xSI       #include &lt;sys/resource.h&gt;

40810           int setrlimit(int resource, const struct rlimit \*rlp);

40811

40812 **DESCRIPTION**40813           Refer to *getrlimit()*.

40814 **NAME**

40815           setservent — network services database functions

40816 **SYNOPSIS**

40817           #include <netdb.h>

40818           void setservent(int *stayopen*);

40819 **DESCRIPTION**

40820           Refer to *endservent()*.

40821 **NAME**

40822 setsid — create session and set process group ID

40823 **SYNOPSIS**

40824 #include &lt;unistd.h&gt;

40825 pid\_t setsid(void);

40826 **DESCRIPTION**

40827 The *setsid()* function shall create a new session, if the calling process is not a process group  
40828 leader. Upon return the calling process shall be the session leader of this new session, shall be  
40829 the process group leader of a new process group, and shall have no controlling terminal. The  
40830 process group ID of the calling process shall be set equal to the process ID of the calling process.  
40831 The calling process shall be the only process in the new process group and the only process in  
40832 the new session.

40833 **RETURN VALUE**

40834 Upon successful completion, *setsid()* shall return the value of the new process group ID of the  
40835 calling process. Otherwise, it shall return (**pid\_t**)-1 and set *errno* to indicate the error.

40836 **ERRORS**40837 The *setsid()* function shall fail if:

40838 [EPERM] The calling process is already a process group leader, or the process group ID  
40839 of a process other than the calling process matches the process ID of the  
40840 calling process.

40841 **EXAMPLES**

40842 None.

40843 **APPLICATION USAGE**

40844 None.

40845 **RATIONALE**

40846 The *setsid()* function is similar to the *setpgrp()* function of System V. System V, without job  
40847 control, groups processes into process groups and creates new process groups via *setpgrp()*; only  
40848 one process group may be part of a login session.

40849 Job control allows multiple process groups within a login session. In order to limit job control  
40850 actions so that they can only affect processes in the same login session, this volume of  
40851 IEEE Std 1003.1-2001 adds the concept of a session that is created via *setsid()*. The *setsid()*  
40852 function also creates the initial process group contained in the session. Additional process  
40853 groups can be created via the *setpgid()* function. A System V process group would correspond to  
40854 a POSIX System Interfaces session containing a single POSIX process group. Note that this  
40855 function requires that the calling process not be a process group leader. The usual way to ensure  
40856 this is true is to create a new process with *fork()* and have it call *setsid()*. The *fork()* function  
40857 guarantees that the process ID of the new process does not match any existing process group ID.

40858 **FUTURE DIRECTIONS**

40859 None.

40860 **SEE ALSO**

40861 *getsid()*, *setpgid()*, *setpgrp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>,  
40862 <unistd.h>

40863 **CHANGE HISTORY**

40864 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

40865 **Issue 6**

40866 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

40867 The following new requirements on POSIX implementations derive from alignment with the  
40868 Single UNIX Specification:

- 40869 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
40870 required for conforming implementations of previous POSIX specifications, it was not  
40871 required for UNIX applications.

## 40872 NAME

40873 setsockopt — set the socket options

## 40874 SYNOPSIS

40875 #include &lt;sys/socket.h&gt;

```
40876 int setsockopt(int socket, int level, int option_name,
40877 const void *option_value, socklen_t option_len);
```

## 40878 DESCRIPTION

40879 The *setsockopt()* function shall set the option specified by the *option\_name* argument, at the  
 40880 protocol level specified by the *level* argument, to the value pointed to by the *option\_value*  
 40881 argument for the socket associated with the file descriptor specified by the *socket* argument.

40882 The *level* argument specifies the protocol level at which the option resides. To set options at the  
 40883 socket level, specify the *level* argument as SOL\_SOCKET. To set options at other levels, supply  
 40884 the appropriate *level* identifier for the protocol controlling the option. For example, to indicate  
 40885 that an option is interpreted by the TCP (Transport Control Protocol), set *level* to IPPROTO\_TCP  
 40886 as defined in the <netinet/in.h> header.

40887 The *option\_name* argument specifies a single option to set. The *option\_name* argument and any  
 40888 specified options are passed uninterpreted to the appropriate protocol module for  
 40889 interpretations. The <sys/socket.h> header defines the socket-level options. The options are as  
 40890 follows:

40891 SO\_DEBUG Turns on recording of debugging information. This option enables or  
 40892 disables debugging in the underlying protocol modules. This option takes  
 40893 an **int** value. This is a Boolean option.

40894 SO\_BROADCAST Permits sending of broadcast messages, if this is supported by the  
 40895 protocol. This option takes an **int** value. This is a Boolean option.

40896 SO\_REUSEADDR Specifies that the rules used in validating addresses supplied to *bind()*  
 40897 should allow reuse of local addresses, if this is supported by the protocol.  
 40898 This option takes an **int** value. This is a Boolean option.

40899 SO\_KEEPALIVE Keeps connections active by enabling the periodic transmission of  
 40900 messages, if this is supported by the protocol. This option takes an **int**  
 40901 value.

40902 If the connected socket fails to respond to these messages, the connection  
 40903 is broken and threads writing to that socket are notified with a SIGPIPE  
 40904 signal. This is a Boolean option.

40905 SO\_LINGER Lingers on a *close()* if data is present. This option controls the action  
 40906 taken when unsent messages queue on a socket and *close()* is performed.  
 40907 If SO\_LINGER is set, the system shall block the process during *close()*  
 40908 until it can transmit the data or until the time expires. If SO\_LINGER is  
 40909 not specified, and *close()* is issued, the system handles the call in a way  
 40910 that allows the process to continue as quickly as possible. This option  
 40911 takes a **linger** structure, as defined in the <sys/socket.h> header, to  
 40912 specify the state of the option and linger interval.

40913 SO\_OOBINLINE Leaves received out-of-band data (data marked urgent) inline. This  
 40914 option takes an **int** value. This is a Boolean option.

40915 SO\_SNDBUF Sets send buffer size. This option takes an **int** value.

|       |                                                                                                                  |                                                                                      |
|-------|------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| 40916 | SO_RCVBUF                                                                                                        | Sets receive buffer size. This option takes an <b>int</b> value.                     |
| 40917 | SO_DONTROUTE                                                                                                     | Requests that outgoing messages bypass the standard routing facilities.              |
| 40918 |                                                                                                                  | The destination shall be on a directly-connected network, and messages               |
| 40919 |                                                                                                                  | are directed to the appropriate network interface according to the                   |
| 40920 |                                                                                                                  | destination address. The effect, if any, of this option depends on what              |
| 40921 |                                                                                                                  | protocol is in use. This option takes an <b>int</b> value. This is a Boolean option. |
| 40922 | SO_RCVLOWAT                                                                                                      | Sets the minimum number of bytes to process for socket input operations.             |
| 40923 |                                                                                                                  | The default value for SO_RCVLOWAT is 1. If SO_RCVLOWAT is set to a                   |
| 40924 |                                                                                                                  | larger value, blocking receive calls normally wait until they have received          |
| 40925 |                                                                                                                  | the smaller of the low water mark value or the requested amount. (They               |
| 40926 |                                                                                                                  | may return less than the low water mark if an error occurs, a signal is              |
| 40927 |                                                                                                                  | caught, or the type of data next in the receive queue is different from that         |
| 40928 |                                                                                                                  | returned; for example, out-of-band data.) This option takes an <b>int</b> value.     |
| 40929 |                                                                                                                  | Note that not all implementations allow this option to be set.                       |
| 40930 | SO_RCVTIMEO                                                                                                      | Sets the timeout value that specifies the maximum amount of time an                  |
| 40931 |                                                                                                                  | input function waits until it completes. It accepts a <b>timeval</b> structure with  |
| 40932 |                                                                                                                  | the number of seconds and microseconds specifying the limit on how                   |
| 40933 |                                                                                                                  | long to wait for an input operation to complete. If a receive operation has          |
| 40934 |                                                                                                                  | blocked for this much time without receiving additional data, it shall               |
| 40935 |                                                                                                                  | return with a partial count or <i>errno</i> set to [EAGAIN] or                       |
| 40936 |                                                                                                                  | [EWOULDBLOCK] if no data is received. The default for this option is                 |
| 40937 |                                                                                                                  | zero, which indicates that a receive operation shall not time out. This              |
| 40938 |                                                                                                                  | option takes a <b>timeval</b> structure. Note that not all implementations allow     |
| 40939 |                                                                                                                  | this option to be set.                                                               |
| 40940 | SO_SNDLOWAT                                                                                                      | Sets the minimum number of bytes to process for socket output                        |
| 40941 |                                                                                                                  | operations. Non-blocking output operations shall process no data if flow             |
| 40942 |                                                                                                                  | control does not allow the smaller of the send low water mark value or               |
| 40943 |                                                                                                                  | the entire request to be processed. This option takes an <b>int</b> value. Note      |
| 40944 |                                                                                                                  | that not all implementations allow this option to be set.                            |
| 40945 | SO_SNDTIMEO                                                                                                      | Sets the timeout value specifying the amount of time that an output                  |
| 40946 |                                                                                                                  | function blocks because flow control prevents data from being sent. If a             |
| 40947 |                                                                                                                  | send operation has blocked for this time, it shall return with a partial             |
| 40948 |                                                                                                                  | count or with <i>errno</i> set to [EAGAIN] or [EWOULDBLOCK] if no data is            |
| 40949 |                                                                                                                  | sent. The default for this option is zero, which indicates that a send               |
| 40950 |                                                                                                                  | operation shall not time out. This option stores a <b>timeval</b> structure. Note    |
| 40951 |                                                                                                                  | that not all implementations allow this option to be set.                            |
| 40952 | For Boolean options, 0 indicates that the option is disabled and 1 indicates that the option is                  |                                                                                      |
| 40953 | enabled.                                                                                                         |                                                                                      |
| 40954 | Options at other protocol levels vary in format and name.                                                        |                                                                                      |
| 40955 | <b>RETURN VALUE</b>                                                                                              |                                                                                      |
| 40956 | Upon successful completion, <i>setsockopt()</i> shall return 0. Otherwise, -1 shall be returned and <i>errno</i> |                                                                                      |
| 40957 | set to indicate the error.                                                                                       |                                                                                      |
| 40958 | <b>ERRORS</b>                                                                                                    |                                                                                      |
| 40959 | The <i>setsockopt()</i> function shall fail if:                                                                  |                                                                                      |
| 40960 | [EBADF]                                                                                                          | The <i>socket</i> argument is not a valid file descriptor.                           |
| 40961 | [EDOM]                                                                                                           | The send and receive timeout values are too big to fit into the timeout fields in    |
| 40962 |                                                                                                                  | the socket structure.                                                                |

- 40963 [EINVAL] The specified option is invalid at the specified socket level or the socket has  
40964 been shut down.
- 40965 [EISCONN] The socket is already connected, and a specified option cannot be set while the  
40966 socket is connected.
- 40967 [ENOPROTOOPT]  
40968 The option is not supported by the protocol.
- 40969 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 40970 The *setsockopt()* function may fail if:
- 40971 [ENOMEM] There was insufficient memory available for the operation to complete.
- 40972 [ENOBUFS] Insufficient resources are available in the system to complete the call.
- 40973 **EXAMPLES**
- 40974 None.
- 40975 **APPLICATION USAGE**
- 40976 The *setsockopt()* function provides an application program with the means to control socket  
40977 behavior. An application program can use *setsockopt()* to allocate buffer space, control timeouts,  
40978 or permit socket data broadcasts. The `<sys/socket.h>` header defines the socket-level options  
40979 available to *setsockopt()*.
- 40980 Options may exist at multiple protocol levels. The `SO_` options are always present at the  
40981 uppermost socket level.
- 40982 **RATIONALE**
- 40983 None.
- 40984 **FUTURE DIRECTIONS**
- 40985 None.
- 40986 **SEE ALSO**
- 40987 Section 2.10 (on page 58), *bind()*, *endprotoent()*, *getsockopt()*, *socket()*, the Base Definitions volume  
40988 of IEEE Std 1003.1-2001, `<netinet/in.h>`, `<sys/socket.h>`
- 40989 **CHANGE HISTORY**
- 40990 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

40991 **NAME**

40992            setstate — switch pseudo-random number generator state arrays

40993 **SYNOPSIS**

40994 xSI        #include <stdlib.h>

40995            char \*setstate(const char \*state);

40996

40997 **DESCRIPTION**

40998            Refer to *initstate()*.

40999 **NAME**

41000        setuid — set user ID

41001 **SYNOPSIS**

41002        #include &lt;unistd.h&gt;

41003        int setuid(uid\_t uid);

41004 **DESCRIPTION**41005        If the process has appropriate privileges, *setuid()* shall set the real user ID, effective user ID, and  
41006        the saved set-user-ID of the calling process to *uid*.41007        If the process does not have appropriate privileges, but *uid* is equal to the real user ID or the  
41008        saved set-user-ID, *setuid()* shall set the effective user ID to *uid*; the real user ID and saved set-  
41009        user-ID shall remain unchanged.41010        The *setuid()* function shall not affect the supplementary group list in any way.41011 **RETURN VALUE**41012        Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
41013        indicate the error.41014 **ERRORS**41015        The *setuid()* function shall fail, return -1, and set *errno* to the corresponding value if one or more  
41016        of the following are true:41017        [EINVAL]        The value of the *uid* argument is invalid and not supported by the  
41018        implementation.41019        [EPERM]        The process does not have appropriate privileges and *uid* does not match the  
41020        real user ID or the saved set-user-ID.41021 **EXAMPLES**

41022        None.

41023 **APPLICATION USAGE**

41024        None.

41025 **RATIONALE**41026        The various behaviors of the *setuid()* and *setgid()* functions when called by non-privileged  
41027        processes reflect the behavior of different historical implementations. For portability, it is  
41028        recommended that new non-privileged applications use the *seteuid()* and *setegid()* functions  
41029        instead.41030        The saved set-user-ID capability allows a program to regain the effective user ID established at  
41031        the last *exec* call. Similarly, the saved set-group-ID capability allows a program to regain the  
41032        effective group ID established at the last *exec* call. These capabilities are derived from System V.  
41033        Without them, a program might have to run as superuser in order to perform the same  
41034        functions, because superuser can write on the user's files. This is a problem because such a  
41035        program can write on any user's files, and so must be carefully written to emulate the  
41036        permissions of the calling process properly. In System V, these capabilities have traditionally  
41037        been implemented only via the *setuid()* and *setgid()* functions for non-privileged processes. The  
41038        fact that the behavior of those functions was different for privileged processes made them  
41039        difficult to use. The POSIX.1-1990 standard defined the *setuid()* function to behave differently  
41040        for privileged and unprivileged users. When the caller had the appropriate privilege, the  
41041        function set the calling process' real user ID, effective user ID, and saved set-user ID on  
41042        implementations that supported it. When the caller did not have the appropriate privilege, the  
41043        function set only the effective user ID, subject to permission checks. The former use is generally  
41044        needed for utilities like *login* and *su*, which are not conforming applications and thus outside the

41045 scope of IEEE Std 1003.1-2001. These utilities wish to change the user ID irrevocably to a new  
41046 value, generally that of an unprivileged user. The latter use is needed for conforming  
41047 applications that are installed with the set-user-ID bit and need to perform operations using the  
41048 real user ID.

41049 IEEE Std 1003.1-2001 augments the latter functionality with a mandatory feature named  
41050 `_POSIX_SAVED_IDS`. This feature permits a set-user-ID application to switch its effective user  
41051 ID back and forth between the values of its *exec*-time real user ID and effective user ID.  
41052 Unfortunately, the POSIX.1-1990 standard did not permit a conforming application using this  
41053 feature to work properly when it happened to be executed with the (implementation-defined)  
41054 appropriate privilege. Furthermore, the application did not even have a means to tell whether it  
41055 had this privilege. Since the saved set-user-ID feature is quite desirable for applications, as  
41056 evidenced by the fact that NIST required it in FIPS 151-2, it has been mandated by  
41057 IEEE Std 1003.1-2001. However, there are implementors who have been reluctant to support it  
41058 given the limitation described above.

41059 The 4.3BSD system handles the problem by supporting separate functions: *setuid()* (which  
41060 always sets both the real and effective user IDs, like *setuid()* in IEEE Std 1003.1-2001 for  
41061 privileged users), and *seteuid()* (which always sets just the effective user ID, like *setuid()* in  
41062 IEEE Std 1003.1-2001 for non-privileged users). This separation of functionality into distinct  
41063 functions seems desirable. 4.3BSD does not support the saved set-user-ID feature. It supports  
41064 similar functionality of switching the effective user ID back and forth via *setreuid()*, which  
41065 permits reversing the real and effective user IDs. This model seems less desirable than the saved  
41066 set-user-ID because the real user ID changes as a side effect. The current 4.4BSD includes saved  
41067 effective IDs and uses them for *seteuid()* and *setegid()* as described above. The *setreuid()* and  
41068 *setregid()* functions will be deprecated or removed.

41069 The solution here is:

- 41070 • Require that all implementations support the functionality of the saved set-user-ID, which is  
41071 set by the *exec* functions and by privileged calls to *setuid()*.
- 41072 • Add the *seteuid()* and *setegid()* functions as portable alternatives to *setuid()* and *setgid()* for  
41073 non-privileged and privileged processes.

41074 Historical systems have provided two mechanisms for a set-user-ID process to change its  
41075 effective user ID to be the same as its real user ID in such a way that it could return to the  
41076 original effective user ID: the use of the *setuid()* function in the presence of a saved set-user-ID,  
41077 or the use of the BSD *setreuid()* function, which was able to swap the real and effective user IDs.  
41078 The changes included in IEEE Std 1003.1-2001 provide a new mechanism using *seteuid()*  
41079 in conjunction with a saved set-user-ID. Thus, all implementations with the new *seteuid()*  
41080 mechanism will have a saved set-user-ID for each process, and most of the behavior controlled  
41081 by `_POSIX_SAVED_IDS` has been changed to agree with the case where the option was defined.  
41082 The *kill()* function is an exception. Implementors of the new *seteuid()* mechanism will generally  
41083 be required to maintain compatibility with the older mechanisms previously supported by their  
41084 systems. However, compatibility with this use of *setreuid()* and with the `_POSIX_SAVED_IDS`  
41085 behavior of *kill()* is unfortunately complicated. If an implementation with a saved set-user-ID  
41086 allows a process to use *setreuid()* to swap its real and effective user IDs, but were to leave the  
41087 saved set-user-ID unmodified, the process would then have an effective user ID equal to the  
41088 original real user ID, and both real and saved set-user-ID would be equal to the original effective  
41089 user ID. In that state, the real user would be unable to kill the process, even though the effective  
41090 user ID of the process matches that of the real user, if the *kill()* behavior of `_POSIX_SAVED_IDS`  
41091 was used. This is obviously not acceptable. The alternative choice, which is used in at least one  
41092 implementation, is to change the saved set-user-ID to the effective user ID during most calls to  
41093 *setreuid()*. The standard developers considered that alternative to be less correct than the

41094 retention of the old behavior of *kill()* in such systems. Current conforming applications shall  
41095 accommodate either behavior from *kill()*, and there appears to be no strong reason for *kill()* to  
41096 check the saved set-user-ID rather than the effective user ID.

41097 **FUTURE DIRECTIONS**

41098 None.

41099 **SEE ALSO**

41100 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, the  
41101 Base Definitions volume of IEEE Std 1003.1-2001, `<sys/types.h>`, `<unistd.h>`

41102 **CHANGE HISTORY**

41103 First released in Issue 1. Derived from Issue 1 of the SVID.

41104 **Issue 6**

41105 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

41106 The following new requirements on POSIX implementations derive from alignment with the  
41107 Single UNIX Specification:

- 41108 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
41109 required for conforming implementations of previous POSIX specifications, it was not  
41110 required for UNIX applications.
- 41111 • The functionality associated with `_POSIX_SAVED_IDS` is now mandatory. This is a FIPS  
41112 requirement.

41113 The following changes were made to align with the IEEE P1003.1a draft standard:

- 41114 • The effects of *setuid()* in processes without appropriate privileges are changed.
- 41115 • A requirement that the supplementary group list is not affected is added.

41116 **NAME**

41117           setutxent — reset the user accounting database to the first entry

41118 **SYNOPSIS**

41119 XSI       #include <utmpx.h>

41120           void setutxent(void);

41121

41122 **DESCRIPTION**

41123           Refer to *endutxent()*.

41124 **NAME**

41125       setvbuf — assign buffering to a stream

41126 **SYNOPSIS**

41127       #include &lt;stdio.h&gt;

41128       int setvbuf(FILE \*restrict *stream*, char \*restrict *buf*, int *type*,  
41129           size\_t *size*);41130 **DESCRIPTION**41131 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
41132       conflict between the requirements described here and the ISO C standard is unintentional. This  
41133       volume of IEEE Std 1003.1-2001 defers to the ISO C standard.41134       The *setvbuf()* function may be used after the stream pointed to by *stream* is associated with an  
41135       open file but before any other operation (other than an unsuccessful call to *setvbuf()*) is  
41136       performed on the stream. The argument *type* determines how *stream* shall be buffered, as  
41137       follows:

- 41138
- {\_IOFBF} shall cause input/output to be fully buffered.
  - {\_IOLBF} shall cause input/output to be line buffered.
  - {\_IONBF} shall cause input/output to be unbuffered.

41141       If *buf* is not a null pointer, the array it points to may be used instead of a buffer allocated by  
41142       *setvbuf()* and the argument *size* specifies the size of the array; otherwise, *size* may determine the  
41143       size of a buffer allocated by the *setvbuf()* function. The contents of the array at any time are  
41144       unspecified.

41145       For information about streams, see Section 2.5 (on page 34).

41146 **RETURN VALUE**41147       Upon successful completion, *setvbuf()* shall return 0. Otherwise, it shall return a non-zero value  
41148 cx       if an invalid value is given for *type* or if the request cannot be honored, and may set *errno* to  
41149       indicate the error.41150 **ERRORS**41151       The *setvbuf()* function may fail if:41152 cx       [EBADF]       The file descriptor underlying *stream* is not valid.41153 **EXAMPLES**

41154       None.

41155 **APPLICATION USAGE**41156       A common source of error is allocating buffer space as an “automatic” variable in a code block,  
41157       and then failing to close the stream in the same block.41158       With *setvbuf()*, allocating a buffer of *size* bytes does not necessarily imply that all of *size* bytes are  
41159       used for the buffer area.41160       Applications should note that many implementations only provide line buffering on input from  
41161       terminal devices.41162 **RATIONALE**

41163       None.

41164 **FUTURE DIRECTIONS**

41165           None.

41166 **SEE ALSO**

41167           Section 2.5 (on page 34), *fopen()*, *setbuf()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
41168           <**stdio.h**>

41169 **CHANGE HISTORY**

41170           First released in Issue 1. Derived from Issue 1 of the SVID.

41171 **Issue 6**

41172           Extensions beyond the ISO C standard are marked.

41173           The *setvbuf()* prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

## 41174 NAME

41175 shm\_open — open a shared memory object (**REALTIME**)

## 41176 SYNOPSIS

41177 SHM #include &lt;sys/mman.h&gt;

41178 int shm\_open(const char \*name, int oflag, mode\_t mode);

41179

## 41180 DESCRIPTION

41181 The *shm\_open()* function shall establish a connection between a shared memory object and a file  
 41182 descriptor. It shall create an open file description that refers to the shared memory object and a  
 41183 file descriptor that refers to that open file description. The file descriptor is used by other  
 41184 functions to refer to that shared memory object. The *name* argument points to a string naming a  
 41185 shared memory object. It is unspecified whether the name appears in the file system and is  
 41186 visible to other functions that take pathnames as arguments. The *name* argument conforms to the  
 41187 construction rules for a pathname. If *name* begins with the slash character, then processes calling  
 41188 *shm\_open()* with the same value of *name* refer to the same shared memory object, as long as that  
 41189 name has not been removed. If *name* does not begin with the slash character, the effect is  
 41190 implementation-defined. The interpretation of slash characters other than the leading slash  
 41191 character in *name* is implementation-defined.

41192 If successful, *shm\_open()* shall return a file descriptor for the shared memory object that is the  
 41193 lowest numbered file descriptor not currently open for that process. The open file description is  
 41194 new, and therefore the file descriptor does not share it with any other processes. It is unspecified  
 41195 whether the file offset is set. The FD\_CLOEXEC file descriptor flag associated with the new file  
 41196 descriptor is set.

41197 The file status flags and file access modes of the open file description are according to the value  
 41198 of *oflag*. The *oflag* argument is the bitwise-inclusive OR of the following flags defined in the  
 41199 <fcntl.h> header. Applications specify exactly one of the first two values (access modes) below  
 41200 in the value of *oflag*:

41201 O\_RDONLY Open for read access only.

41202 O\_RDWR Open for read or write access.

41203 Any combination of the remaining flags may be specified in the value of *oflag*:

41204 O\_CREAT If the shared memory object exists, this flag has no effect, except as noted  
 41205 under O\_EXCL below. Otherwise, the shared memory object is created; the user ID of the shared memory object shall be set to the effective user ID of the  
 41206 process; the group ID of the shared memory object is set to a system default group ID or to the effective group ID of the process. The permission bits of the  
 41207 shared memory object shall be set to the value of the *mode* argument except  
 41208 those set in the file mode creation mask of the process. When bits in *mode*  
 41209 other than the file permission bits are set, the effect is unspecified. The *mode*  
 41210 argument does not affect whether the shared memory object is opened for  
 41211 reading, for writing, or for both. The shared memory object has a size of zero.  
 41212  
 41213

41214 O\_EXCL If O\_EXCL and O\_CREAT are set, *shm\_open()* fails if the shared memory  
 41215 object exists. The check for the existence of the shared memory object and the  
 41216 creation of the object if it does not exist is atomic with respect to other  
 41217 processes executing *shm\_open()* naming the same shared memory object with  
 41218 O\_EXCL and O\_CREAT set. If O\_EXCL is set and O\_CREAT is not set, the  
 41219 result is undefined.

41220 O\_TRUNC If the shared memory object exists, and it is successfully opened O\_RDWR,  
 41221 the object shall be truncated to zero length and the mode and owner shall be  
 41222 unchanged by this function call. The result of using O\_TRUNC with  
 41223 O\_RDONLY is undefined.

41224 When a shared memory object is created, the state of the shared memory object, including all  
 41225 data associated with the shared memory object, persists until the shared memory object is  
 41226 unlinked and all other references are gone. It is unspecified whether the name and shared  
 41227 memory object state remain valid after a system reboot.

#### 41228 RETURN VALUE

41229 Upon successful completion, the *shm\_open()* function shall return a non-negative integer  
 41230 representing the lowest numbered unused file descriptor. Otherwise, it shall return  $-1$  and set  
 41231 *errno* to indicate the error.

#### 41232 ERRORS

41233 The *shm\_open()* function shall fail if:

41234 [EACCES] The shared memory object exists and the permissions specified by *oflag* are  
 41235 denied, or the shared memory object does not exist and permission to create  
 41236 the shared memory object is denied, or O\_TRUNC is specified and write  
 41237 permission is denied.

41238 [EEXIST] O\_CREAT and O\_EXCL are set and the named shared memory object already  
 41239 exists.

41240 [EINTR] The *shm\_open()* operation was interrupted by a signal.

41241 [EINVAL] The *shm\_open()* operation is not supported for the given name.

41242 [EMFILE] Too many file descriptors are currently in use by this process.

41243 [ENAMETOOLONG]

41244 The length of the *name* argument exceeds {PATH\_MAX} or a pathname  
 41245 component is longer than {NAME\_MAX}.

41246 [ENFILE] Too many shared memory objects are currently open in the system.

41247 [ENOENT] O\_CREAT is not set and the named shared memory object does not exist.

41248 [ENOSPC] There is insufficient space for the creation of the new shared memory object.

#### 41249 EXAMPLES

41250 None.

#### 41251 APPLICATION USAGE

41252 None.

#### 41253 RATIONALE

41254 When the Memory Mapped Files option is supported, the normal *open()* call is used to obtain a  
 41255 descriptor to a file to be mapped according to existing practice with *mmap()*. When the Shared  
 41256 Memory Objects option is supported, the *shm\_open()* function shall obtain a descriptor to the  
 41257 shared memory object to be mapped.

41258 There is ample precedent for having a file descriptor represent several types of objects. In the  
 41259 POSIX.1-1990 standard, a file descriptor can represent a file, a pipe, a FIFO, a tty, or a directory.  
 41260 Many implementations simply have an operations vector, which is indexed by the file descriptor  
 41261 type and does very different operations. Note that in some cases the file descriptor passed to  
 41262 generic operations on file descriptors is returned by *open()* or *creat()* and in some cases returned  
 41263 by alternate functions, such as *pipe()*. The latter technique is used by *shm\_open()*.

41264 Note that such shared memory objects can actually be implemented as mapped files. In both  
41265 cases, the size can be set after the open using *ftruncate()*. The *shm\_open()* function itself does not  
41266 create a shared object of a specified size because this would duplicate an extant function that set  
41267 the size of an object referenced by a file descriptor.

41268 On implementations where memory objects are implemented using the existing file system, the  
41269 *shm\_open()* function may be implemented using a macro that invokes *open()*, and the  
41270 *shm\_unlink()* function may be implemented using a macro that invokes *unlink()*.

41271 For implementations without a permanent file system, the definition of the name of the memory  
41272 objects is allowed not to survive a system reboot. Note that this allows systems with a  
41273 permanent file system to implement memory objects as data structures internal to the  
41274 implementation as well.

41275 On implementations that choose to implement memory objects using memory directly, a  
41276 *shm\_open()* followed by an *ftruncate()* and *close()* can be used to preallocate a shared memory  
41277 area and to set the size of that preallocation. This may be necessary for systems without virtual  
41278 memory hardware support in order to ensure that the memory is contiguous.

41279 The set of valid open flags to *shm\_open()* was restricted to *O\_RDONLY*, *O\_RDWR*, *O\_CREAT*,  
41280 and *O\_TRUNC* because these could be easily implemented on most memory mapping systems.  
41281 This volume of IEEE Std 1003.1-2001 is silent on the results if the implementation cannot supply  
41282 the requested file access because of implementation-defined reasons, including hardware ones.

41283 The error conditions [EACCES] and [ENOTSUP] are provided to inform the application that the  
41284 implementation cannot complete a request.

41285 [EACCES] indicates for implementation-defined reasons, probably hardware-related, that the  
41286 implementation cannot comply with a requested mode because it conflicts with another  
41287 requested mode. An example might be that an application desires to open a memory object two  
41288 times, mapping different areas with different access modes. If the implementation cannot map a  
41289 single area into a process space in two places, which would be required if different access modes  
41290 were required for the two areas, then the implementation may inform the application at the time  
41291 of the second open.

41292 [ENOTSUP] indicates for implementation-defined reasons, probably hardware-related, that the  
41293 implementation cannot comply with a requested mode at all. An example would be that the  
41294 hardware of the implementation cannot support write-only shared memory areas.

41295 On all implementations, it may be desirable to restrict the location of the memory objects to  
41296 specific file systems for performance (such as a RAM disk) or implementation-defined reasons  
41297 (shared memory supported directly only on certain file systems). The *shm\_open()* function may  
41298 be used to enforce these restrictions. There are a number of methods available to the application  
41299 to determine an appropriate name of the file or the location of an appropriate directory. One  
41300 way is from the environment via *getenv()*. Another would be from a configuration file.

41301 This volume of IEEE Std 1003.1-2001 specifies that memory objects have initial contents of zero  
41302 when created. This is consistent with current behavior for both files and newly allocated  
41303 memory. For those implementations that use physical memory, it would be possible that such  
41304 implementations could simply use available memory and give it to the process uninitialized.  
41305 This, however, is not consistent with standard behavior for the uninitialized data area, the stack,  
41306 and of course, files. Finally, it is highly desirable to set the allocated memory to zero for security  
41307 reasons. Thus, initializing memory objects to zero is required.

41308 **FUTURE DIRECTIONS**

41309           None.

41310 **SEE ALSO**

41311           *close()*, *dup()*, *exec*, *fcntl()*, *mmap()*, *shmat()*, *shmctl()*, *shmdt()*, *shm\_unlink()*, *umask()*, the Base  
41312           Definitions volume of IEEE Std 1003.1-2001, <fcntl.h>, <sys/mman.h>

41313 **CHANGE HISTORY**

41314           First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

41315 **Issue 6**

41316           The *shm\_open()* function is marked as part of the Shared Memory Objects option.

41317           The [ENOSYS] error condition has been removed as stubs need not be provided if an  
41318           implementation does not support the Shared Memory Objects option.

41319 **NAME**41320 shm\_unlink — remove a shared memory object (**REALTIME**)41321 **SYNOPSIS**

41322 SHM #include &lt;sys/mman.h&gt;

41323 int shm\_unlink(const char \*name);

41324

41325 **DESCRIPTION**41326 The *shm\_unlink()* function shall remove the name of the shared memory object named by the  
41327 string pointed to by *name*.41328 If one or more references to the shared memory object exist when the object is unlinked, the  
41329 name shall be removed before *shm\_unlink()* returns, but the removal of the memory object  
41330 contents shall be postponed until all open and map references to the shared memory object have  
41331 been removed.41332 Even if the object continues to exist after the last *shm\_unlink()*, reuse of the name shall  
41333 subsequently cause *shm\_open()* to behave as if no shared memory object of this name exists (that  
41334 is, *shm\_open()* will fail if **O\_CREAT** is not set, or will create a new shared memory object if  
41335 **O\_CREAT** is set).41336 **RETURN VALUE**41337 Upon successful completion, a value of zero shall be returned. Otherwise, a value of -1 shall be  
41338 returned and *errno* set to indicate the error. If -1 is returned, the named shared memory object  
41339 shall not be changed by this function call.41340 **ERRORS**41341 The *shm\_unlink()* function shall fail if:

41342 [EACCES] Permission is denied to unlink the named shared memory object.

41343 [ENAMETOOLONG]

41344 The length of the *name* argument exceeds {**PATH\_MAX**} or a pathname  
41345 component is longer than {**NAME\_MAX**}.

41346 [ENOENT] The named shared memory object does not exist.

41347 **EXAMPLES**

41348 None.

41349 **APPLICATION USAGE**41350 Names of memory objects that were allocated with *open()* are deleted with *unlink()* in the usual  
41351 fashion. Names of memory objects that were allocated with *shm\_open()* are deleted with  
41352 *shm\_unlink()*. Note that the actual memory object is not destroyed until the last close and  
41353 unmap on it have occurred if it was already in use.41354 **RATIONALE**

41355 None.

41356 **FUTURE DIRECTIONS**

41357 None.

41358 **SEE ALSO**41359 *close()*, *mmap()*, *munmap()*, *shmat()*, *shmctl()*, *shmdt()*, *shm\_open()*, the Base Definitions volume  
41360 of IEEE Std 1003.1-2001, <sys/mman.h>

## 41361 CHANGE HISTORY

41362 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

## 41363 Issue 6

41364 The *shm\_unlink()* function is marked as part of the Shared Memory Objects option.

41365 In the DESCRIPTION, text is added to clarify that reusing the same name after a *shm\_unlink()*  
41366 will not attach to the old shared memory object.

41367 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
41368 implementation does not support the Shared Memory Objects option.

41369 **NAME**41370 `shmat` — XSI shared memory attach operation41371 **SYNOPSIS**41372 XSI 

```
#include <sys/shm.h>
```

41373 

```
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

41374

41375 **DESCRIPTION**

41376 The `shmat()` function operates on XSI shared memory (see the Base Definitions volume of  
 41377 IEEE Std 1003.1-2001, Section 3.340, Shared Memory Object). It is unspecified whether this  
 41378 function interoperates with the realtime interprocess communication facilities defined in Section  
 41379 2.8 (on page 41).

41380 The `shmat()` function attaches the shared memory segment associated with the shared memory  
 41381 identifier specified by *shmid* to the address space of the calling process. The segment is attached  
 41382 at the address specified by one of the following criteria:

- 41383 • If *shmaddr* is a null pointer, the segment is attached at the first available address as selected  
 41384 by the system.
- 41385 • If *shmaddr* is not a null pointer and (*shmflg* &SHM\_RND) is non-zero, the segment is attached  
 41386 at the address given by (*shmaddr* - ((*uintptr\_t*)*shmaddr* %SHMLBA)). The character '`%`' is the  
 41387 C-language remainder operator.
- 41388 • If *shmaddr* is not a null pointer and (*shmflg* &SHM\_RND) is 0, the segment is attached at the  
 41389 address given by *shmaddr*.
- 41390 • The segment is attached for reading if (*shmflg* &SHM\_RDONLY) is non-zero and the calling  
 41391 process has read permission; otherwise, if it is 0 and the calling process has read and write  
 41392 permission, the segment is attached for reading and writing.

41393 **RETURN VALUE**

41394 Upon successful completion, `shmat()` shall increment the value of *shm\_nattch* in the data  
 41395 structure associated with the shared memory ID of the attached shared memory segment and  
 41396 return the segment's start address.

41397 Otherwise, the shared memory segment shall not be attached, `shmat()` shall return `-1`, and *errno*  
 41398 shall be set to indicate the error.

41399 **ERRORS**41400 The `shmat()` function shall fail if:

- |                                           |          |                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------------------------------|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 41401<br>41402                            | [EACCES] | Operation permission is denied to the calling process; see Section 2.7 (on page 39).                                                                                                                                                                                                                                                                                                                               |
| 41403<br>41404<br>41405<br>41406<br>41407 | [EINVAL] | The value of <i>shmid</i> is not a valid shared memory identifier, the <i>shmaddr</i> is not a null pointer, and the value of ( <i>shmaddr</i> - (( <i>uintptr_t</i> ) <i>shmaddr</i> %SHMLBA)) is an illegal address for attaching shared memory; or the <i>shmaddr</i> is not a null pointer, ( <i>shmflg</i> &SHM_RND) is 0, and the value of <i>shmaddr</i> is an illegal address for attaching shared memory. |
| 41408<br>41409                            | [EMFILE] | The number of shared memory segments attached to the calling process would exceed the system-imposed limit.                                                                                                                                                                                                                                                                                                        |
| 41410<br>41411                            | [ENOMEM] | The available data space is not large enough to accommodate the shared memory segment.                                                                                                                                                                                                                                                                                                                             |

41412 **EXAMPLES**

41413 None.

41414 **APPLICATION USAGE**

41415 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
41416 Application developers who need to use IPC should design their applications so that modules  
41417 using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the  
41418 alternative interfaces.

41419 **RATIONALE**

41420 None.

41421 **FUTURE DIRECTIONS**

41422 None.

41423 **SEE ALSO**

41424 Section 2.7 (on page 39), Section 2.8 (on page 41), *exec*, *exit()*, *fork()*, *shmctl()*, *shmdt()*, *shmget()*,  
41425 *shm\_open()*, *shm\_unlink()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/shm.h>

41426 **CHANGE HISTORY**

41427 First released in Issue 2. Derived from Issue 2 of the SVID.

41428 **Issue 5**

41429 Moved from SHARED MEMORY to BASE.

41430 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
41431 DIRECTIONS to a new APPLICATION USAGE section.

41432 **Issue 6**

41433 The Open Group Corrigendum U021/13 is applied.

41434 **NAME**

41435 shmctl — XSI shared memory control operations

41436 **SYNOPSIS**41437 XSI 

```
#include <sys/shm.h>
```

41438 

```
int shmctl(int shmid, int cmd, struct shm_id_ds *buf);
```

41439

41440 **DESCRIPTION**

41441 The *shmctl()* function operates on XSI shared memory (see the Base Definitions volume of  
 41442 IEEE Std 1003.1-2001, Section 3.340, Shared Memory Object). It is unspecified whether this  
 41443 function interoperates with the realtime interprocess communication facilities defined in Section  
 41444 2.8 (on page 41).

41445 The *shmctl()* function provides a variety of shared memory control operations as specified by  
 41446 *cmd*. The following values for *cmd* are available:

41447 **IPC\_STAT** Place the current value of each member of the **shm\_id\_ds** data structure  
 41448 associated with *shmid* into the structure pointed to by *buf*. The contents of the  
 41449 structure are defined in `<sys/shm.h>`.

41450 **IPC\_SET** Set the value of the following members of the **shm\_id\_ds** data structure  
 41451 associated with *shmid* to the corresponding value found in the structure  
 41452 pointed to by *buf*:

41453 shm\_perm.uid  
 41454 shm\_perm.gid  
 41455 shm\_perm.mode Low-order nine bits.

41456 **IPC\_SET** can only be executed by a process that has an effective user ID equal  
 41457 to either that of a process with appropriate privileges or to the value of  
 41458 *shm\_perm.cuid* or *shm\_perm.uid* in the **shm\_id\_ds** data structure associated with  
 41459 *shmid*.

41460 **IPC\_RMID** Remove the shared memory identifier specified by *shmid* from the system and  
 41461 destroy the shared memory segment and **shm\_id\_ds** data structure associated  
 41462 with it. **IPC\_RMID** can only be executed by a process that has an effective user  
 41463 ID equal to either that of a process with appropriate privileges or to the value  
 41464 of *shm\_perm.cuid* or *shm\_perm.uid* in the **shm\_id\_ds** data structure associated  
 41465 with *shmid*.

41466 **RETURN VALUE**

41467 Upon successful completion, *shmctl()* shall return 0; otherwise, it shall return `-1` and set *errno* to  
 41468 indicate the error.

41469 **ERRORS**41470 The *shmctl()* function shall fail if:

41471 **[EACCES]** The argument *cmd* is equal to **IPC\_STAT** and the calling process does not have  
 41472 read permission; see Section 2.7 (on page 39).

41473 **[EINVAL]** The value of *shmid* is not a valid shared memory identifier, or the value of *cmd*  
 41474 is not a valid command.

41475 **[EPERM]** The argument *cmd* is equal to **IPC\_RMID** or **IPC\_SET** and the effective user ID  
 41476 of the calling process is not equal to that of a process with appropriate  
 41477 privileges and it is not equal to the value of *shm\_perm.cuid* or *shm\_perm.uid* in  
 41478 the data structure associated with *shmid*.

- 41479 The *shmctl()* function may fail if:
- 41480 [EOVERFLOW] The *cmd* argument is `IPC_STAT` and the *gid* or *uid* value is too large to be  
41481 stored in the structure pointed to by the *buf* argument.
- 41482 **EXAMPLES**
- 41483 None.
- 41484 **APPLICATION USAGE**
- 41485 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
41486 Application developers who need to use IPC should design their applications so that modules  
41487 using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the  
41488 alternative interfaces.
- 41489 **RATIONALE**
- 41490 None.
- 41491 **FUTURE DIRECTIONS**
- 41492 None.
- 41493 **SEE ALSO**
- 41494 Section 2.7 (on page 39), Section 2.8 (on page 41), *shmat()*, *shmdt()*, *shmget()*, *shm\_open()*,  
41495 *shm\_unlink()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/shm.h>
- 41496 **CHANGE HISTORY**
- 41497 First released in Issue 2. Derived from Issue 2 of the SVID.
- 41498 **Issue 5**
- 41499 Moved from SHARED MEMORY to BASE.
- 41500 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
41501 DIRECTIONS to a new APPLICATION USAGE section.

41502 **NAME**

41503 shmdt — XSI shared memory detach operation

41504 **SYNOPSIS**41505 XSI 

```
#include <sys/shm.h>
```

41506 

```
int shmdt(const void *shmaddr);
```

41507

41508 **DESCRIPTION**

41509 The *shmdt()* function operates on XSI shared memory (see the Base Definitions volume of  
 41510 IEEE Std 1003.1-2001, Section 3.340, Shared Memory Object). It is unspecified whether this  
 41511 function interoperates with the realtime interprocess communication facilities defined in Section  
 41512 2.8 (on page 41).

41513 The *shmdt()* function detaches the shared memory segment located at the address specified by  
 41514 *shmaddr* from the address space of the calling process.

41515 **RETURN VALUE**

41516 Upon successful completion, *shmdt()* shall decrement the value of *shm\_nattch* in the data  
 41517 structure associated with the shared memory ID of the attached shared memory segment and  
 41518 return 0.

41519 Otherwise, the shared memory segment shall not be detached, *shmdt()* shall return  $-1$ , and *errno*  
 41520 shall be set to indicate the error.

41521 **ERRORS**41522 The *shmdt()* function shall fail if:

41523 [EINVAL] The value of *shmaddr* is not the data segment start address of a shared  
 41524 memory segment.

41525 **EXAMPLES**

41526 None.

41527 **APPLICATION USAGE**

41528 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
 41529 Application developers who need to use IPC should design their applications so that modules  
 41530 using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the  
 41531 alternative interfaces.

41532 **RATIONALE**

41533 None.

41534 **FUTURE DIRECTIONS**

41535 None.

41536 **SEE ALSO**

41537 Section 2.7 (on page 39), Section 2.8 (on page 41), *exec*, *exit()*, *fork()*, *shmat()*, *shmctl()*, *shmget()*,  
 41538 *shm\_open()*, *shm\_unlink()*, the Base Definitions volume of IEEE Std 1003.1-2001, *<sys/shm.h>*

41539 **CHANGE HISTORY**

41540 First released in Issue 2. Derived from Issue 2 of the SVID.

41541 **Issue 5**

41542 Moved from SHARED MEMORY to BASE.

41543 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
 41544 DIRECTIONS to a new APPLICATION USAGE section.

41545 **NAME**

41546 shmget — get an XSI shared memory segment

41547 **SYNOPSIS**41548 XSI 

```
#include <sys/shm.h>
```

41549 

```
int shmget(key_t key, size_t size, int shmflg);
```

41550

41551 **DESCRIPTION**

41552 The *shmget()* function operates on XSI shared memory (see the Base Definitions volume of  
 41553 IEEE Std 1003.1-2001, Section 3.340, Shared Memory Object). It is unspecified whether this  
 41554 function interoperates with the realtime interprocess communication facilities defined in Section  
 41555 2.8 (on page 41).

41556 The *shmget()* function shall return the shared memory identifier associated with *key*.

41557 A shared memory identifier, associated data structure, and shared memory segment of at least  
 41558 *size* bytes (see <sys/shm.h>) are created for *key* if one of the following is true:

- 41559 • The argument *key* is equal to `IPC_PRIVATE`.
- 41560 • The argument *key* does not already have a shared memory identifier associated with it and  
 41561 (*shmflg* & `IPC_CREAT`) is non-zero.

41562 Upon creation, the data structure associated with the new shared memory identifier shall be  
 41563 initialized as follows:

- 41564 • The values of *shm\_perm.cuid*, *shm\_perm.uid*, *shm\_perm.cgid*, and *shm\_perm.gid* are set equal to  
 41565 the effective user ID and effective group ID, respectively, of the calling process.
- 41566 • The low-order nine bits of *shm\_perm.mode* are set equal to the low-order nine bits of *shmflg*.
- 41567 • The value of *shm\_segsz* is set equal to the value of *size*.
- 41568 • The values of *shm\_lpid*, *shm\_nattch*, *shm\_atime*, and *shm\_dtime* are set equal to 0.
- 41569 • The value of *shm\_ctime* is set equal to the current time.

41570 When the shared memory segment is created, it shall be initialized with all zero values.

41571 **RETURN VALUE**

41572 Upon successful completion, *shmget()* shall return a non-negative integer, namely a shared  
 41573 memory identifier; otherwise, it shall return `-1` and set *errno* to indicate the error.

41574 **ERRORS**

41575 The *shmget()* function shall fail if:

- |                         |          |                                                                                                                                                                                               |
|-------------------------|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 41576<br>41577<br>41578 | [EACCES] | A shared memory identifier exists for <i>key</i> but operation permission as specified by the low-order nine bits of <i>shmflg</i> would not be granted; see Section 2.7 (on page 39).        |
| 41579<br>41580          | [EEXIST] | A shared memory identifier exists for the argument <i>key</i> but ( <i>shmflg</i> & <code>IPC_CREAT</code> ) && ( <i>shmflg</i> & <code>IPC_EXCL</code> ) is non-zero.                        |
| 41581<br>41582          | [EINVAL] | A shared memory segment is to be created and the value of <i>size</i> is less than the system-imposed minimum or greater than the system-imposed maximum.                                     |
| 41583<br>41584<br>41585 | [EINVAL] | No shared memory segment is to be created and a shared memory segment exists for <i>key</i> but the size of the segment associated with it is less than <i>size</i> and <i>size</i> is not 0. |

- 41586 [ENOENT] A shared memory identifier does not exist for the argument *key* and (*shmflg*  
41587 &IPC\_CREAT) is 0.
- 41588 [ENOMEM] A shared memory identifier and associated shared memory segment shall be  
41589 created, but the amount of available physical memory is not sufficient to fill  
41590 the request.
- 41591 [ENOSPC] A shared memory identifier is to be created, but the system-imposed limit on  
41592 the maximum number of allowed shared memory identifiers system-wide  
41593 would be exceeded.
- 41594 **EXAMPLES**
- 41595 None.
- 41596 **APPLICATION USAGE**
- 41597 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
41598 Application developers who need to use IPC should design their applications so that modules  
41599 using the IPC routines described in Section 2.7 (on page 39) can be easily modified to use the  
41600 alternative interfaces.
- 41601 **RATIONALE**
- 41602 None.
- 41603 **FUTURE DIRECTIONS**
- 41604 None.
- 41605 **SEE ALSO**
- 41606 Section 2.7 (on page 39), Section 2.8 (on page 41), *shmat()*, *shmctl()*, *shmdt()*, *shm\_open()*,  
41607 *shm\_unlink()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/shm.h>
- 41608 **CHANGE HISTORY**
- 41609 First released in Issue 2. Derived from Issue 2 of the SVID.
- 41610 **Issue 5**
- 41611 Moved from SHARED MEMORY to BASE.
- 41612 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
41613 DIRECTIONS to a new APPLICATION USAGE section.

41614 **NAME**

41615 shutdown — shut down socket send and receive operations

41616 **SYNOPSIS**

41617 #include <sys/socket.h>

41618 int shutdown(int *socket*, int *how*);

41619 **DESCRIPTION**

41620 The *shutdown()* function shall cause all or part of a full-duplex connection on the socket  
41621 associated with the file descriptor *socket* to be shut down.

41622 The *shutdown()* function takes the following arguments:

41623 *socket* Specifies the file descriptor of the socket.

41624 *how* Specifies the type of shutdown. The values are as follows:

41625 SHUT\_RD Disables further receive operations.

41626 SHUT\_WR Disables further send operations.

41627 SHUT\_RDWR Disables further send and receive operations.

41628 The *shutdown()* function disables subsequent send and/or receive operations on a socket,  
41629 depending on the value of the *how* argument.

41630 **RETURN VALUE**

41631 Upon successful completion, *shutdown()* shall return 0; otherwise, -1 shall be returned and *errno*  
41632 set to indicate the error.

41633 **ERRORS**

41634 The *shutdown()* function shall fail if:

41635 [EBADF] The *socket* argument is not a valid file descriptor.

41636 [EINVAL] The *how* argument is invalid.

41637 [ENOTCONN] The socket is not connected.

41638 [ENOTSOCK] The *socket* argument does not refer to a socket.

41639 The *shutdown()* function may fail if:

41640 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

41641 **EXAMPLES**

41642 None.

41643 **APPLICATION USAGE**

41644 None.

41645 **RATIONALE**

41646 None.

41647 **FUTURE DIRECTIONS**

41648 None.

41649 **SEE ALSO**

41650 *getsockopt()*, *read()*, *recv()*, *recvfrom()*, *recvmsg()*, *select()*, *send()*, *sendto()*, *setsockopt()*, *socket()*,  
41651 *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/socket.h>

41652 **CHANGE HISTORY**

41653 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

## 41654 NAME

41655 sigaction — examine and change a signal action

## 41656 SYNOPSIS

41657 cx #include &lt;signal.h&gt;

```
41658 int sigaction(int sig, const struct sigaction *restrict act,
41659 struct sigaction *restrict oact);
```

41660

## 41661 DESCRIPTION

41662 The *sigaction()* function allows the calling process to examine and/or specify the action to be  
 41663 associated with a specific signal. The argument *sig* specifies the signal; acceptable values are  
 41664 defined in <signal.h>.

41665 The structure **sigaction**, used to describe an action to be taken, is defined in the <signal.h>  
 41666 header to include at least the following members:

41667

41668

| Member Type                               | Member Name         | Description                                                                           |
|-------------------------------------------|---------------------|---------------------------------------------------------------------------------------|
| <b>void(*) (int)</b>                      | <i>sa_handler</i>   | Pointer to a signal-catching function or one of the macros SIG_IGN or SIG_DFL.        |
| <b>sigset_t</b>                           | <i>sa_mask</i>      | Additional set of signals to be blocked during execution of signal-catching function. |
| <b>int</b>                                | <i>sa_flags</i>     | Special flags to affect behavior of signal.                                           |
| <b>void(*) (int, siginfo_t *, void *)</b> | <i>sa_sigaction</i> | Pointer to a signal-catching function.                                                |

41677

41678 The storage occupied by *sa\_handler* and *sa\_sigaction* may overlap, and a conforming application  
 41679 shall not use both simultaneously.

41680 If the argument *act* is not a null pointer, it points to a structure specifying the action to be  
 41681 associated with the specified signal. If the argument *oact* is not a null pointer, the action  
 41682 previously associated with the signal is stored in the location pointed to by the argument *oact*. If  
 41683 the argument *act* is a null pointer, signal handling is unchanged; thus, the call can be used to  
 41684 enquire about the current handling of a given signal. The SIGKILL and SIGSTOP signals shall  
 41685 not be added to the signal mask using this mechanism; this restriction shall be enforced by the  
 41686 system without causing an error to be indicated.

41687 If the SA\_SIGINFO flag (see below) is cleared in the *sa\_flags* field of the **sigaction** structure, the  
 41688 XSI|RTS *sa\_handler* field identifies the action to be associated with the specified signal. If the  
 41689 SA\_SIGINFO flag is set in the *sa\_flags* field, and the implementation supports the Realtime  
 41690 Signals Extension option or the XSI Extension option, the *sa\_sigaction* field specifies a signal-  
 41691 catching function. If the SA\_SIGINFO bit is cleared and the *sa\_handler* field specifies a signal-  
 41692 catching function, or if the SA\_SIGINFO bit is set, the *sa\_mask* field identifies a set of signals that  
 41693 shall be added to the signal mask of the thread before the signal-catching function is invoked. If  
 41694 the *sa\_handler* field specifies a signal-catching function, the *sa\_mask* field identifies a set of  
 41695 signals that shall be added to the process' signal mask before the signal-catching function is  
 41696 invoked.

41697 The *sa\_flags* field can be used to modify the behavior of the specified signal.

41698 The following flags, defined in the <signal.h> header, can be set in *sa\_flags*:

|               |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 41699 XSI     | SA_NOCLDSTOP | Do not generate SIGCHLD when children stop or stopped children continue.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 41700         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41701         |              | If <i>sig</i> is SIGCHLD and the SA_NOCLDSTOP flag is not set in <i>sa_flags</i> , and the implementation supports the SIGCHLD signal, then a SIGCHLD signal shall be generated for the calling process whenever any of its child processes stop and a SIGCHLD signal may be generated for the calling process whenever any of its stopped child processes are continued. If <i>sig</i> is SIGCHLD and the SA_NOCLDSTOP flag is set in <i>sa_flags</i> , then the implementation shall not generate a SIGCHLD signal in this way.                                         |
| 41702         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41703         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41704 XSI     |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41705         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41706         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41707         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41708 XSI     | SA_ONSTACK   | If set and an alternate signal stack has been declared with <i>sigaltstack()</i> , the signal shall be delivered to the calling process on that stack. Otherwise, the signal shall be delivered on the current stack.                                                                                                                                                                                                                                                                                                                                                     |
| 41709         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41710         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41711 XSI     | SA_RESETHAND | If set, the disposition of the signal shall be reset to SIG_DFL and the SA_SIGINFO flag shall be cleared on entry to the signal handler.                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 41712         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41713         |              | <b>Note:</b> SIGILL and SIGTRAP cannot be automatically reset when delivered; the system silently enforces this restriction.                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 41714         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41715         |              | Otherwise, the disposition of the signal shall not be modified on entry to the signal handler.                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 41716         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41717         |              | In addition, if this flag is set, <i>sigaction()</i> behaves as if the SA_NODEFER flag were also set.                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 41718         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41719 XSI     | SA_RESTART   | This flag affects the behavior of interruptible functions; that is, those specified to fail with <i>errno</i> set to [EINTR]. If set, and a function specified as interruptible is interrupted by this signal, the function shall restart and shall not fail with [EINTR] unless otherwise specified. If the flag is not set, interruptible functions interrupted by this signal shall fail with <i>errno</i> set to [EINTR].                                                                                                                                             |
| 41720         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41721         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41722         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41723         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41724         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41725         | SA_SIGINFO   | If cleared and the signal is caught, the signal-catching function shall be entered as:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 41726         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41727         |              | <pre>void func(int signo);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 41728         |              | where <i>signo</i> is the only argument to the signal-catching function. In this case, the application shall use the <i>sa_handler</i> member to describe the signal-catching function and the application shall not modify the <i>sa_sigaction</i> member.                                                                                                                                                                                                                                                                                                               |
| 41729         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41730         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41731         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41732 XSI RTS |              | If SA_SIGINFO is set and the signal is caught, the signal-catching function shall be entered as:                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 41733         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41734         |              | <pre>void func(int signo, siginfo_t *info, void *context);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 41735         |              | where two additional arguments are passed to the signal-catching function. The second argument shall point to an object of type <b>siginfo_t</b> explaining the reason why the signal was generated; the third argument can be cast to a pointer to an object of type <b>ucontext_t</b> to refer to the receiving process' context that was interrupted when the signal was delivered. In this case, the application shall use the <i>sa_sigaction</i> member to describe the signal-catching function and the application shall not modify the <i>sa_handler</i> member. |
| 41736         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41737         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41738         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41739         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41740         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41741         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41742         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41743         |              | The <i>si_signo</i> member contains the system-generated signal number.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

|               |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 41744 XSI     |              | The <i>si_errno</i> member may contain implementation-defined additional error information; if non-zero, it contains an error number identifying the condition that caused the signal to be generated.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 41745         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41746         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41747 XSI RTS |              | The <i>si_code</i> member contains a code identifying the cause of the signal.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 41748 XSI     |              | If the value of <i>si_code</i> is less than or equal to 0, then the signal was generated by a process and <i>si_pid</i> and <i>si_uid</i> , respectively, indicate the process ID and the real user ID of the sender. The <code>&lt;signal.h&gt;</code> header description contains information about the signal-specific contents of the elements of the <code>siginfo_t</code> type.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 41749         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41750         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41751         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41752         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41753 XSI     | SA_NOCLDWAIT | If set, and <i>sig</i> equals SIGCHLD, child processes of the calling processes shall not be transformed into zombie processes when they terminate. If the calling process subsequently waits for its children, and the process has no unwaited-for children that were transformed into zombie processes, it shall block until all of its children terminate, and <i>wait()</i> , <i>waitid()</i> , and <i>waitpid()</i> shall fail and set <i>errno</i> to [ECHILD]. Otherwise, terminating child processes shall be transformed into zombie processes, unless SIGCHLD is set to SIG_IGN.                                                                                                                                                                                                                                                                                |
| 41754         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41755         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41756         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41757         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41758         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41759         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41760         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41761 XSI     | SA_NODEFER   | If set and <i>sig</i> is caught, <i>sig</i> shall not be added to the process' signal mask on entry to the signal handler unless it is included in <i>sa_mask</i> . Otherwise, <i>sig</i> shall always be added to the process' signal mask on entry to the signal handler.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 41762         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41763         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41764         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41765         |              | When a signal is caught by a signal-catching function installed by <i>sigaction()</i> , a new signal mask is calculated and installed for the duration of the signal-catching function (or until a call to either <i>sigprocmask()</i> or <i>sigsuspend()</i> is made). This mask is formed by taking the union of the current signal mask and the value of the <i>sa_mask</i> for the signal being delivered unless SA_NODEFER or SA_RESETHAND is set, and then including the signal being delivered. If and when the user's signal handler returns normally, the original signal mask is restored.                                                                                                                                                                                                                                                                      |
| 41766         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41767         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41768 XSI     |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41769         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41770         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41771         |              | Once an action is installed for a specific signal, it shall remain installed until another action is explicitly requested (by another call to <i>sigaction()</i> ), until the SA_RESETHAND flag causes resetting of the handler, or until one of the <i>exec</i> functions is called.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 41772 XSI     |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41773         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41774         |              | If the previous action for <i>sig</i> had been established by <i>signal()</i> , the values of the fields returned in the structure pointed to by <i>oact</i> are unspecified, and in particular <i>oact-&gt;sa_handler</i> is not necessarily the same value passed to <i>signal()</i> . However, if a pointer to the same structure or a copy thereof is passed to a subsequent call to <i>sigaction()</i> via the <i>act</i> argument, handling of the signal shall be as if the original call to <i>signal()</i> were repeated.                                                                                                                                                                                                                                                                                                                                        |
| 41775         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41776         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41777         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41778         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41779         |              | If <i>sigaction()</i> fails, no new signal handler is installed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 41780         |              | It is unspecified whether an attempt to set the action for a signal that cannot be caught or ignored to SIG_DFL is ignored or causes an error to be returned with <i>errno</i> set to [EINVAL].                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41781         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41782         |              | If SA_SIGINFO is not set in <i>sa_flags</i> , then the disposition of subsequent occurrences of <i>sig</i> when it is already pending is implementation-defined; the signal-catching function shall be invoked with a single argument. If the implementation supports the Realtime Signals Extension option, and if SA_SIGINFO is set in <i>sa_flags</i> , then subsequent occurrences of <i>sig</i> generated by <i>sigqueue()</i> or as a result of any signal-generating function that supports the specification of an application-defined value (when <i>sig</i> is already pending) shall be queued in FIFO order until delivered or accepted; the signal-catching function shall be invoked with three arguments. The application specified value is passed to the signal-catching function as the <i>si_value</i> member of the <code>siginfo_t</code> structure. |
| 41783         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41784 RTS     |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41785         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41786         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41787         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41788         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41789         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 41790         |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

41791 The result of the use of *sigaction()* and a *sigwait()* function concurrently within a process on the  
41792 same signal is unspecified.

#### 41793 RETURN VALUE

41794 Upon successful completion, *sigaction()* shall return 0; otherwise, -1 shall be returned, *errno* shall  
41795 be set to indicate the error, and no new signal-catching function shall be installed.

#### 41796 ERRORS

41797 The *sigaction()* function shall fail if:

41798 [EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a  
41799 signal that cannot be caught or ignore a signal that cannot be ignored.

41800 [ENOTSUP] The SA\_SIGINFO bit flag is set in the *sa\_flags* field of the **sigaction** structure,  
41801 and the implementation does not support either the Realtime Signals  
41802 Extension option, or the XSI Extension option.

41803 The *sigaction()* function may fail if:

41804 [EINVAL] An attempt was made to set the action to SIG\_DFL for a signal that cannot be  
41805 caught or ignored (or both).

#### 41806 EXAMPLES

41807 None.

#### 41808 APPLICATION USAGE

41809 The *sigaction()* function supersedes the *signal()* function, and should be used in preference. In  
41810 particular, *sigaction()* and *signal()* should not be used in the same process to control the same  
41811 signal. The behavior of reentrant functions, as defined in the DESCRIPTION, is as specified by  
41812 this volume of IEEE Std 1003.1-2001, regardless of invocation from a signal-catching function.  
41813 This is the only intended meaning of the statement that reentrant functions may be used in  
41814 signal-catching functions without restrictions. Applications must still consider all effects of such  
41815 functions on such things as data structures, files, and process state. In particular, application  
41816 writers need to consider the restrictions on interactions when interrupting *sleep()* and  
41817 interactions among multiple handles for a file description. The fact that any specific function is  
41818 listed as reentrant does not necessarily mean that invocation of that function from a signal-  
41819 catching function is recommended.

41820 In order to prevent errors arising from interrupting non-reentrant function calls, applications  
41821 should protect calls to these functions either by blocking the appropriate signals or through the  
41822 use of some programmatic semaphore (see *semget()*, *sem\_init()*, *sem\_open()*, and so on). Note in  
41823 particular that even the “safe” functions may modify *errno*; the signal-catching function, if not  
41824 executing as an independent thread, may want to save and restore its value. Naturally, the same  
41825 principles apply to the reentrancy of application routines and asynchronous data access. Note  
41826 that *longjmp()* and *siglongjmp()* are not in the list of reentrant functions. This is because the code  
41827 executing after *longjmp()* and *siglongjmp()* can call any unsafe functions with the same danger as  
41828 calling those unsafe functions directly from the signal handler. Applications that use *longjmp()*  
41829 and *siglongjmp()* from within signal handlers require rigorous protection in order to be portable.  
41830 Many of the other functions that are excluded from the list are traditionally implemented using  
41831 either *malloc()* or *free()* functions or the standard I/O library, both of which traditionally use  
41832 data structures in a non-reentrant manner. Since any combination of different functions using a  
41833 common data structure can cause reentrancy problems, this volume of IEEE Std 1003.1-2001  
41834 does not define the behavior when any unsafe function is called in a signal handler that  
41835 interrupts an unsafe function.

41836 If the signal occurs other than as the result of calling *abort()*, *kill()*, or *raise()*, the behavior is  
41837 undefined if the signal handler calls any function in the standard library other than one of the

41838 functions listed in the table above or refers to any object with static storage duration other than  
 41839 by assigning a value to a static storage duration variable of type **volatile sig\_atomic\_t**.  
 41840 Furthermore, if such a call fails, the value of *errno* is unspecified.

41841 Usually, the signal is executed on the stack that was in effect before the signal was delivered. An  
 41842 alternate stack may be specified to receive a subset of the signals being caught.

41843 When the signal handler returns, the receiving process resumes execution at the point it was  
 41844 interrupted unless the signal handler makes other arrangements. If *longjmp()* or *\_longjmp()* is  
 41845 used to leave the signal handler, then the signal mask must be explicitly restored by the process.

41846 This volume of IEEE Std 1003.1-2001 defines the third argument of a signal handling function  
 41847 when SA\_SIGINFO is set as a **void \*** instead of a **ucontext\_t \***, but without requiring type  
 41848 checking. New applications should explicitly cast the third argument of the signal handling  
 41849 function to **ucontext\_t \***.

41850 The BSD optional four argument signal handling function is not supported by this volume of  
 41851 IEEE Std 1003.1-2001. The BSD declaration would be:

```
41852 void handler(int sig, int code, struct sigcontext *scp,
41853 char *addr);
```

41854 where *sig* is the signal number, *code* is additional information on certain signals, *scp* is a pointer  
 41855 to the **sigcontext** structure, and *addr* is additional address information. Much the same  
 41856 information is available in the objects pointed to by the second argument of the signal handler  
 41857 specified when SA\_SIGINFO is set.

#### 41858 RATIONALE

41859 Although this volume of IEEE Std 1003.1-2001 requires that signals that cannot be ignored shall  
 41860 not be added to the signal mask when a signal-catching function is entered, there is no explicit  
 41861 requirement that subsequent calls to *sigaction()* reflect this in the information returned in the *oact*  
 41862 argument. In other words, if SIGKILL is included in the *sa\_mask* field of *act*, it is unspecified  
 41863 whether or not a subsequent call to *sigaction()* returns with SIGKILL included in the *sa\_mask*  
 41864 field of *oact*.

41865 The SA\_NOCLDSTOP flag, when supplied in the *act->sa\_flags* parameter, allows overloading  
 41866 SIGCHLD with the System V semantics that each SIGCHLD signal indicates a single terminated  
 41867 child. Most conforming applications that catch SIGCHLD are expected to install signal-catching  
 41868 functions that repeatedly call the *waitpid()* function with the WNOHANG flag set, acting on  
 41869 each child for which status is returned, until *waitpid()* returns zero. If stopped children are not of  
 41870 interest, the use of the SA\_NOCLDSTOP flag can prevent the overhead from invoking the  
 41871 signal-catching routine when they stop.

41872 Some historical implementations also define other mechanisms for stopping processes, such as  
 41873 the *ptrace()* function. These implementations usually do not generate a SIGCHLD signal when  
 41874 processes stop due to this mechanism; however, that is beyond the scope of this volume of  
 41875 IEEE Std 1003.1-2001.

41876 This volume of IEEE Std 1003.1-2001 requires that calls to *sigaction()* that supply a NULL *act*  
 41877 argument succeed, even in the case of signals that cannot be caught or ignored (that is, SIGKILL  
 41878 or SIGSTOP). The System V *signal()* and BSD *sigvec()* functions return [EINVAL] in these cases  
 41879 and, in this respect, their behavior varies from *sigaction()*.

41880 This volume of IEEE Std 1003.1-2001 requires that *sigaction()* properly save and restore a signal  
 41881 action set up by the ISO C standard *signal()* function. However, there is no guarantee that the  
 41882 reverse is true, nor could there be given the greater amount of information conveyed by the  
 41883 **sigaction** structure. Because of this, applications should avoid using both functions for the same  
 41884 signal in the same process. Since this cannot always be avoided in case of general-purpose

- 41885 library routines, they should always be implemented with *sigaction()*.
- 41886 It was intended that the *signal()* function should be implementable as a library routine using  
41887 *sigaction()*.
- 41888 The POSIX Realtime Extension extends the *sigaction()* function as specified by the POSIX.1-1990  
41889 standard to allow the application to request on a per-signal basis via an additional signal action  
41890 flag that the extra parameters, including the application-defined signal value, if any, be passed  
41891 to the signal-catching function.
- 41892 **FUTURE DIRECTIONS**
- 41893 None.
- 41894 **SEE ALSO**
- 41895 Section 2.4 (on page 28), *bsd\_signal()*, *kill()*, *\_longjmp()*, *longjmp()*, *raise()*, *semget()*, *sem\_init()*,  
41896 *sem\_open()*, *sigaddset()*, *sigaltstack()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*, *signal()*,  
41897 *sigprocmask()*, *sigsuspend()*, *wait()*, *waitid()*, *waitpid()*, the Base Definitions volume of  
41898 IEEE Std 1003.1-2001, <**signal.h**>, <**ucontext.h**>
- 41899 **CHANGE HISTORY**
- 41900 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.
- 41901 **Issue 5**
- 41902 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and POSIX  
41903 Threads Extension.
- 41904 In the DESCRIPTION, the second argument to *func* when SA\_SIGINFO is set is no longer  
41905 permitted to be NULL, and the description of permitted **siginfo\_t** contents is expanded by  
41906 reference to <**signal.h**>.
- 41907 Since the X/OPEN UNIX Extension functionality is now folded into the BASE, the [ENOTSUP]  
41908 error is deleted.
- 41909 **Issue 6**
- 41910 The Open Group Corrigendum U028/7 is applied. In the paragraph entitled “Signal Effects on  
41911 Other Functions”, a reference to *sigpending()* is added.
- 41912 In the DESCRIPTION, the text “Signal Generation and Delivery”, “Signal Actions”, and “Signal  
41913 Effects on Other Functions” are moved to a separate section of this volume of  
41914 IEEE Std 1003.1-2001.
- 41915 Text describing functionality from the Realtime Signals option is marked.
- 41916 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:
- 41917 • The [ENOTSUP] error condition is added.
- 41918 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.
- 41919 The **restrict** keyword is added to the *sigaction()* prototype for alignment with the  
41920 ISO/IEC 9899: 1999 standard.
- 41921 References to the *wait3()* function are removed.
- 41922 The SYNOPSIS is marked CX since the presence of this function in the <**signal.h**> header is an  
41923 extension over the ISO C standard.
- 41924 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/57 is applied, changing text in the table  
41925 describing the **sigaction** structure. |

41926 **NAME**

41927 sigaddset — add a signal to a signal set

41928 **SYNOPSIS**41929 **CX** #include <signal.h>

41930 int sigaddset(sigset\_t \*set, int signo);

41931

41932 **DESCRIPTION**41933 The *sigaddset()* function adds the individual signal specified by the *signo* to the signal set pointed  
41934 to by *set*.41935 Applications shall call either *sigemptyset()* or *sigfillset()* at least once for each object of type  
41936 **sigset\_t** prior to any other use of that object. If such an object is not initialized in this way, but is  
41937 nonetheless supplied as an argument to any of *pthread\_sigmask()*, *sigaction()*, *sigaddset()*,  
41938 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or  
41939 *sigwaitinfo()*, the results are undefined.41940 **RETURN VALUE**41941 Upon successful completion, *sigaddset()* shall return 0; otherwise, it shall return -1 and set *errno*  
41942 to indicate the error.41943 **ERRORS**41944 The *sigaddset()* function may fail if:41945 [EINVAL] The value of the *signo* argument is an invalid or unsupported signal number.41946 **EXAMPLES**

41947 None.

41948 **APPLICATION USAGE**

41949 None.

41950 **RATIONALE**

41951 None.

41952 **FUTURE DIRECTIONS**

41953 None.

41954 **SEE ALSO**41955 Section 2.4 (on page 28), *sigaction()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*,  
41956 *sigpending()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
41957 <signal.h>41958 **CHANGE HISTORY**

41959 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

41960 **Issue 5**41961 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
41962 previous issues.41963 **Issue 6**

41964 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

41965 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an  
41966 extension over the ISO C standard.

41967 **NAME**

41968 sigaltstack — set and get signal alternate stack context

41969 **SYNOPSIS**41970 XSI `#include <signal.h>`41971 `int sigaltstack(const stack_t *restrict ss, stack_t *restrict oss);`

41972

41973 **DESCRIPTION**

41974 The *sigaltstack()* function allows a process to define and examine the state of an alternate stack  
 41975 for signal handlers for the current thread. Signals that have been explicitly declared to execute  
 41976 on the alternate stack shall be delivered on the alternate stack.

41977 If *ss* is not a null pointer, it points to a **stack\_t** structure that specifies the alternate signal stack  
 41978 that shall take effect upon return from *sigaltstack()*. The *ss\_flags* member specifies the new stack  
 41979 state. If it is set to *SS\_DISABLE*, the stack is disabled and *ss\_sp* and *ss\_size* are ignored.  
 41980 Otherwise, the stack shall be enabled, and the *ss\_sp* and *ss\_size* members specify the new address  
 41981 and size of the stack.

41982 The range of addresses starting at *ss\_sp* up to but not including *ss\_sp+ss\_size* is available to the  
 41983 implementation for use as the stack. This function makes no assumptions regarding which end  
 41984 is the stack base and in which direction the stack grows as items are pushed.

41985 If *oss* is not a null pointer, on successful completion it shall point to a **stack\_t** structure that  
 41986 specifies the alternate signal stack that was in effect prior to the call to *sigaltstack()*. The *ss\_sp*  
 41987 and *ss\_size* members specify the address and size of that stack. The *ss\_flags* member specifies the  
 41988 stack's state, and may contain one of the following values:

41989 **SS\_ONSTACK** The process is currently executing on the alternate signal stack. Attempts to  
 41990 modify the alternate signal stack while the process is executing on it fail. This  
 41991 flag shall not be modified by processes.

41992 **SS\_DISABLE** The alternate signal stack is currently disabled.

41993 The value *SIGSTKSZ* is a system default specifying the number of bytes that would be used to  
 41994 cover the usual case when manually allocating an alternate stack area. The value *MINSIGSTKSZ*  
 41995 is defined to be the minimum stack size for a signal handler. In computing an alternate stack  
 41996 size, a program should add that amount to its stack requirements to allow for the system  
 41997 implementation overhead. The constants *SS\_ONSTACK*, *SS\_DISABLE*, *SIGSTKSZ*, and  
 41998 *MINSIGSTKSZ* are defined in **<signal.h>**.

41999 After a successful call to one of the *exec* functions, there are no alternate signal stacks in the new  
 42000 process image.

42001 In some implementations, a signal (whether or not indicated to execute on the alternate stack)  
 42002 shall always execute on the alternate stack if it is delivered while another signal is being caught  
 42003 using the alternate stack.

42004 Use of this function by library threads that are not bound to kernel-scheduled entities results in  
 42005 undefined behavior.

42006 **RETURN VALUE**

42007 Upon successful completion, *sigaltstack()* shall return 0; otherwise, it shall return *-1* and set *errno*  
 42008 to indicate the error.

42009 **ERRORS**42010 The *sigaltstack()* function shall fail if:42011 [EINVAL] The *ss* argument is not a null pointer, and the *ss\_flags* member pointed to by *ss*  
42012 contains flags other than *SS\_DISABLE*.42013 [ENOMEM] The size of the alternate stack area is less than *MINSIGSTKSZ*.

42014 [EPERM] An attempt was made to modify an active stack.

42015 **EXAMPLES**42016 **Allocating Memory for an Alternate Stack**

42017 The following example illustrates a method for allocating memory for an alternate stack.

```

42018 #include <signal.h>
42019 ...
42020 if ((sigstk.ss_sp = malloc(SIGSTKSZ)) == NULL)
42021 /* Error return. */
42022 sigstk.ss_size = SIGSTKSZ;
42023 sigstk.ss_flags = 0;
42024 if (sigaltstack(&sigstk, (stack_t *)0) < 0)
42025 perror("sigaltstack");

```

42026 **APPLICATION USAGE**

42027 On some implementations, stack space is automatically extended as needed. On those  
42028 implementations, automatic extension is typically not available for an alternate stack. If the stack  
42029 overflows, the behavior is undefined.

42030 **RATIONALE**

42031 None.

42032 **FUTURE DIRECTIONS**

42033 None.

42034 **SEE ALSO**

42035 Section 2.4 (on page 28), *sigaction()*, *sigsetjmp()*, the Base Definitions volume of  
42036 IEEE Std 1003.1-2001, <**signal.h**>

42037 **CHANGE HISTORY**

42038 First released in Issue 4, Version 2.

42039 **Issue 5**

42040 Moved from X/OPEN UNIX extension to BASE.

42041 The last sentence of the DESCRIPTION was included as an APPLICATION USAGE note in  
42042 previous issues.

42043 **Issue 6**

42044 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

42045 The **restrict** keyword is added to the *sigaltstack()* prototype for alignment with the  
42046 ISO/IEC 9899:1999 standard.

42047 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/58 is applied, updating the first sentence  
42048 to include “for the current thread”.

42049 **NAME**

42050 sigdelset — delete a signal from a signal set

42051 **SYNOPSIS**

42052 cx #include &lt;signal.h&gt;

42053 int sigdelset(sigset\_t \*set, int signo);

42054

42055 **DESCRIPTION**42056 The *sigdelset()* function deletes the individual signal specified by *signo* from the signal set  
42057 pointed to by *set*.42058 Applications should call either *sigemptyset()* or *sigfillset()* at least once for each object of type  
42059 **sigset\_t** prior to any other use of that object. If such an object is not initialized in this way, but is  
42060 nonetheless supplied as an argument to any of *pthread\_sigmask()*, *sigaction()*, *sigaddset()*,  
42061 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or  
42062 *sigwaitinfo()*, the results are undefined.42063 **RETURN VALUE**42064 Upon successful completion, *sigdelset()* shall return 0; otherwise, it shall return -1 and set *errno*  
42065 to indicate the error.42066 **ERRORS**42067 The *sigdelset()* function may fail if:42068 [EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal  
42069 number.42070 **EXAMPLES**

42071 None.

42072 **APPLICATION USAGE**

42073 None.

42074 **RATIONALE**

42075 None.

42076 **FUTURE DIRECTIONS**

42077 None.

42078 **SEE ALSO**42079 Section 2.4 (on page 28), *sigaction()*, *sigaddset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*,  
42080 *sigpending()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
42081 <signal.h>42082 **CHANGE HISTORY**

42083 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

42084 **Issue 5**42085 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
42086 previous issues.42087 **Issue 6**42088 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an  
42089 extension over the ISO C standard.

42090 **NAME**

42091 sigemptyset — initialize and empty a signal set

42092 **SYNOPSIS**

42093 cx #include &lt;signal.h&gt;

42094 int sigemptyset(sigset\_t \*set);

42095

42096 **DESCRIPTION**42097 The *sigemptyset()* function initializes the signal set pointed to by *set*, such that all signals defined  
42098 in IEEE Std 1003.1-2001 are excluded.42099 **RETURN VALUE**42100 Upon successful completion, *sigemptyset()* shall return 0; otherwise, it shall return -1 and set  
42101 *errno* to indicate the error.42102 **ERRORS**

42103 No errors are defined.

42104 **EXAMPLES**

42105 None.

42106 **APPLICATION USAGE**

42107 None.

42108 **RATIONALE**42109 The implementation of the *sigemptyset()* (or *sigfillset()*) function could quite trivially clear (or  
42110 set) all the bits in the signal set. Alternatively, it would be reasonable to initialize part of the  
42111 structure, such as a version field, to permit binary-compatibility between releases where the size  
42112 of the set varies. For such reasons, either *sigemptyset()* or *sigfillset()* must be called prior to any  
42113 other use of the signal set, even if such use is read-only (for example, as an argument to  
42114 *sigpending()*). This function is not intended for dynamic allocation.42115 The *sigfillset()* and *sigemptyset()* functions require that the resulting signal set include (or  
42116 exclude) all the signals defined in this volume of IEEE Std 1003.1-2001. Although it is outside the  
42117 scope of this volume of IEEE Std 1003.1-2001 to place this requirement on signals that are  
42118 implemented as extensions, it is recommended that implementation-defined signals also be  
42119 affected by these functions. However, there may be a good reason for a particular signal not to  
42120 be affected. For example, blocking or ignoring an implementation-defined signal may have  
42121 undesirable side effects, whereas the default action for that signal is harmless. In such a case, it  
42122 would be preferable for such a signal to be excluded from the signal set returned by *sigfillset()*.42123 In early proposals there was no distinction between invalid and unsupported signals (the names  
42124 of optional signals that were not supported by an implementation were not defined by that  
42125 implementation). The [EINVAL] error was thus specified as a required error for invalid signals.  
42126 With that distinction, it is not necessary to require implementations of these functions to  
42127 determine whether an optional signal is actually supported, as that could have a significant  
42128 performance impact for little value. The error could have been required for invalid signals and  
42129 optional for unsupported signals, but this seemed unnecessarily complex. Thus, the error is  
42130 optional in both cases.42131 **FUTURE DIRECTIONS**

42132 None.

42133 **SEE ALSO**

42134 Section 2.4 (on page 28), *sigaction()*, *sigaddset()*, *sigdelset()*, *sigfillset()*, *sigismember()*, *sigpending()*,  
42135 *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std 1003.1-2001, <signal.h>

42136 **CHANGE HISTORY**

42137 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

42138 **Issue 6**

42139 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an  
42140 extension over the ISO C standard.

42141 **NAME**

42142 sigfillset — initialize and fill a signal set

42143 **SYNOPSIS**42144 **CX** #include <signal.h>

42145 int sigfillset(sigset\_t \*set);

42146

42147 **DESCRIPTION**42148 The *sigfillset()* function shall initialize the signal set pointed to by *set*, such that all signals  
42149 defined in this volume of IEEE Std 1003.1-2001 are included.42150 **RETURN VALUE**42151 Upon successful completion, *sigfillset()* shall return 0; otherwise, it shall return -1 and set *errno*  
42152 to indicate the error.42153 **ERRORS**

42154 No errors are defined.

42155 **EXAMPLES**

42156 None.

42157 **APPLICATION USAGE**

42158 None.

42159 **RATIONALE**42160 Refer to *sigemptyset()* (on page 1354).42161 **FUTURE DIRECTIONS**

42162 None.

42163 **SEE ALSO**42164 Section 2.4 (on page 28), *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigismember()*,  
42165 *sigpending()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
42166 <signal.h>42167 **CHANGE HISTORY**

42168 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

42169 **Issue 6**42170 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an  
42171 extension over the ISO C standard.

42172 **NAME**

42173 sighold, sigignore, sigpause, sigrelse, sigset — signal management

42174 **SYNOPSIS**

```

42175 XSI #include <signal.h>

42176 int sighold(int sig);
42177 int sigignore(int sig);
42178 int sigpause(int sig);
42179 int sigrelse(int sig);
42180 void (*sigset(int sig, void (*disp)(int)))(int);
42181

```

42182 **DESCRIPTION**

42183 Use of any of these functions is unspecified in a multi-threaded process.

42184 The *sighold()*, *sigignore()*, *sigpause()*, *sigrelse()*, and *sigset()* functions provide simplified signal  
42185 management.

42186 The *sigset()* function shall modify signal dispositions. The *sig* argument specifies the signal,  
42187 which may be any signal except SIGKILL and SIGSTOP. The *disp* argument specifies the signal's  
42188 disposition, which may be SIG\_DFL, SIG\_IGN, or the address of a signal handler. If *sigset()* is  
42189 used, and *disp* is the address of a signal handler, the system shall add *sig* to the calling process'  
42190 signal mask before executing the signal handler; when the signal handler returns, the system  
42191 shall restore the calling process' signal mask to its state prior to the delivery of the signal. In  
42192 addition, if *sigset()* is used, and *disp* is equal to SIG\_HOLD, *sig* shall be added to the calling  
42193 process' signal mask and *sig*'s disposition shall remain unchanged. If *sigset()* is used, and *disp*  
42194 is not equal to SIG\_HOLD, *sig* shall be removed from the calling process' signal mask.

42195 The *sighold()* function shall add *sig* to the calling process' signal mask.42196 The *sigrelse()* function shall remove *sig* from the calling process' signal mask.42197 The *sigignore()* function shall set the disposition of *sig* to SIG\_IGN.

42198 The *sigpause()* function shall remove *sig* from the calling process' signal mask and suspend the  
42199 calling process until a signal is received. The *sigpause()* function shall restore the process' signal  
42200 mask to its original state before returning.

42201 If the action for the SIGCHLD signal is set to SIG\_IGN, child processes of the calling processes  
42202 shall not be transformed into zombie processes when they terminate. If the calling process  
42203 subsequently waits for its children, and the process has no unwaited-for children that were  
42204 transformed into zombie processes, it shall block until all of its children terminate, and *wait()*,  
42205 *waitid()*, and *waitpid()* shall fail and set *errno* to [ECHILD].

42206 **RETURN VALUE**

42207 Upon successful completion, *sigset()* shall return SIG\_HOLD if the signal had been blocked and  
42208 the signal's previous disposition if it had not been blocked. Otherwise, SIG\_ERR shall be  
42209 returned and *errno* set to indicate the error.

42210 The *sigpause()* function shall suspend execution of the thread until a signal is received,  
42211 whereupon it shall return -1 and set *errno* to [EINTR].

42212 For all other functions, upon successful completion, 0 shall be returned. Otherwise, -1 shall be  
42213 returned and *errno* set to indicate the error.

42214 **ERRORS**

42215 These functions shall fail if:

42216 [EINVAL] The *sig* argument is an illegal signal number.42217 The *sigset()* and *sigignore()* functions shall fail if:42218 [EINVAL] An attempt is made to catch a signal that cannot be caught, or to ignore a  
42219 signal that cannot be ignored.42220 **EXAMPLES**

42221 None.

42222 **APPLICATION USAGE**42223 The *sigaction()* function provides a more comprehensive and reliable mechanism for controlling  
42224 signals; new applications should use *sigaction()* rather than *sigset()*.42225 The *sighold()* function, in conjunction with *sigrelse()* or *sigpause()*, may be used to establish  
42226 critical regions of code that require the delivery of a signal to be temporarily deferred.42227 The *sigsuspend()* function should be used in preference to *sigpause()* for broader portability.42228 **RATIONALE**

42229 None.

42230 **FUTURE DIRECTIONS**

42231 None.

42232 **SEE ALSO**42233 Section 2.4 (on page 28), *exec*, *pause()*, *sigaction()*, *signal()*, *sigsuspend()*, *waitid()*, the Base  
42234 Definitions volume of IEEE Std 1003.1-2001, <**signal.h**>42235 **CHANGE HISTORY**

42236 First released in Issue 4, Version 2.

42237 **Issue 5**

42238 Moved from X/OPEN UNIX extension to BASE.

42239 The DESCRIPTION is updated to indicate that the *sigpause()* function restores the process'  
42240 signal mask to its original state before returning.42241 The RETURN VALUE section is updated to indicate that the *sigpause()* function suspends  
42242 execution of the process until a signal is received, whereupon it returns -1 and sets *errno* to  
42243 [EINTR].42244 **Issue 6**

42245 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

42246 References to the *wait3()* function are removed.

42247 The XSI functions are split out into their own reference page.

42248 **NAME**

42249 siginterrupt — allow signals to interrupt functions

42250 **SYNOPSIS**

42251 XSI #include &lt;signal.h&gt;

42252 int siginterrupt(int sig, int flag);

42253

42254 **DESCRIPTION**42255 The *siginterrupt()* function shall change the restart behavior when a function is interrupted by  
42256 the specified signal. The function *siginterrupt(sig, flag)* has an effect as if implemented as:

```

42257 int siginterrupt(int sig, int flag) {
42258 int ret;
42259 struct sigaction act;
42260
42261 (void) sigaction(sig, NULL, &act);
42262 if (flag)
42263 act.sa_flags &= ~SA_RESTART;
42264 else
42265 act.sa_flags |= SA_RESTART;
42266 ret = sigaction(sig, &act, NULL);
42267 return ret;
42268 }

```

42268 **RETURN VALUE**42269 Upon successful completion, *siginterrupt()* shall return 0; otherwise, -1 shall be returned and  
42270 *errno* set to indicate the error.42271 **ERRORS**42272 The *siginterrupt()* function shall fail if:42273 [EINVAL] The *sig* argument is not a valid signal number.42274 **EXAMPLES**

42275 None.

42276 **APPLICATION USAGE**42277 The *siginterrupt()* function supports programs written to historical system interfaces. A  
42278 conforming application, when being written or rewritten, should use *sigaction()* with the  
42279 SA\_RESTART flag instead of *siginterrupt()*.42280 **RATIONALE**

42281 None.

42282 **FUTURE DIRECTIONS**

42283 None.

42284 **SEE ALSO**42285 Section 2.4 (on page 28), *sigaction()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
42286 <signal.h>42287 **CHANGE HISTORY**

42288 First released in Issue 4, Version 2.

42289 **Issue 5**

42290 Moved from X/OPEN UNIX extension to BASE.

42291 **Issue 6**

42292 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/59 is applied, correcting the declaration in |  
42293 the sample implementation given in the DESCRIPTION. |

42294 **NAME**

42295 sigismember — test for a signal in a signal set

42296 **SYNOPSIS**42297 **CX** #include <signal.h>

42298 int sigismember(const sigset\_t \*set, int signo);

42299

42300 **DESCRIPTION**42301 The *sigismember()* function shall test whether the signal specified by *signo* is a member of the set  
42302 pointed to by *set*.42303 Applications should call either *sigemptyset()* or *sigfillset()* at least once for each object of type  
42304 **sigset\_t** prior to any other use of that object. If such an object is not initialized in this way, but is  
42305 nonetheless supplied as an argument to any of *pthread\_sigmask()*, *sigaction()*, *sigaddset()*,  
42306 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or  
42307 *sigwaitinfo()*, the results are undefined.42308 **RETURN VALUE**42309 Upon successful completion, *sigismember()* shall return 1 if the specified signal is a member of  
42310 the specified set, or 0 if it is not. Otherwise, it shall return -1 and set *errno* to indicate the error.42311 **ERRORS**42312 The *sigismember()* function may fail if:42313 [EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal  
42314 number.42315 **EXAMPLES**

42316 None.

42317 **APPLICATION USAGE**

42318 None.

42319 **RATIONALE**

42320 None.

42321 **FUTURE DIRECTIONS**

42322 None.

42323 **SEE ALSO**42324 Section 2.4 (on page 28), *sigaction()*, *sigaddset()*, *sigdelset()*, *sigfillset()*, *sigemptyset()*, *sigpending()*,  
42325 *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std 1003.1-2001, <signal.h>42326 **CHANGE HISTORY**

42327 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

42328 **Issue 5**42329 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
42330 previous issues.42331 **Issue 6**42332 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an  
42333 extension over the ISO C standard.

42334 **NAME**

42335 siglongjmp — non-local goto with signal handling

42336 **SYNOPSIS**

42337 CX #include &lt;setjmp.h&gt;

42338 void siglongjmp(sigjmp\_buf env, int val);

42339

42340 **DESCRIPTION**42341 The *siglongjmp()* function shall be equivalent to the *longjmp()* function, except as follows:

- 42342
- References to *setjmp()* shall be equivalent to *sigsetjmp()*.
  - The *siglongjmp()* function shall restore the saved signal mask if and only if the *env* argument was initialized by a call to *sigsetjmp()* with a non-zero *savemask* argument.

42345 **RETURN VALUE**42346 After *siglongjmp()* is completed, program execution shall continue as if the corresponding  
42347 invocation of *sigsetjmp()* had just returned the value specified by *val*. The *siglongjmp()* function  
42348 shall not cause *sigsetjmp()* to return 0; if *val* is 0, *sigsetjmp()* shall return the value 1.42349 **ERRORS**

42350 No errors are defined.

42351 **EXAMPLES**

42352 None.

42353 **APPLICATION USAGE**42354 The distinction between *setjmp()* or *longjmp()* and *sigsetjmp()* or *siglongjmp()* is only significant  
42355 for programs which use *sigaction()*, *sigprocmask()*, or *sigsuspend()*.42356 **RATIONALE**

42357 None.

42358 **FUTURE DIRECTIONS**

42359 None.

42360 **SEE ALSO**42361 *longjmp()*, *setjmp()*, *sigprocmask()*, *sigsetjmp()*, *sigsuspend()*, the Base Definitions volume of  
42362 IEEE Std 1003.1-2001, <setjmp.h>42363 **CHANGE HISTORY**

42364 First released in Issue 3. Included for alignment with the ISO POSIX-1 standard.

42365 **Issue 5**

42366 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

42367 **Issue 6**42368 The DESCRIPTION is rewritten in terms of *longjmp()*.42369 The SYNOPSIS is marked CX since the presence of this function in the <setjmp.h> header is an  
42370 extension over the ISO C standard.

42371 **NAME**

42372        signal — signal management

42373 **SYNOPSIS**

42374        #include &lt;signal.h&gt;

42375        void (\*signal(int *sig*, void (\**func*)(int)))(int);42376 **DESCRIPTION**42377 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
42378 conflict between the requirements described here and the ISO C standard is unintentional. This  
42379 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.42380 **CX**        Use of this function is unspecified in a multi-threaded process.42381        The *signal()* function chooses one of three ways in which receipt of the signal number *sig* is to be  
42382 subsequently handled. If the value of *func* is SIG\_DFL, default handling for that signal shall  
42383 occur. If the value of *func* is SIG\_IGN, the signal shall be ignored. Otherwise, the application  
42384 shall ensure that *func* points to a function to be called when that signal occurs. An invocation of  
42385 such a function because of a signal, or (recursively) of any further functions called by that  
42386 invocation (other than functions in the standard library), is called a “signal handler”.42387        When a signal occurs, and *func* points to a function, it is implementation-defined whether the  
42388 equivalent of a:42389        signal(*sig*, SIG\_DFL);42390        is executed or the implementation prevents some implementation-defined set of signals (at least  
42391 including *sig*) from occurring until the current signal handling has completed. (If the value of *sig*  
42392 is SIGILL, the implementation may alternatively define that no action is taken.) Next the  
42393 equivalent of:42394        (\**func*)(*sig*);42395        is executed. If and when the function returns, if the value of *sig* was SIGFPE, SIGILL, or  
42396 SIGSEGV or any other implementation-defined value corresponding to a computational  
42397 exception, the behavior is undefined. Otherwise, the program shall resume execution at the  
42398 **CX** point it was interrupted. If the signal occurs as the result of calling the *abort()*, *raise()*, *kill()*,  
42399 *pthread\_kill()*, or *sigqueue()* function, the signal handler shall not call the *raise()* function.42400 **CX**        If the signal occurs other than as the result of calling *abort()*, *raise()*, *kill()*, *pthread\_kill()*, or  
42401 *sigqueue()*, the behavior is undefined if the signal handler refers to any object with static storage  
42402 duration other than by assigning a value to an object declared as volatile **sig\_atomic\_t**, or if the  
42403 signal handler calls any function in the standard library other than one of the functions listed in  
42404 Section 2.4 (on page 28). Furthermore, if such a call fails, the value of *errno* is unspecified.

42405        At program start-up, the equivalent of:

42406        signal(*sig*, SIG\_IGN);

42407        is executed for some signals, and the equivalent of:

42408        signal(*sig*, SIG\_DFL);42409 **CX**        is executed for all other signals (see *exec*).42410 **RETURN VALUE**42411        If the request can be honored, *signal()* shall return the value of *func* for the most recent call to  
42412 *signal()* for the specified signal *sig*. Otherwise, SIG\_ERR shall be returned and a positive value  
42413 shall be stored in *errno*.

42414 **ERRORS**

42415 The *signal()* function shall fail if:

42416 CX [EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a  
42417 signal that cannot be caught or ignore a signal that cannot be ignored.

42418 The *signal()* function may fail if:

42419 CX [EINVAL] An attempt was made to set the action to SIG\_DFL for a signal that cannot be  
42420 caught or ignored (or both).

42421 **EXAMPLES**

42422 None.

42423 **APPLICATION USAGE**

42424 The *sigaction()* function provides a more comprehensive and reliable mechanism for controlling  
42425 signals; new applications should use *sigaction()* rather than *signal()*.

42426 **RATIONALE**

42427 None.

42428 **FUTURE DIRECTIONS**

42429 None.

42430 **SEE ALSO**

42431 Section 2.4 (on page 28), *exec*, *pause()*, *sigaction()*, *sigsuspend()*, *waitid()*, the Base Definitions  
42432 volume of IEEE Std 1003.1-2001, <signal.h>

42433 **CHANGE HISTORY**

42434 First released in Issue 1. Derived from Issue 1 of the SVID.

42435 **Issue 5**

42436 Moved from X/OPEN UNIX extension to BASE.

42437 The DESCRIPTION is updated to indicate that the *sigpause()* function restores the process'  
42438 signal mask to its original state before returning.

42439 The RETURN VALUE section is updated to indicate that the *sigpause()* function suspends  
42440 execution of the process until a signal is received, whereupon it returns -1 and sets *errno* to  
42441 [EINTR].

42442 **Issue 6**

42443 Extensions beyond the ISO C standard are marked.

42444 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

42445 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

42446 References to the *wait3()* function are removed.

42447 The *sighold()*, *sigignore()*, *sigrelse()*, and *sigset()* functions are split out onto their own reference  
42448 page.

42449 **NAME**

42450 signbit — test sign

42451 **SYNOPSIS**

42452 #include &lt;math.h&gt;

42453 int signbit(real-floating x);

42454 **DESCRIPTION**

42455 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
42456 conflict between the requirements described here and the ISO C standard is unintentional. This  
42457 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

42458 The *signbit()* macro shall determine whether the sign of its argument value is negative. NaNs,  
42459 zeros, and infinities have a sign bit.

42460 **RETURN VALUE**

42461 The *signbit()* macro shall return a non-zero value if and only if the sign of its argument value is  
42462 negative.

42463 **ERRORS**

42464 No errors are defined.

42465 **EXAMPLES**

42466 None.

42467 **APPLICATION USAGE**

42468 None.

42469 **RATIONALE**

42470 None.

42471 **FUTURE DIRECTIONS**

42472 None.

42473 **SEE ALSO**

42474 *fpclassify()*, *isfinite()*, *isinf()*, *isnan()*, *isnormal()*, the Base Definitions volume of  
42475 IEEE Std 1003.1-2001, <math.h>

42476 **CHANGE HISTORY**

42477 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

42478 **NAME**

42479 sigpause — remove a signal from the signal mask and suspend the thread

42480 **SYNOPSIS**

42481 XSI #include <signal.h>

42482 int sigpause(int sig);

42483

42484 **DESCRIPTION**

42485 Refer to *sighold()*.

42486 **NAME**

42487 sigpending — examine pending signals

42488 **SYNOPSIS**

42489 cx #include &lt;signal.h&gt;

42490 int sigpending(sigset\_t \*set);

42491

42492 **DESCRIPTION**

42493 The *sigpending()* function shall store, in the location referenced by the *set* argument, the set of  
42494 signals that are blocked from delivery to the calling thread and that are pending on the process  
42495 or the calling thread.

42496 **RETURN VALUE**

42497 Upon successful completion, *sigpending()* shall return 0; otherwise, -1 shall be returned and  
42498 *errno* set to indicate the error.

42499 **ERRORS**

42500 No errors are defined.

42501 **EXAMPLES**

42502 None.

42503 **APPLICATION USAGE**

42504 None.

42505 **RATIONALE**

42506 None.

42507 **FUTURE DIRECTIONS**

42508 None.

42509 **SEE ALSO**

42510 *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*, *sigprocmask()*, the Base Definitions  
42511 volume of IEEE Std 1003.1-2001, <**signal.h**>

42512 **CHANGE HISTORY**

42513 First released in Issue 3.

42514 **Issue 5**

42515 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

42516 **Issue 6**

42517 The SYNOPSIS is marked CX since the presence of this function in the <**signal.h**> header is an  
42518 extension over the ISO C standard.

42519 **NAME**

42520 sigprocmask — examine and change blocked signals

42521 **SYNOPSIS**

42522 cx `#include <signal.h>`

42523 `int sigprocmask(int how, const sigset_t *restrict set,`  
42524 `sigset_t *restrict oset);`

42525

42526 **DESCRIPTION**

42527 Refer to *pthread\_sigmask()*.

42528 **NAME**42529 sigqueue — queue a signal to a process (**REALTIME**)42530 **SYNOPSIS**

42531 RTS #include &lt;signal.h&gt;

42532 int sigqueue(pid\_t pid, int signo, const union sigval value);

42533

42534 **DESCRIPTION**

42535 The *sigqueue()* function shall cause the signal specified by *signo* to be sent with the value  
 42536 specified by *value* to the process specified by *pid*. If *signo* is zero (the null signal), error checking  
 42537 is performed but no signal is actually sent. The null signal can be used to check the validity of  
 42538 *pid*.

42539 The conditions required for a process to have permission to queue a signal to another process  
 42540 are the same as for the *kill()* function.

42541 The *sigqueue()* function shall return immediately. If SA\_SIGINFO is set for *signo* and if the  
 42542 resources were available to queue the signal, the signal shall be queued and sent to the receiving  
 42543 process. If SA\_SIGINFO is not set for *signo*, then *signo* shall be sent at least once to the receiving  
 42544 process; it is unspecified whether *value* shall be sent to the receiving process as a result of this  
 42545 call.

42546 If the value of *pid* causes *signo* to be generated for the sending process, and if *signo* is not blocked  
 42547 for the calling thread and if no other thread has *signo* unblocked or is waiting in a *sigwait()*  
 42548 function for *signo*, either *signo* or at least the pending, unblocked signal shall be delivered to the  
 42549 calling thread before the *sigqueue()* function returns. Should any multiple pending signals in the  
 42550 range SIGRTMIN to SIGRTMAX be selected for delivery, it shall be the lowest numbered one.  
 42551 The selection order between realtime and non-realtime signals, or between multiple pending  
 42552 non-realtime signals, is unspecified.

42553 **RETURN VALUE**

42554 Upon successful completion, the specified signal shall have been queued, and the *sigqueue()*  
 42555 function shall return a value of zero. Otherwise, the function shall return a value of -1 and set  
 42556 *errno* to indicate the error.

42557 **ERRORS**42558 The *sigqueue()* function shall fail if:

42559 [EAGAIN] No resources are available to queue the signal. The process has already  
 42560 queued {SIGQUEUE\_MAX} signals that are still pending at the receiver(s), or  
 42561 a system-wide resource limit has been exceeded.

42562 [EINVAL] The value of the *signo* argument is an invalid or unsupported signal number.

42563 [EPERM] The process does not have the appropriate privilege to send the signal to the  
 42564 receiving process.

42565 [ESRCH] The process *pid* does not exist.

42566 **EXAMPLES**

42567 None.

42568 **APPLICATION USAGE**

42569 None.

42570 **RATIONALE**

42571 The *sigqueue()* function allows an application to queue a realtime signal to itself or to another  
42572 process, specifying the application-defined value. This is common practice in realtime  
42573 applications on existing realtime systems. It was felt that specifying another function in the  
42574 *sig...* name space already carved out for signals was preferable to extending the interface to  
42575 *kill()*.

42576 Such a function became necessary when the put/get event function of the message queues was  
42577 removed. It should be noted that the *sigqueue()* function implies reduced performance in a  
42578 security-conscious implementation as the access permissions between the sender and receiver  
42579 have to be checked on each send when the *pid* is resolved into a target process. Such access  
42580 checks were necessary only at message queue open in the previous interface.

42581 The standard developers required that *sigqueue()* have the same semantics with respect to the  
42582 null signal as *kill()*, and that the same permission checking be used. But because of the difficulty  
42583 of implementing the “broadcast” semantic of *kill()* (for example, to process groups) and the  
42584 interaction with resource allocation, this semantic was not adopted. The *sigqueue()* function  
42585 queues a signal to a single process specified by the *pid* argument.

42586 The *sigqueue()* function can fail if the system has insufficient resources to queue the signal. An  
42587 explicit limit on the number of queued signals that a process could send was introduced. While  
42588 the limit is “per-sender”, this volume of IEEE Std 1003.1-2001 does not specify that the resources  
42589 be part of the state of the sender. This would require either that the sender be maintained after  
42590 exit until all signals that it had sent to other processes were handled or that all such signals that  
42591 had not yet been acted upon be removed from the queue(s) of the receivers. This volume of  
42592 IEEE Std 1003.1-2001 does not preclude this behavior, but an implementation that allocated  
42593 queuing resources from a system-wide pool (with per-sender limits) and that leaves queued  
42594 signals pending after the sender exits is also permitted.

42595 **FUTURE DIRECTIONS**

42596 None.

42597 **SEE ALSO**

42598 Section 2.8.1 (on page 41), the Base Definitions volume of IEEE Std 1003.1-2001, &lt;signal.h&gt;

42599 **CHANGE HISTORY**

42600 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
42601 POSIX Threads Extension.

42602 **Issue 6**42603 The *sigqueue()* function is marked as part of the Realtime Signals Extension option.

42604 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
42605 implementation does not support the Realtime Signals Extension option.

42606 **NAME**

42607 sigrelse, sigset — signal management

42608 **SYNOPSIS**42609 XSI `#include <signal.h>`42610 `int sigrelse(int sig);`42611 `void (*sigset(int sig, void (*disp)(int)))(int);`

42612

42613 **DESCRIPTION**42614 Refer to *sighold()*.

42615 **NAME**

42616 sigsetjmp — set jump point for a non-local goto

42617 **SYNOPSIS**42618 `CX` #include <setjmp.h>42619 `int sigsetjmp(sigjmp_buf env, int savemask);`

42620

42621 **DESCRIPTION**42622 The *sigsetjmp()* function shall be equivalent to the *setjmp()* function, except as follows:

- 42623 • References to *setjmp()* are equivalent to *sigsetjmp()*.
- 42624 • References to *longjmp()* are equivalent to *siglongjmp()*.
- 42625 • If the value of the *savemask* argument is not 0, *sigsetjmp()* shall also save the current signal mask of the calling thread as part of the calling environment.

42627 **RETURN VALUE**

42628 If the return is from a successful direct invocation, *sigsetjmp()* shall return 0. If the return is from  
 42629 a call to *siglongjmp()*, *sigsetjmp()* shall return a non-zero value.

42630 **ERRORS**

42631 No errors are defined.

42632 **EXAMPLES**

42633 None.

42634 **APPLICATION USAGE**

42635 The distinction between *setjmp()/longjmp()* and *sigsetjmp()/siglongjmp()* is only significant for  
 42636 programs which use *sigaction()*, *sigprocmask()*, or *sigsuspend()*.

42637 Note that since this function is defined in terms of *setjmp()*, if *savemask* is zero, it is unspecified  
 42638 whether the signal mask is saved.

42639 **RATIONALE**

42640 The ISO C standard specifies various restrictions on the usage of the *setjmp()* macro in order to  
 42641 permit implementors to recognize the name in the compiler and not implement an actual  
 42642 function. These same restrictions apply to the *sigsetjmp()* macro.

42643 There are processors that cannot easily support these calls, but this was not considered a  
 42644 sufficient reason to exclude them.

42645 4.2 BSD, 4.3 BSD, and XSI-conformant systems provide functions named *\_setjmp()* and  
 42646 *\_longjmp()* that, together with *setjmp()* and *longjmp()*, provide the same functionality as  
 42647 *sigsetjmp()* and *siglongjmp()*. On those systems, *setjmp()* and *longjmp()* save and restore signal  
 42648 masks, while *\_setjmp()* and *\_longjmp()* do not. On System V Release 3 and in corresponding  
 42649 issues of the SVID, *setjmp()* and *longjmp()* are explicitly defined not to save and restore signal  
 42650 masks. In order to permit existing practice in both cases, the relation of *setjmp()* and *longjmp()* to  
 42651 signal masks is not specified, and a new set of functions is defined instead.

42652 The *longjmp()* and *siglongjmp()* functions operate as in the previous issue provided the matching  
 42653 *setjmp()* or *sigsetjmp()* has been performed in the same thread. Non-local jumps into contexts  
 42654 saved by other threads would be at best a questionable practice and were not considered worthy  
 42655 of standardization.

42656 **FUTURE DIRECTIONS**

42657           None.

42658 **SEE ALSO**42659           *siglongjmp()*, *signal()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of  
42660           IEEE Std 1003.1-2001, <**setjmp.h**>42661 **CHANGE HISTORY**

42662           First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

42663 **Issue 5**

42664           The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

42665 **Issue 6**42666           The DESCRIPTION is reworded in terms of *setjmp()*.42667           The SYNOPSIS is marked CX since the presence of this function in the <**setjmp.h**> header is an  
42668           extension over the ISO C standard.

42669 **NAME**

42670 sigsuspend — wait for a signal

42671 **SYNOPSIS**42672 cx `#include <signal.h>`42673 `int sigsuspend(const sigset_t *sigmask);`

42674

42675 **DESCRIPTION**

42676 The *sigsuspend()* function shall replace the current signal mask of the calling thread with the set  
42677 of signals pointed to by *sigmask* and then suspend the thread until delivery of a signal whose  
42678 action is either to execute a signal-catching function or to terminate the process. This shall not  
42679 cause any other signals that may have been pending on the process to become pending on the  
42680 thread.

42681 If the action is to terminate the process then *sigsuspend()* shall never return. If the action is to  
42682 execute a signal-catching function, then *sigsuspend()* shall return after the signal-catching  
42683 function returns, with the signal mask restored to the set that existed prior to the *sigsuspend()*  
42684 call.

42685 It is not possible to block signals that cannot be ignored. This is enforced by the system without  
42686 causing an error to be indicated.

42687 **RETURN VALUE**

42688 Since *sigsuspend()* suspends thread execution indefinitely, there is no successful completion  
42689 return value. If a return occurs, `-1` shall be returned and *errno* set to indicate the error.

42690 **ERRORS**42691 The *sigsuspend()* function shall fail if:

42692 [EINTR] A signal is caught by the calling process and control is returned from the  
42693 signal-catching function.

42694 **EXAMPLES**

42695 None.

42696 **APPLICATION USAGE**

42697 Normally, at the beginning of a critical code section, a specified set of signals is blocked using  
42698 the *sigprocmask()* function. When the thread has completed the critical section and needs to wait  
42699 for the previously blocked signal(s), it pauses by calling *sigsuspend()* with the mask that was  
42700 returned by the *sigprocmask()* call.

42701 **RATIONALE**

42702 None.

42703 **FUTURE DIRECTIONS**

42704 None.

42705 **SEE ALSO**

42706 Section 2.4 (on page 28), *pause()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, the  
42707 Base Definitions volume of IEEE Std 1003.1-2001, `<signal.h>`

42708 **CHANGE HISTORY**

42709 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

42710 **Issue 5**

42711 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

42712 **Issue 6**

42713 The text in the RETURN VALUE section has been changed from “suspends process execution”  
42714 to “suspends thread execution”. This reflects IEEE PASC Interpretation 1003.1c #40.

42715 Text in the APPLICATION USAGE section has been replaced.

42716 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an  
42717 extension over the ISO C standard.

## 42718 NAME

42719 sigtimedwait, sigwaitinfo — wait for queued signals (**REALTIME**)

## 42720 SYNOPSIS

42721 RTS 

```
#include <signal.h>
```

```
42722 int sigtimedwait(const sigset_t *restrict set,
42723 siginfo_t *restrict info,
42724 const struct timespec *restrict timeout);
42725 int sigwaitinfo(const sigset_t *restrict set,
42726 siginfo_t *restrict info);
42727
```

## 42728 DESCRIPTION

42729 The *sigtimedwait()* function shall be equivalent to *sigwaitinfo()* except that if none of the signals  
 42730 specified by *set* are pending, *sigtimedwait()* shall wait for the time interval specified in the  
 42731 **timespec** structure referenced by *timeout*. If the **timespec** structure pointed to by *timeout* is  
 42732 zero-valued and if none of the signals specified by *set* are pending, then *sigtimedwait()* shall  
 42733 MON return immediately with an error. If *timeout* is the NULL pointer, the behavior is unspecified. If  
 42734 the Monotonic Clock option is supported, the CLOCK\_MONOTONIC clock shall be used to  
 42735 measure the time interval specified by the *timeout* argument.

42736 The *sigwaitinfo()* function selects the pending signal from the set specified by *set*. Should any of  
 42737 multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it shall be the  
 42738 lowest numbered one. The selection order between realtime and non-realtime signals, or  
 42739 between multiple pending non-realtime signals, is unspecified. If no signal in *set* is pending at  
 42740 the time of the call, the calling thread shall be suspended until one or more signals in *set* become  
 42741 pending or until it is interrupted by an unblocked, caught signal.

42742 The *sigwaitinfo()* function shall be equivalent to the *sigwait()* function if the *info* argument is  
 42743 NULL. If the *info* argument is non-NULL, the *sigwaitinfo()* function shall be equivalent to  
 42744 *sigwait()*, except that the selected signal number shall be stored in the *si\_signo* member, and the  
 42745 cause of the signal shall be stored in the *si\_code* member. If any value is queued to the selected  
 42746 signal, the first such queued value shall be dequeued and, if the *info* argument is non-NULL, the  
 42747 value shall be stored in the *si\_value* member of *info*. The system resource used to queue the  
 42748 signal shall be released and returned to the system for other use. If no value is queued, the  
 42749 content of the *si\_value* member is undefined. If no further signals are queued for the selected  
 42750 signal, the pending indication for that signal shall be reset.

## 42751 RETURN VALUE

42752 Upon successful completion (that is, one of the signals specified by *set* is pending or is  
 42753 generated) *sigwaitinfo()* and *sigtimedwait()* shall return the selected signal number. Otherwise,  
 42754 the function shall return a value of  $-1$  and set *errno* to indicate the error.

## 42755 ERRORS

42756 The *sigtimedwait()* function shall fail if:42757 [EAGAIN] No signal specified by *set* was generated within the specified timeout period.42758 The *sigtimedwait()* and *sigwaitinfo()* functions may fail if:

42759 [EINTR] The wait was interrupted by an unblocked, caught signal. It shall be  
 42760 documented in system documentation whether this error causes these  
 42761 functions to fail.

- 42762 The *sigtimedwait()* function may also fail if:
- 42763 [EINVAL] The *timeout* argument specified a *tv\_nsec* value less than zero or greater than  
42764 or equal to 1 000 million.
- 42765 An implementation only checks for this error if no signal is pending in *set* and it is necessary to  
42766 wait.
- 42767 **EXAMPLES**
- 42768 None.
- 42769 **APPLICATION USAGE**
- 42770 The *sigtimedwait()* function times out and returns an [EAGAIN] error. Application writers  
42771 should note that this is inconsistent with other functions such as *pthread\_cond\_timedwait()* that  
42772 return [ETIMEDOUT].
- 42773 **RATIONALE**
- 42774 Existing programming practice on realtime systems uses the ability to pause waiting for a  
42775 selected set of events and handle the first event that occurs in-line instead of in a signal-handling  
42776 function. This allows applications to be written in an event-directed style similar to a state  
42777 machine. This style of programming is useful for largescale transaction processing in which the  
42778 overall throughput of an application and the ability to clearly track states are more important  
42779 than the ability to minimize the response time of individual event handling.
- 42780 It is possible to construct a signal-waiting macro function out of the realtime signal function  
42781 mechanism defined in this volume of IEEE Std 1003.1-2001. However, such a macro has to  
42782 include the definition of a generalized handler for all signals to be waited on. A significant  
42783 portion of the overhead of handler processing can be avoided if the signal-waiting function is  
42784 provided by the kernel. This volume of IEEE Std 1003.1-2001 therefore provides two signal-  
42785 waiting functions—one that waits indefinitely and one with a timeout—as part of the overall  
42786 realtime signal function specification.
- 42787 The specification of a function with a timeout allows an application to be written that can be  
42788 broken out of a wait after a set period of time if no event has occurred. It was argued that setting  
42789 a timer event before the wait and recognizing the timer event in the wait would also implement  
42790 the same functionality, but at a lower performance level. Because of the performance  
42791 degradation associated with the user-level specification of a timer event and the subsequent  
42792 cancellation of that timer event after the wait completes for a valid event, and the complexity  
42793 associated with handling potential race conditions associated with the user-level method, the  
42794 separate function has been included.
- 42795 Note that the semantics of the *sigwaitinfo()* function are nearly identical to that of the *sigwait()*  
42796 function defined by this volume of IEEE Std 1003.1-2001. The only difference is that *sigwaitinfo()*  
42797 returns the queued signal value in the *value* argument. The return of the queued value is  
42798 required so that applications can differentiate between multiple events queued to the same  
42799 signal number.
- 42800 The two distinct functions are being maintained because some implementations may choose to  
42801 implement the POSIX Threads Extension functions and not implement the queued signals  
42802 extensions. Note, though, that *sigwaitinfo()* does not return the queued value if the *value*  
42803 argument is NULL, so the POSIX Threads Extension *sigwait()* function can be implemented as a  
42804 macro on *sigwaitinfo()*.
- 42805 The *sigtimedwait()* function was separated from the *sigwaitinfo()* function to address concerns  
42806 regarding the overloading of the *timeout* pointer to indicate indefinite wait (no timeout), timed  
42807 wait, and immediate return, and concerns regarding consistency with other functions where the  
42808 conditional and timed waits were separate functions from the pure blocking function. The

42809 semantics of *sigtimedwait()* are specified such that *sigwaitinfo()* could be implemented as a  
42810 macro with a NULL pointer for *timeout*.

42811 The *sigwait* functions provide a synchronous mechanism for threads to wait for  
42812 asynchronously-generated signals. One important question was how many threads that are  
42813 suspended in a call to a *sigwait()* function for a signal should return from the call when the  
42814 signal is sent. Four choices were considered:

- 42815 1. Return an error for multiple simultaneous calls to *sigwait* functions for the same signal.
- 42816 2. One or more threads return.
- 42817 3. All waiting threads return.
- 42818 4. Exactly one thread returns.

42819 Prohibiting multiple calls to *sigwait()* for the same signal was felt to be overly restrictive. The  
42820 “one or more” behavior made implementation of conforming packages easy at the expense of  
42821 forcing POSIX threads clients to protect against multiple simultaneous calls to *sigwait()* in  
42822 application code in order to achieve predictable behavior. There was concern that the “all  
42823 waiting threads” behavior would result in “signal broadcast storms”, consuming excessive CPU  
42824 resources by replicating the signals in the general case. Furthermore, no convincing examples  
42825 could be presented that delivery to all was either simpler or more powerful than delivery to one.

42826 Thus, the consensus was that exactly one thread that was suspended in a call to a *sigwait*  
42827 function for a signal should return when that signal occurs. This is not an onerous restriction as:

- 42828 • A multi-way signal wait can be built from the single-way wait.
- 42829 • Signals should only be handled by application-level code, as library routines cannot guess  
42830 what the application wants to do with signals generated for the entire process.
- 42831 • Applications can thus arrange for a single thread to wait for any given signal and call any  
42832 needed routines upon its arrival.

42833 In an application that is using signals for interprocess communication, signal processing is  
42834 typically done in one place. Alternatively, if the signal is being caught so that process cleanup  
42835 can be done, the signal handler thread can call separate process cleanup routines for each  
42836 portion of the application. Since the application main line started each portion of the application,  
42837 it is at the right abstraction level to tell each portion of the application to clean up.

42838 Certainly, there exist programming styles where it is logical to consider waiting for a single  
42839 signal in multiple threads. A simple *sigwait\_multiple()* routine can be constructed to achieve this  
42840 goal. A possible implementation would be to have each *sigwait\_multiple()* caller registered as  
42841 having expressed interest in a set of signals. The caller then waits on a thread-specific condition  
42842 variable. A single server thread calls a *sigwait()* function on the union of all registered signals.  
42843 When the *sigwait()* function returns, the appropriate state is set and condition variables are  
42844 broadcast. New *sigwait\_multiple()* callers may cause the pending *sigwait()* call to be canceled  
42845 and reissued in order to update the set of signals being waited for.

#### 42846 FUTURE DIRECTIONS

42847 None.

#### 42848 SEE ALSO

42849 Section 2.8.1 (on page 41), *pause()*, *pthread\_sigmask()*, *sigaction()*, *sigpending()*, *sigsuspend()*,  
42850 *sigwait()*, the Base Definitions volume of IEEE Std 1003.1-2001, <signal.h>, <time.h>

42851 **CHANGE HISTORY**

42852 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
42853 POSIX Threads Extension.

42854 **Issue 6**

42855 These functions are marked as part of the Realtime Signals Extension option.

42856 The Open Group Corrigendum U035/3 is applied. The SYNOPSIS of the *sigwaitinfo()* function  
42857 has been corrected so that the second argument is of type **siginfo\_t** \*.

42858 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
42859 implementation does not support the Realtime Signals Extension option.

42860 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the  
42861 CLOCK\_MONOTONIC clock, if supported, is used to measure timeout intervals.

42862 The **restrict** keyword is added to the *sigtimedwait()* and *sigwaitinfo()* prototypes for alignment  
42863 with the ISO/IEC 9899:1999 standard.

42864 **NAME**

42865 sigwait — wait for queued signals

42866 **SYNOPSIS**42867 `CX` #include <signal.h>

42868 int sigwait(const sigset\_t \*restrict set, int \*restrict sig);

42869

42870 **DESCRIPTION**

42871 The *sigwait()* function shall select a pending signal from *set*, atomically clear it from the system's  
 42872 set of pending signals, and return that signal number in the location referenced by *sig*. If prior to  
 42873 the call to *sigwait()* there are multiple pending instances of a single signal number, it is  
 42874 implementation-defined whether upon successful return there are any remaining pending  
 42875 signals for that signal number. If the implementation supports queued signals and there are  
 42876 multiple signals queued for the signal number selected, the first such queued signal shall cause a  
 42877 return from *sigwait()* and the remainder shall remain queued. If no signal in *set* is pending at the  
 42878 time of the call, the thread shall be suspended until one or more becomes pending. The signals  
 42879 defined by *set* shall have been blocked at the time of the call to *sigwait()*; otherwise, the behavior  
 42880 is undefined. The effect of *sigwait()* on the signal actions for the signals in *set* is unspecified.

42881 If more than one thread is using *sigwait()* to wait for the same signal, no more than one of these  
 42882 threads shall return from *sigwait()* with the signal number. Which thread returns from *sigwait()*  
 42883 if more than a single thread is waiting is unspecified.

42884 `RTS` Should any of the multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it  
 42885 shall be the lowest numbered one. The selection order between realtime and non-realtime  
 42886 signals, or between multiple pending non-realtime signals, is unspecified.

42887 **RETURN VALUE**

42888 Upon successful completion, *sigwait()* shall store the signal number of the received signal at the  
 42889 location referenced by *sig* and return zero. Otherwise, an error number shall be returned to  
 42890 indicate the error.

42891 **ERRORS**42892 The *sigwait()* function may fail if:42893 [EINVAL] The *set* argument contains an invalid or unsupported signal number.42894 **EXAMPLES**

42895 None.

42896 **APPLICATION USAGE**

42897 None.

42898 **RATIONALE**

42899 To provide a convenient way for a thread to wait for a signal, this volume of  
 42900 IEEE Std 1003.1-2001 provides the *sigwait()* function. For most cases where a thread has to wait  
 42901 for a signal, the *sigwait()* function should be quite convenient, efficient, and adequate.

42902 However, requests were made for a lower-level primitive than *sigwait()* and for semaphores that  
 42903 could be used by threads. After some consideration, threads were allowed to use semaphores  
 42904 and *sem\_post()* was defined to be async-signal and async-cancel-safe.

42905 In summary, when it is necessary for code run in response to an asynchronous signal to notify a  
 42906 thread, *sigwait()* should be used to handle the signal. Alternatively, if the implementation  
 42907 provides semaphores, they also can be used, either following *sigwait()* or from within a signal  
 42908 handling routine previously registered with *sigaction()*.

42909 **FUTURE DIRECTIONS**

42910 None.

42911 **SEE ALSO**

42912 Section 2.4 (on page 28), Section 2.8.1 (on page 41), *pause()*, *pthread\_sigmask()*, *sigaction()*,  
42913 *sigpending()*, *sigsuspend()*, *sigwaitinfo()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
42914 <**signal.h**>, <**time.h**>

42915 **CHANGE HISTORY**

42916 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
42917 POSIX Threads Extension.

42918 **Issue 6**

42919 The **restrict** keyword is added to the *sigwait()* prototype for alignment with the  
42920 ISO/IEC 9899:1999 standard.

42921 **NAME**

42922 sigwaitinfo — wait for queued signals (**REALTIME**)

42923 **SYNOPSIS**

42924 RTS `#include <signal.h>`

42925 `int sigwaitinfo(const sigset_t *restrict set, siginfo_t *restrict info);`

42926

42927 **DESCRIPTION**

42928 Refer to *sigtimedwait()*.

42929 **NAME**

42930 sin, sinf, sinl — sine function

42931 **SYNOPSIS**

42932 #include &lt;math.h&gt;

42933 double sin(double x);

42934 float sinf(float x);

42935 long double sinl(long double x);

42936 **DESCRIPTION**

42937 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

42940 These functions shall compute the sine of their argument *x*, measured in radians.

42941 An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-zero, an error has occurred.

42945 **RETURN VALUE**42946 Upon successful completion, these functions shall return the sine of *x*.42947 **MX** If *x* is NaN, a NaN shall be returned.42948 If *x* is  $\pm 0$ , *x* shall be returned.42949 If *x* is subnormal, a range error may occur and *x* should be returned.

42950 If *x* is  $\pm\text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

42952 **ERRORS**

42953 These functions shall fail if:

42954 **MX** **Domain Error** The *x* argument is  $\pm\text{Inf}$ .

42955 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

42959 These functions may fail if:

42960 **MX** **Range Error** The value of *x* is subnormal

42961 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

42965 **EXAMPLES**42966 **Taking the Sine of a 45-Degree Angle**

```
42967 #include <math.h>
42968 ...
42969 double radians = 45.0 * M_PI / 180;
42970 double result;
42971 ...
42972 result = sin(radians);
```

42973 **APPLICATION USAGE**

42974 These functions may lose accuracy when their argument is near a multiple of  $\pi$  or is far from 0.0.

42975 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
42976 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

42977 **RATIONALE**

42978 None.

42979 **FUTURE DIRECTIONS**

42980 None.

42981 **SEE ALSO**

42982 *asin()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
42983 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

42984 **CHANGE HISTORY**

42985 First released in Issue 1. Derived from Issue 1 of the SVID.

42986 **Issue 5**

42987 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes  
42988 in previous issues.

42989 **Issue 6**

42990 The *sinf()* and *sinl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

42991 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
42992 revised to align with the ISO/IEC 9899:1999 standard.

42993 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
42994 marked.

42995 **NAME**

42996       sinh, sinhf, sinhl — hyperbolic sine functions

42997 **SYNOPSIS**

42998       #include &lt;math.h&gt;

42999       double sinh(double x);

43000       float sinhf(float x);

43001       long double sinhl(long double x);

43002 **DESCRIPTION**43003 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
43004 conflict between the requirements described here and the ISO C standard is unintentional. This  
43005 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.43006       These functions shall compute the hyperbolic sine of their argument *x*.43007       An application wishing to check for error situations should set *errno* to zero and call  
43008 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
43009 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
43010 zero, an error has occurred.43011 **RETURN VALUE**43012       Upon successful completion, these functions shall return the hyperbolic sine of *x*.43013       If the result would cause an overflow, a range error shall occur and  $\pm$ HUGE\_VAL,  
43014  $\pm$ HUGE\_VALF, and  $\pm$ HUGE\_VALL (with the same sign as *x*) shall be returned as appropriate for  
43015 the type of the function.43016 **MX**       If *x* is NaN, a NaN shall be returned.43017       If *x* is  $\pm 0$  or  $\pm$ Inf, *x* shall be returned.43018       If *x* is subnormal, a range error may occur and *x* should be returned.43019 **ERRORS**

43020       These functions shall fail if:

43021       Range Error       The result would cause an overflow.

43022               If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
43023 then *errno* shall be set to [ERANGE]. If the integer expression  
43024 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow  
43025 floating-point exception shall be raised.

43026       These functions may fail if:

43027 **MX**       Range Error       The value *x* is subnormal.43028               If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
43029 then *errno* shall be set to [ERANGE]. If the integer expression  
43030 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow  
43031 floating-point exception shall be raised.

43032 **EXAMPLES**

43033 None.

43034 **APPLICATION USAGE**

43035 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
43036 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

43037 **RATIONALE**

43038 None.

43039 **FUTURE DIRECTIONS**

43040 None.

43041 **SEE ALSO**

43042 *asinh()*, *cosh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tanh()*, the Base Definitions volume of  
43043 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,  
43044 <math.h>

43045 **CHANGE HISTORY**

43046 First released in Issue 1. Derived from Issue 1 of the SVID.

43047 **Issue 5**

43048 The DESCRIPTION is updated to indicate how an application should check for an error. This  
43049 text was previously published in the APPLICATION USAGE section.

43050 **Issue 6**43051 The *sinhf()* and *sinhl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

43052 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
43053 revised to align with the ISO/IEC 9899:1999 standard.

43054 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
43055 marked.

43056 **NAME**

43057       sinl — sine function

43058 **SYNOPSIS**

43059       #include &lt;math.h&gt;

43060       long double sinl(long double x);

43061 **DESCRIPTION**43062       Refer to *sin()*.

43063 **NAME**

43064 sleep — suspend execution for an interval of time

43065 **SYNOPSIS**

43066 #include &lt;unistd.h&gt;

43067 unsigned sleep(unsigned *seconds*);43068 **DESCRIPTION**

43069 The *sleep()* function shall cause the calling thread to be suspended from execution until either  
43070 the number of realtime seconds specified by the argument *seconds* has elapsed or a signal is  
43071 delivered to the calling thread and its action is to invoke a signal-catching function or to  
43072 terminate the process. The suspension time may be longer than requested due to the scheduling  
43073 of other activity by the system.

43074 If a SIGALRM signal is generated for the calling process during execution of *sleep()* and if the  
43075 SIGALRM signal is being ignored or blocked from delivery, it is unspecified whether *sleep()*  
43076 returns when the SIGALRM signal is scheduled. If the signal is being blocked, it is also  
43077 unspecified whether it remains pending after *sleep()* returns or it is discarded.

43078 If a SIGALRM signal is generated for the calling process during execution of *sleep()*, except as a  
43079 result of a prior call to *alarm()*, and if the SIGALRM signal is not being ignored or blocked from  
43080 delivery, it is unspecified whether that signal has any effect other than causing *sleep()* to return.

43081 If a signal-catching function interrupts *sleep()* and examines or changes either the time a  
43082 SIGALRM is scheduled to be generated, the action associated with the SIGALRM signal, or  
43083 whether the SIGALRM signal is blocked from delivery, the results are unspecified.

43084 If a signal-catching function interrupts *sleep()* and calls *siglongjmp()* or *longjmp()* to restore an  
43085 environment saved prior to the *sleep()* call, the action associated with the SIGALRM signal and  
43086 the time at which a SIGALRM signal is scheduled to be generated are unspecified. It is also  
43087 unspecified whether the SIGALRM signal is blocked, unless the process' signal mask is restored  
43088 as part of the environment.

43089 XSI Interactions between *sleep()* and any of *setitimer()*, *ualarm()*, or *usleep()* are unspecified.

43090 **RETURN VALUE**

43091 If *sleep()* returns because the requested time has elapsed, the value returned shall be 0. If *sleep()*  
43092 returns due to delivery of a signal, the return value shall be the “unslept” amount (the requested  
43093 time minus the time actually slept) in seconds.

43094 **ERRORS**

43095 No errors are defined.

43096 **EXAMPLES**

43097 None.

43098 **APPLICATION USAGE**

43099 None.

43100 **RATIONALE**

43101 There are two general approaches to the implementation of the *sleep()* function. One is to use the  
43102 *alarm()* function to schedule a SIGALRM signal and then suspend the process waiting for that  
43103 signal. The other is to implement an independent facility. This volume of IEEE Std 1003.1-2001  
43104 permits either approach.

43105 In order to comply with the requirement that no primitive shall change a process attribute unless  
43106 explicitly described by this volume of IEEE Std 1003.1-2001, an implementation using SIGALRM  
43107 must carefully take into account any SIGALRM signal scheduled by previous *alarm()* calls, the

43108 action previously established for SIGALRM, and whether SIGALRM was blocked. If a SIGALRM  
43109 has been scheduled before the *sleep()* would ordinarily complete, the *sleep()* must be shortened  
43110 to that time and a SIGALRM generated (possibly simulated by direct invocation of the signal-  
43111 catching function) before *sleep()* returns. If a SIGALRM has been scheduled after the *sleep()*  
43112 would ordinarily complete, it must be rescheduled for the same time before *sleep()* returns. The  
43113 action and blocking for SIGALRM must be saved and restored.

43114 Historical implementations often implement the SIGALRM-based version using *alarm()* and  
43115 *pause()*. One such implementation is prone to infinite hangups, as described in *pause()*. Another  
43116 such implementation uses the C-language *setjmp()* and *longjmp()* functions to avoid that  
43117 window. That implementation introduces a different problem: when the SIGALRM signal  
43118 interrupts a signal-catching function installed by the user to catch a different signal, the  
43119 *longjmp()* aborts that signal-catching function. An implementation based on *sigprocmask()*,  
43120 *alarm()*, and *sigsuspend()* can avoid these problems.

43121 Despite all reasonable care, there are several very subtle, but detectable and unavoidable,  
43122 differences between the two types of implementations. These are the cases mentioned in this  
43123 volume of IEEE Std 1003.1-2001 where some other activity relating to SIGALRM takes place, and  
43124 the results are stated to be unspecified. All of these cases are sufficiently unusual as not to be of  
43125 concern to most applications.

43126 See also the discussion of the term *realtime* in *alarm()*.

43127 Since *sleep()* can be implemented using *alarm()*, the discussion about alarms occurring early  
43128 under *alarm()* applies to *sleep()* as well.

43129 Application writers should note that the type of the argument *seconds* and the return value of  
43130 *sleep()* is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces Application  
43131 cannot pass a value greater than the minimum guaranteed value for {UINT\_MAX}, which the  
43132 ISO C standard sets as 65 535, and any application passing a larger value is restricting its  
43133 portability. A different type was considered, but historical implementations, including those  
43134 with a 16-bit **int** type, consistently use either **unsigned** or **int**.

43135 Scheduling delays may cause the process to return from the *sleep()* function significantly after  
43136 the requested time. In such cases, the return value should be set to zero, since the formula  
43137 (requested time minus the time actually spent) yields a negative number and *sleep()* returns an  
43138 **unsigned**.

#### 43139 FUTURE DIRECTIONS

43140 None.

#### 43141 SEE ALSO

43142 *alarm()*, *getitimer()*, *nanosleep()*, *pause()*, *sigaction()*, *sigsetjmp()*, *ualarm()*, *usleep()*, the Base  
43143 Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

#### 43144 CHANGE HISTORY

43145 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 43146 Issue 5

43147 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

43148 **NAME**

43149        **snprintf** — print formatted output

43150 **SYNOPSIS**

43151        #include <stdio.h>

43152        int snprintf(char \*restrict *s*, size\_t *n*,

43153            const char \*restrict *format*, ...);

43154 **DESCRIPTION**

43155        Refer to *fprintf*().

43156 **NAME**

43157 socketmark — determine whether a socket is at the out-of-band mark

43158 **SYNOPSIS**

43159 #include &lt;sys/socket.h&gt;

43160 int socketmark(int s);

43161 **DESCRIPTION**

43162 The *socketmark()* function shall determine whether the socket specified by the descriptor *s* is at  
 43163 the out-of-band data mark (see the System Interfaces volume of IEEE Std 1003.1-2001, Section  
 43164 2.10.12, Socket Out-of-Band Data State). If the protocol for the socket supports out-of-band data  
 43165 by marking the stream with an out-of-band data mark, the *socketmark()* function shall return 1  
 43166 when all data preceding the mark has been read and the out-of-band data mark is the first  
 43167 element in the receive queue. The *socketmark()* function shall not remove the mark from the  
 43168 stream.

43169 **RETURN VALUE**

43170 Upon successful completion, the *socketmark()* function shall return a value indicating whether  
 43171 the socket is at an out-of-band data mark. If the protocol has marked the data stream and all data  
 43172 preceding the mark has been read, the return value shall be 1; if there is no mark, or if data  
 43173 precedes the mark in the receive queue, the *socketmark()* function shall return 0. Otherwise, it  
 43174 shall return a value of  $-1$  and set *errno* to indicate the error.

43175 **ERRORS**43176 The *socketmark()* function shall fail if:43177 [EBADF] The *s* argument is not a valid file descriptor.43178 [ENOTTY] The *s* argument does not specify a descriptor for a socket.43179 **EXAMPLES**

43180 None.

43181 **APPLICATION USAGE**

43182 The use of this function between receive operations allows an application to determine which  
 43183 received data precedes the out-of-band data and which follows the out-of-band data.

43184 There is an inherent race condition in the use of this function. On an empty receive queue, the  
 43185 current read of the location might well be at the “mark”, but the system has no way of knowing  
 43186 that the next data segment that will arrive from the network will carry the mark, and  
 43187 *socketmark()* will return false, and the next read operation will silently consume the mark.

43188 Hence, this function can only be used reliably when the application already knows that the out-  
 43189 of-band data has been seen by the system or that it is known that there is data waiting to be read  
 43190 at the socket (via SIGURG or *select()*). See Section 2.10.11 (on page 61), Section 2.10.12 (on page  
 43191 61), Section 2.10.14 (on page 62), and *pselect()* for details.

43192 **RATIONALE**

43193 The *socketmark()* function replaces the historical SIOCATMARK command to *ioctl()* which  
 43194 implemented the same functionality on many implementations. Using a wrapper function  
 43195 follows the adopted conventions to avoid specifying commands to the *ioctl()* function, other  
 43196 than those now included to support XSI STREAMS. The *socketmark()* function could be  
 43197 implemented as follows:

43198 #include &lt;sys/ioctl.h&gt;

43199 int socketmark(int s)

43200 {

```
43201 int val;
43202 if (ioctl(s, SIOCATMARK, &val) == -1)
43203 return(-1);
43204 return(val);
43205 }
```

43206 The use of [ENOTTY] to indicate an incorrect descriptor type matches the historical behavior of  
43207 SIOCATMARK.

43208 **FUTURE DIRECTIONS**

43209 None.

43210 **SEE ALSO**

43211 *pselect()*, *recv()*, *recvmsg()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/socket.h>

43212 **CHANGE HISTORY**

43213 First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

43214 **NAME**

43215 socket — create an endpoint for communication

43216 **SYNOPSIS**

43217 #include &lt;sys/socket.h&gt;

43218 int socket(int *domain*, int *type*, int *protocol*);43219 **DESCRIPTION**43220 The *socket()* function shall create an unbound socket in a communications domain, and return a  
43221 file descriptor that can be used in later function calls that operate on sockets.43222 The *socket()* function takes the following arguments:43223 *domain* Specifies the communications domain in which a socket is to be created.43224 *type* Specifies the type of socket to be created.43225 *protocol* Specifies a particular protocol to be used with the socket. Specifying a *protocol*  
43226 of 0 causes *socket()* to use an unspecified default protocol appropriate for the  
43227 requested socket type.43228 The *domain* argument specifies the address family used in the communications domain. The  
43229 address families supported by the system are implementation-defined.43230 Symbolic constants that can be used for the domain argument are defined in the <sys/socket.h>  
43231 header.43232 The *type* argument specifies the socket type, which determines the semantics of communication  
43233 over the socket. The following socket types are defined; implementations may specify additional  
43234 socket types:43235 SOCK\_STREAM Provides sequenced, reliable, bidirectional, connection-mode byte  
43236 streams, and may provide a transmission mechanism for out-of-band  
43237 data.43238 SOCK\_DGRAM Provides datagrams, which are connectionless-mode, unreliable messages  
43239 of fixed maximum length.43240 SOCK\_SEQPACKET Provides sequenced, reliable, bidirectional, connection-mode  
43241 transmission paths for records. A record can be sent using one or more  
43242 output operations and received using one or more input operations, but a  
43243 single operation never transfers part of more than one record. Record  
43244 boundaries are visible to the receiver via the MSG\_EOR flag.43245 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address  
43246 family. If the *protocol* argument is zero, the default protocol for this address family and type shall  
43247 be used. The protocols supported by the system are implementation-defined.43248 The process may need to have appropriate privileges to use the *socket()* function or to create  
43249 some sockets.43250 **RETURN VALUE**43251 Upon successful completion, *socket()* shall return a non-negative integer, the socket file  
43252 descriptor. Otherwise, a value of -1 shall be returned and *errno* set to indicate the error.43253 **ERRORS**43254 The *socket()* function shall fail if:

43255 [EAFNOSUPPORT]

43256 The implementation does not support the specified address family.

- 43257 [EMFILE] No more file descriptors are available for this process.
- 43258 [ENFILE] No more file descriptors are available for the system.
- 43259 [EPROTONOSUPPORT]  
 43260 The protocol is not supported by the address family, or the protocol is not  
 43261 supported by the implementation.
- 43262 [EPROTOYPE] The socket type is not supported by the protocol.
- 43263 The *socket()* function may fail if:
- 43264 [EACCES] The process does not have appropriate privileges.
- 43265 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 43266 [ENOMEM] Insufficient memory was available to fulfill the request.
- 43267 **EXAMPLES**
- 43268 None.
- 43269 **APPLICATION USAGE**
- 43270 The documentation for specific address families specifies which protocols each address family  
 43271 supports. The documentation for specific protocols specifies which socket types each protocol  
 43272 supports.
- 43273 The application can determine whether an address family is supported by trying to create a  
 43274 socket with *domain* set to the protocol in question.
- 43275 **RATIONALE**
- 43276 None.
- 43277 **FUTURE DIRECTIONS**
- 43278 None.
- 43279 **SEE ALSO**
- 43280 *accept()*, *bind()*, *connect()*, *getsockname()*, *getsockopt()*, *listen()*, *recv()*, *recvfrom()*, *recvmsg()*,  
 43281 *send()*, *sendmsg()*, *setsockopt()*, *shutdown()*, *socketpair()*, the Base Definitions volume of  
 43282 IEEE Std 1003.1-2001, <[netinet/in.h](#)>, <[sys/socket.h](#)>
- 43283 **CHANGE HISTORY**
- 43284 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

43285 **NAME**

43286 socketpair — create a pair of connected sockets

43287 **SYNOPSIS**

43288 #include &lt;sys/socket.h&gt;

43289 int socketpair(int *domain*, int *type*, int *protocol*,  
43290 int *socket\_vector*[2]);43291 **DESCRIPTION**43292 The *socketpair()* function shall create an unbound pair of connected sockets in a specified *domain*,  
43293 of a specified *type*, under the protocol optionally specified by the *protocol* argument. The two  
43294 sockets shall be identical. The file descriptors used in referencing the created sockets shall be  
43295 returned in *socket\_vector*[0] and *socket\_vector*[1].43296 The *socketpair()* function takes the following arguments:

|       |                      |                                                                                                                                                                                                                               |
|-------|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 43297 | <i>domain</i>        | Specifies the communications domain in which the sockets are to be created.                                                                                                                                                   |
| 43298 | <i>type</i>          | Specifies the type of sockets to be created.                                                                                                                                                                                  |
| 43299 | <i>protocol</i>      | Specifies a particular protocol to be used with the sockets. Specifying a<br>43300 <i>protocol</i> of 0 causes <i>socketpair()</i> to use an unspecified default protocol<br>43301 appropriate for the requested socket type. |
| 43302 | <i>socket_vector</i> | Specifies a 2-integer array to hold the file descriptors of the created socket<br>43303 pair.                                                                                                                                 |

43304 The *type* argument specifies the socket type, which determines the semantics of communications  
43305 over the socket. The following socket types are defined; implementations may specify additional  
43306 socket types:

|       |                |                                                                                                                                                                                                                                                                                                                                                                                     |
|-------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 43307 | SOCK_STREAM    | Provides sequenced, reliable, bidirectional, connection-mode byte<br>43308 streams, and may provide a transmission mechanism for out-of-band<br>43309 data.                                                                                                                                                                                                                         |
| 43310 | SOCK_DGRAM     | Provides datagrams, which are connectionless-mode, unreliable messages<br>43311 of fixed maximum length.                                                                                                                                                                                                                                                                            |
| 43312 | SOCK_SEQPACKET | Provides sequenced, reliable, bidirectional, connection-mode<br>43313 transmission paths for records. A record can be sent using one or more<br>43314 output operations and received using one or more input operations, but a<br>43315 single operation never transfers part of more than one record. Record<br>43316 boundaries are visible to the receiver via the MSG_EOR flag. |

43317 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address  
43318 family. If the *protocol* argument is zero, the default protocol for this address family and type shall  
43319 be used. The protocols supported by the system are implementation-defined.43320 The process may need to have appropriate privileges to use the *socketpair()* function or to create  
43321 some sockets.43322 **RETURN VALUE**43323 Upon successful completion, this function shall return 0; otherwise,  $-1$  shall be returned and  
43324 *errno* set to indicate the error.43325 **ERRORS**43326 The *socketpair()* function shall fail if:

43327 [EAFNOSUPPORT]

43328 The implementation does not support the specified address family.

- 43329 [EMFILE] No more file descriptors are available for this process.
- 43330 [ENFILE] No more file descriptors are available for the system.
- 43331 [EOPNOTSUPP] The specified protocol does not permit creation of socket pairs.
- 43332 [EPROTONOSUPPORT]  
 43333 The protocol is not supported by the address family, or the protocol is not  
 43334 supported by the implementation.
- 43335 [EPROTOTYPE] The socket type is not supported by the protocol.
- 43336 The *socketpair()* function may fail if:
- 43337 [EACCES] The process does not have appropriate privileges.
- 43338 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 43339 [ENOMEM] Insufficient memory was available to fulfill the request.

43340 **EXAMPLES**

43341 None.

43342 **APPLICATION USAGE**

43343 The documentation for specific address families specifies which protocols each address family  
 43344 supports. The documentation for specific protocols specifies which socket types each protocol  
 43345 supports.

43346 The *socketpair()* function is used primarily with UNIX domain sockets and need not be  
 43347 supported for other domains.

43348 **RATIONALE**

43349 None.

43350 **FUTURE DIRECTIONS**

43351 None.

43352 **SEE ALSO**43353 *socket()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/socket.h>43354 **CHANGE HISTORY**

43355 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

43356 **NAME**

43357        sprintf — print formatted output

43358 **SYNOPSIS**

43359        #include &lt;stdio.h&gt;

43360        int sprintf(char \*restrict *s*, const char \*restrict *format*, ...);43361 **DESCRIPTION**43362        Refer to *fprintf()*.

43363 **NAME**

43364 sqrt, sqrtf, sqrtl — square root function

43365 **SYNOPSIS**

43366 #include &lt;math.h&gt;

43367 double sqrt(double x);

43368 float sqrtf(float x);

43369 long double sqrtl(long double x);

43370 **DESCRIPTION**

43371 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 43372 conflict between the requirements described here and the ISO C standard is unintentional. This  
 43373 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

43374 These functions shall compute the square root of their argument  $x$ ,  $\sqrt{x}$ .

43375 An application wishing to check for error situations should set *errno* to zero and call  
 43376 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 43377 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 43378 zero, an error has occurred.

43379 **RETURN VALUE**43380 Upon successful completion, these functions shall return the square root of  $x$ .

43381 **MX** For finite values of  $x < -0$ , a domain error shall occur, and either a NaN (if supported), or an  
 43382 implementation-defined value shall be returned.

43383 **MX** If  $x$  is NaN, a NaN shall be returned.

43384 If  $x$  is  $\pm 0$  or  $+\text{Inf}$ ,  $x$  shall be returned.

43385 If  $x$  is  $-\text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an implementation-  
 43386 defined value shall be returned.

43387 **ERRORS**

43388 These functions shall fail if:

43389 **MX** Domain Error The finite value of  $x$  is  $< -0$ , or  $x$  is  $-\text{Inf}$ .

43390 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 43391 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling  
 43392 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 43393 shall be raised.

43394 **EXAMPLES**43395 **Taking the Square Root of 9.0**

43396 #include &lt;math.h&gt;

43397 ...

43398 double x = 9.0;

43399 double result;

43400 ...

43401 result = sqrt(x);

43402 **APPLICATION USAGE**

43403           On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`  
43404           `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

43405 **RATIONALE**

43406           None.

43407 **FUTURE DIRECTIONS**

43408           None.

43409 **SEE ALSO**

43410           *feclearexcept()*, *fetetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
43411           Section 4.18, Treatment of Error Conditions for Mathematical Functions, `<math.h>`, `<stdio.h>`

43412 **CHANGE HISTORY**

43413           First released in Issue 1. Derived from Issue 1 of the SVID.

43414 **Issue 5**

43415           The DESCRIPTION is updated to indicate how an application should check for an error. This  
43416           text was previously published in the APPLICATION USAGE section.

43417 **Issue 6**

43418           The *sqrtf()* and *sqrtl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

43419           The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
43420           revised to align with the ISO/IEC 9899:1999 standard.

43421           IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
43422           marked.

43423 **NAME**

43424           srand — pseudo-random number generator

43425 **SYNOPSIS**

43426           #include <stdlib.h>

43427           void srand(unsigned *seed*);

43428 **DESCRIPTION**

43429           Refer to *rand()*.

43430 **NAME**

43431           srand48 — seed the uniformly distributed double-precision pseudo-random number generator

43432 **SYNOPSIS**

43433 XSI       #include <stdlib.h>

43434           void srand48(long *seedval*);

43435

43436 **DESCRIPTION**

43437           Refer to *drand48()*.

43438 **NAME**

43439           srandom — seed pseudo-random number generator

43440 **SYNOPSIS**

43441 xSI       #include <stdlib.h>

43442           void srandom(unsigned *seed*);

43443

43444 **DESCRIPTION**

43445           Refer to *initstate()*.

43446 **NAME**43447        `sscanf` — convert formatted input43448 **SYNOPSIS**43449        `#include <stdio.h>`43450        `int sscanf(const char *restrict s, const char *restrict format, ...);`43451 **DESCRIPTION**43452        Refer to `fscanf()`.

## 43453 NAME

43454 stat — get file status

## 43455 SYNOPSIS

43456 #include &lt;sys/stat.h&gt;

43457 int stat(const char \*restrict path, struct stat \*restrict buf);

## 43458 DESCRIPTION

43459 The *stat()* function shall obtain information about the named file and write it to the area pointed  
 43460 to by the *buf* argument. The *path* argument points to a pathname naming a file. Read, write, or  
 43461 execute permission of the named file is not required. An implementation that provides  
 43462 additional or alternate file access control mechanisms may, under implementation-defined  
 43463 conditions, cause *stat()* to fail. In particular, the system may deny the existence of the file  
 43464 specified by *path*.

43465 If the named file is a symbolic link, the *stat()* function shall continue pathname resolution using  
 43466 the contents of the symbolic link, and shall return information pertaining to the resulting file if  
 43467 the file exists.

43468 The *buf* argument is a pointer to a **stat** structure, as defined in the <sys/stat.h> header, into  
 43469 which information is placed concerning the file.

43470 The *stat()* function shall update any time-related fields (as described in the Base Definitions  
 43471 volume of IEEE Std 1003.1-2001, Section 4.7, File Times Update), before writing into the **stat**  
 43472 structure.

43473 Unless otherwise specified, the structure members *st\_mode*, *st\_ino*, *st\_dev*, *st\_uid*, *st\_gid*, *st\_atime*,  
 43474 *st\_ctime*, and *st\_mtime* shall have meaningful values for all file types defined in this volume of  
 43475 IEEE Std 1003.1-2001. The value of the member *st\_nlink* shall be set to the number of links to the  
 43476 file.

## 43477 RETURN VALUE

43478 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 43479 indicate the error.

## 43480 ERRORS

43481 The *stat()* function shall fail if:

43482 [EACCES] Search permission is denied for a component of the path prefix.

43483 [EIO] An error occurred while reading from the file system.

43484 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
43485 argument.

43486 [ENAMETOOLONG]

43487 The length of the *path* argument exceeds {PATH\_MAX} or a pathname  
43488 component is longer than {NAME\_MAX}.43489 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

43490 [ENOTDIR] A component of the path prefix is not a directory.

43491 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file  
43492 serial number cannot be represented correctly in the structure pointed to by  
43493 *buf*.

43494 The `stat()` function may fail if:

43495 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
43496 resolution of the *path* argument.

43497 [ENAMETOOLONG]

43498 As a result of encountering a symbolic link in resolution of the *path* argument,  
43499 the length of the substituted pathname string exceeded {PATH\_MAX}.

43500 [EOVERFLOW] A value to be stored would overflow one of the members of the **stat** structure.

#### 43501 EXAMPLES

##### 43502 Obtaining File Status Information

43503 The following example shows how to obtain file status information for a file named  
43504 **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure.

```
43505 #include <sys/types.h>
43506 #include <sys/stat.h>
43507 #include <fcntl.h>
43508 struct stat buffer;
43509 int status;
43510 ...
43511 status = stat("/home/cnd/mod1", &buffer);
```

##### 43512 Getting Directory Information

43513 The following example fragment gets status information for each entry in a directory. The call to  
43514 the `stat()` function stores file information in the **stat** structure pointed to by *statbuf*. The lines  
43515 that follow the `stat()` call format the fields in the **stat** structure for presentation to the user of the  
43516 program.

```
43517 #include <sys/types.h>
43518 #include <sys/stat.h>
43519 #include <dirent.h>
43520 #include <pwd.h>
43521 #include <grp.h>
43522 #include <time.h>
43523 #include <locale.h>
43524 #include <langinfo.h>
43525 #include <stdio.h>
43526 #include <stdint.h>
43527 struct dirent *dp;
43528 struct stat statbuf;
43529 struct passwd *pwd;
43530 struct group *grp;
43531 struct tm *tm;
43532 char datestring[256];
43533 ...
43534 /* Loop through directory entries. */
43535 while ((dp = readdir(dir)) != NULL) {
43536 /* Get entry's information. */
43537 if (stat(dp->d_name, &statbuf) == -1)
```

```

43538 continue;
43539 /* Print out type, permissions, and number of links. */
43540 printf("%10.10s", sperm (statbuf.st_mode));
43541 printf("%4d", statbuf.st_nlink);
43542 /* Print out owner's name if it is found using getpwuid(). */
43543 if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
43544 printf(" %-8.8s", pwd->pw_name);
43545 else
43546 printf(" %-8d", statbuf.st_uid);
43547 /* Print out group name if it is found using getgrgid(). */
43548 if ((grp = getgrgid(statbuf.st_gid)) != NULL)
43549 printf(" %-8.8s", grp->gr_name);
43550 else
43551 printf(" %-8d", statbuf.st_gid);
43552 /* Print size of file. */
43553 printf(" %9jd", (intmax_t)statbuf.st_size);
43554 tm = localtime(&statbuf.st_mtime);
43555 /* Get localized date string. */
43556 strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
43557 printf(" %s %s\n", datestring, dp->d_name);
43558 }

```

#### 43559 APPLICATION USAGE

43560 None.

#### 43561 RATIONALE

43562 The intent of the paragraph describing “additional or alternate file access control mechanisms”  
43563 is to allow a secure implementation where a process with a label that does not dominate the  
43564 file’s label cannot perform a *stat()* function. This is not related to read permission; a process with  
43565 a label that dominates the file’s label does not need read permission. An implementation that  
43566 supports write-up operations could fail *lstat()* function calls even though it has a valid file  
43567 descriptor open for writing.

#### 43568 FUTURE DIRECTIONS

43569 None.

#### 43570 SEE ALSO

43571 *lstat()*, *lstat()*, *readlink()*, *symlink()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
43572 [<sys/stat.h>](#), [<sys/types.h>](#)

#### 43573 CHANGE HISTORY

43574 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 43575 Issue 5

43576 Large File Summit extensions are added.

#### 43577 Issue 6

43578 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

43579 The following new requirements on POSIX implementations derive from alignment with the  
43580 Single UNIX Specification:

- 43581 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
  - 43582 required for conforming implementations of previous POSIX specifications, it was not
  - 43583 required for UNIX applications.
  - 43584 • The [EIO] mandatory error condition is added.
  - 43585 • The [ELOOP] mandatory error condition is added.
  - 43586 • The [EOVERFLOW] mandatory error condition is added. This change is to support large
  - 43587 files.
  - 43588 • The [ENAMETOOLONG] and the second [EOVERFLOW] optional error conditions are
  - 43589 added.
- 43590 The following changes were made to align with the IEEE P1003.1a draft standard:
- 43591 • Details are added regarding the treatment of symbolic links.
  - 43592 • The [ELOOP] optional error condition is added.
- 43593 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.
- 43594 The **restrict** keyword is added to the `stat()` prototype for alignment with the ISO/IEC 9899:1999
- 43595 standard.

43596 **NAME**

43597           statvfs — get file system information

43598 **SYNOPSIS**

43599 XSI        #include <sys/statvfs.h>

43600           int statvfs(const char \*restrict path, struct statvfs \*restrict buf);

43601

43602 **DESCRIPTION**

43603           Refer to *fstatvfs()*.

43604 **NAME**

43605 stderr, stdin, stdout — standard I/O streams

43606 **SYNOPSIS**

43607 #include &lt;stdio.h&gt;

43608 extern FILE \*stderr, \*stdin, \*stdout;

43609 **DESCRIPTION**

43610 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 43611 conflict between the requirements described here and the ISO C standard is unintentional. This  
 43612 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

43613 A file with associated buffering is called a *stream* and is declared to be a pointer to a defined type  
 43614 **FILE**. The *fopen()* function shall create certain descriptive data for a stream and return a pointer  
 43615 to designate the stream in all further transactions. Normally, there are three open streams with  
 43616 constant pointers declared in the <stdio.h> header and associated with the standard open files.

43617 At program start-up, three streams shall be predefined and need not be opened explicitly:  
 43618 *standard input* (for reading conventional input), *standard output* (for writing conventional output),  
 43619 and *standard error* (for writing diagnostic output). When opened, the standard error stream is not  
 43620 fully buffered; the standard input and standard output streams are fully buffered if and only if  
 43621 the stream can be determined not to refer to an interactive device.

43622 **CX** The following symbolic values in <unistd.h> define the file descriptors that shall be associated  
 43623 with the C-language *stdin*, *stdout*, and *stderr* when the application is started:

43624 **STDIN\_FILENO** Standard input value, *stdin*. Its value is 0.

43625 **STDOUT\_FILENO** Standard output value, *stdout*. Its value is 1.

43626 **STDERR\_FILENO** Standard error value, *stderr*. Its value is 2.

43627 The *stderr* stream is expected to be open for reading and writing.

43628 **RETURN VALUE**

43629 None.

43630 **ERRORS**

43631 No errors are defined.

43632 **EXAMPLES**

43633 None.

43634 **APPLICATION USAGE**

43635 None.

43636 **RATIONALE**

43637 None.

43638 **FUTURE DIRECTIONS**

43639 None.

43640 **SEE ALSO**

43641 *fclose()*, *feof()*, *ferror()*, *fileno()*, *fopen()*, *fread()*, *fseek()*, *getc()*, *gets()*, *popen()*, *printf()*, *putc()*,  
 43642 *puts()*, *read()*, *scanf()*, *setbuf()*, *setvbuf()*, *tmpfile()*, *ungetc()*, *vprintf()*, the Base Definitions  
 43643 volume of IEEE Std 1003.1-2001, <stdio.h>, <unistd.h>

43644 **CHANGE HISTORY**

43645 First released in Issue 1.

43646 **Issue 6**

43647 Extensions beyond the ISO C standard are marked.

43648 A note that *stderr* is expected to be open for reading and writing is added to the DESCRIPTION.

43649 **NAME**

43650           strcasecmp, strncasecmp — case-insensitive string comparisons

43651 **SYNOPSIS**

43652 XSI           #include &lt;strings.h&gt;

43653           int strcmp(const char \*s1, const char \*s2);

43654           int strncasecmp(const char \*s1, const char \*s2, size\_t n);

43655

43656 **DESCRIPTION**

43657           The *strcasecmp()* function shall compare, while ignoring differences in case, the string pointed to  
43658           by *s1* to the string pointed to by *s2*. The *strncasecmp()* function shall compare, while ignoring  
43659           differences in case, not more than *n* bytes from the string pointed to by *s1* to the string pointed to  
43660           by *s2*.

43661           In the POSIX locale, *strcasecmp()* and *strncasecmp()* shall behave as if the strings had been  
43662           converted to lowercase and then a byte comparison performed. The results are unspecified in  
43663           other locales.

43664 **RETURN VALUE**

43665           Upon completion, *strcasecmp()* shall return an integer greater than, equal to, or less than 0, if the  
43666           string pointed to by *s1* is, ignoring case, greater than, equal to, or less than the string pointed to  
43667           by *s2*, respectively.

43668           Upon successful completion, *strncasecmp()* shall return an integer greater than, equal to, or less  
43669           than 0, if the possibly null-terminated array pointed to by *s1* is, ignoring case, greater than, equal  
43670           to, or less than the possibly null-terminated array pointed to by *s2*, respectively.

43671 **ERRORS**

43672           No errors are defined.

43673 **EXAMPLES**

43674           None.

43675 **APPLICATION USAGE**

43676           None.

43677 **RATIONALE**

43678           None.

43679 **FUTURE DIRECTIONS**

43680           None.

43681 **SEE ALSO**

43682           The Base Definitions volume of IEEE Std 1003.1-2001, &lt;strings.h&gt;

43683 **CHANGE HISTORY**

43684           First released in Issue 4, Version 2.

43685 **Issue 5**

43686           Moved from X/OPEN UNIX extension to BASE.

43687 **NAME**43688            **strcat** — concatenate two strings43689 **SYNOPSIS**

43690            #include &lt;string.h&gt;

43691            char \*strcat(char \*restrict *s1*, const char \*restrict *s2*);43692 **DESCRIPTION**

43693 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
43694 conflict between the requirements described here and the ISO C standard is unintentional. This  
43695 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

43696        The *strcat()* function shall append a copy of the string pointed to by *s2* (including the  
43697 terminating null byte) to the end of the string pointed to by *s1*. The initial byte of *s2* overwrites  
43698 the null byte at the end of *s1*. If copying takes place between objects that overlap, the behavior is  
43699 undefined.

43700 **RETURN VALUE**43701        The *strcat()* function shall return *s1*; no return value is reserved to indicate an error.43702 **ERRORS**

43703        No errors are defined.

43704 **EXAMPLES**

43705        None.

43706 **APPLICATION USAGE**

43707        This issue is aligned with the ISO C standard; this does not affect compatibility with XPG3  
43708 applications. Reliable error detection by this function was never guaranteed.

43709 **RATIONALE**

43710        None.

43711 **FUTURE DIRECTIONS**

43712        None.

43713 **SEE ALSO**43714        *strncat()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>43715 **CHANGE HISTORY**

43716        First released in Issue 1. Derived from Issue 1 of the SVID.

43717 **Issue 6**43718        The *strcat()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

43719 **NAME**

43720 strchr — string scanning operation

43721 **SYNOPSIS**

43722 #include &lt;string.h&gt;

43723 char \*strchr(const char \*s, int c);

43724 **DESCRIPTION**

43725 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
43726 conflict between the requirements described here and the ISO C standard is unintentional. This  
43727 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

43728 The *strchr()* function shall locate the first occurrence of *c* (converted to a **char**) in the string  
43729 pointed to by *s*. The terminating null byte is considered to be part of the string.

43730 **RETURN VALUE**

43731 Upon completion, *strchr()* shall return a pointer to the byte, or a null pointer if the byte was not  
43732 found.

43733 **ERRORS**

43734 No errors are defined.

43735 **EXAMPLES**

43736 None.

43737 **APPLICATION USAGE**

43738 None.

43739 **RATIONALE**

43740 None.

43741 **FUTURE DIRECTIONS**

43742 None.

43743 **SEE ALSO**43744 *strchr()*, the Base Definitions volume of IEEE Std 1003.1-2001, <string.h>43745 **CHANGE HISTORY**

43746 First released in Issue 1. Derived from Issue 1 of the SVID.

43747 **Issue 6**

43748 Extensions beyond the ISO C standard are marked.

43749 **NAME**

43750 stricmp — compare two strings

43751 **SYNOPSIS**

43752 #include &lt;string.h&gt;

43753 int stricmp(const char \*s1, const char \*s2);

43754 **DESCRIPTION**

43755 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 43756 conflict between the requirements described here and the ISO C standard is unintentional. This  
 43757 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

43758 The *stricmp()* function shall compare the string pointed to by *s1* to the string pointed to by *s2*.

43759 The sign of a non-zero return value shall be determined by the sign of the difference between the  
 43760 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings  
 43761 being compared.

43762 **RETURN VALUE**

43763 Upon completion, *stricmp()* shall return an integer greater than, equal to, or less than 0, if the  
 43764 string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2*,  
 43765 respectively.

43766 **ERRORS**

43767 No errors are defined.

43768 **EXAMPLES**43769 **Checking a Password Entry**

43770 The following example compares the information read from standard input to the value of the  
 43771 name of the user entry. If the *stricmp()* function returns 0 (indicating a match), a further check  
 43772 will be made to see if the user entered the proper old password. The *crypt()* function shall  
 43773 encrypt the old password entered by the user, using the value of the encrypted password in the  
 43774 **passwd** structure as the salt. If this value matches the value of the encrypted **passwd** in the  
 43775 structure, the entered password *oldpasswd* is the correct user's password. Finally, the program  
 43776 encrypts the new password so that it can store the information in the **passwd** structure.

```

43777 #include <string.h>
43778 #include <unistd.h>
43779 #include <stdio.h>
43780 ...
43781 int valid_change;
43782 struct passwd *p;
43783 char user[100];
43784 char oldpasswd[100];
43785 char newpasswd[100];
43786 char savepasswd[100];
43787 ...
43788 if (stricmp(p->pw_name, user) == 0) {
43789 if (stricmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
43790 strcpy(savepasswd, crypt(newpasswd, user));
43791 p->pw_passwd = savepasswd;
43792 valid_change = 1;
43793 }
43794 else {

```

```
43795 fprintf(stderr, "Old password is not valid\n");
43796 }
43797 }
43798 ...
```

**43799 APPLICATION USAGE**

43800 None.

**43801 RATIONALE**

43802 None.

**43803 FUTURE DIRECTIONS**

43804 None.

**43805 SEE ALSO**

43806 *strncmp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>

**43807 CHANGE HISTORY**

43808 First released in Issue 1. Derived from Issue 1 of the SVID.

**43809 Issue 6**

43810 Extensions beyond the ISO C standard are marked.

43811 **NAME**

43812 strcoll — string comparison using collating information

43813 **SYNOPSIS**

43814 #include &lt;string.h&gt;

43815 int strcoll(const char \*s1, const char \*s2);

43816 **DESCRIPTION**

43817 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 43818 conflict between the requirements described here and the ISO C standard is unintentional. This  
 43819 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

43820 The *strcoll()* function shall compare the string pointed to by *s1* to the string pointed to by *s2*,  
 43821 both interpreted as appropriate to the *LC\_COLLATE* category of the current locale.

43822 CX The *strcoll()* function shall not change the setting of *errno* if successful.

43823 Since no return value is reserved to indicate an error, an application wishing to check for error  
 43824 situations should set *errno* to 0, then call *strcoll()*, then check *errno*.

43825 **RETURN VALUE**

43826 Upon successful completion, *strcoll()* shall return an integer greater than, equal to, or less than 0,  
 43827 according to whether the string pointed to by *s1* is greater than, equal to, or less than the string  
 43828 CX pointed to by *s2* when both are interpreted as appropriate to the current locale. On error,  
 43829 *strcoll()* may set *errno*, but no return value is reserved to indicate an error.

43830 **ERRORS**43831 The *strcoll()* function may fail if:

43832 CX [EINVAL] The *s1* or *s2* arguments contain characters outside the domain of the collating  
 43833 sequence.

43834 **EXAMPLES**43835 **Comparing Nodes**

43836 The following example uses an application-defined function, *node\_compare()*, to compare two  
 43837 nodes based on an alphabetical ordering of the *string* field.

```
43838 #include <string.h>
43839 ...
43840 struct node { /* These are stored in the table. */
43841 char *string;
43842 int length;
43843 };
43844 ...
43845 int node_compare(const void *node1, const void *node2)
43846 {
43847 return strcoll(((const struct node *)node1)->string,
43848 ((const struct node *)node2)->string);
43849 }
43850 ...
```

43851 **APPLICATION USAGE**43852 The *strxfrm()* and *strcmp()* functions should be used for sorting large lists.

43853 **RATIONALE**

43854           None.

43855 **FUTURE DIRECTIONS**

43856           None.

43857 **SEE ALSO**43858           *strcmp()*, *strxfrm()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>43859 **CHANGE HISTORY**

43860           First released in Issue 3.

43861 **Issue 5**43862           The DESCRIPTION is updated to indicate that *errno* does not change if the function is  
43863           successful.43864 **Issue 6**

43865           Extensions beyond the ISO C standard are marked.

43866           The following new requirements on POSIX implementations derive from alignment with the  
43867           Single UNIX Specification:

- 43868
- The [EINVAL] optional error condition is added.

43869           An example is added.

43870 **NAME**

43871 strcpy — copy a string

43872 **SYNOPSIS**

43873 #include &lt;string.h&gt;

43874 char \*strcpy(char \*restrict s1, const char \*restrict s2);

43875 **DESCRIPTION**

43876 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 43877 conflict between the requirements described here and the ISO C standard is unintentional. This  
 43878 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

43879 The *strcpy()* function shall copy the string pointed to by *s2* (including the terminating null byte)  
 43880 into the array pointed to by *s1*. If copying takes place between objects that overlap, the behavior  
 43881 is undefined.

43882 **RETURN VALUE**43883 The *strcpy()* function shall return *s1*; no return value is reserved to indicate an error.43884 **ERRORS**

43885 No errors are defined.

43886 **EXAMPLES**43887 **Initializing a String**43888 The following example copies the string "-----" into the *permstring* variable.

```
43889 #include <string.h>
43890 ...
43891 static char permstring[11];
43892 ...
43893 strcpy(permstring, "-----");
43894 ...
```

43895 **Storing a Key and Data**

43896 The following example allocates space for a key using *malloc()* then uses *strcpy()* to place the  
 43897 key there. Then it allocates space for data using *malloc()*, and uses *strcpy()* to place data there.  
 43898 (The user-defined function *dbfree()* frees memory previously allocated to an array of type **struct**  
 43899 **element** \*.)

```
43900 #include <string.h>
43901 #include <stdlib.h>
43902 #include <stdio.h>
43903 ...
43904 /* Structure used to read data and store it. */
43905 struct element {
43906 char *key;
43907 char *data;
43908 };
43909 struct element *tbl, *curtbl;
43910 char *key, *data;
43911 int count;
43912 ...
43913 void dbfree(struct element *, int);
```

```
43914 ...
43915 if ((curtbl->key = malloc(strlen(key) + 1)) == NULL) {
43916 perror("malloc"); dbfree(tbl, count); return NULL;
43917 }
43918 strcpy(curtbl->key, key);
43919 if ((curtbl->data = malloc(strlen(data) + 1)) == NULL) {
43920 perror("malloc"); free(curtbl->key); dbfree(tbl, count); return NULL;
43921 }
43922 strcpy(curtbl->data, data);
43923 ...
```

**43924 APPLICATION USAGE**

43925 Character movement is performed differently in different implementations. Thus, overlapping  
43926 moves may yield surprises.

43927 This issue is aligned with the ISO C standard; this does not affect compatibility with XPG3  
43928 applications. Reliable error detection by this function was never guaranteed.

**43929 RATIONALE**

43930 None.

**43931 FUTURE DIRECTIONS**

43932 None.

**43933 SEE ALSO**

43934 *strncpy()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>

**43935 CHANGE HISTORY**

43936 First released in Issue 1. Derived from Issue 1 of the SVID.

**43937 Issue 6**

43938 The *strcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

43939 **NAME**

43940 strcspn — get the length of a complementary substring

43941 **SYNOPSIS**

43942 #include <string.h>

43943 size\_t strcspn(const char \*s1, const char \*s2);

43944 **DESCRIPTION**

43945 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
43946 conflict between the requirements described here and the ISO C standard is unintentional. This  
43947 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

43948 The *strcspn()* function shall compute the length (in bytes) of the maximum initial segment of the  
43949 string pointed to by *s1* which consists entirely of bytes *not* from the string pointed to by *s2*.

43950 **RETURN VALUE**

43951 The *strcspn()* function shall return the length of the computed segment of the string pointed to  
43952 by *s1*; no return value is reserved to indicate an error.

43953 **ERRORS**

43954 No errors are defined.

43955 **EXAMPLES**

43956 None.

43957 **APPLICATION USAGE**

43958 None.

43959 **RATIONALE**

43960 None.

43961 **FUTURE DIRECTIONS**

43962 None.

43963 **SEE ALSO**

43964 *strspn()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>

43965 **CHANGE HISTORY**

43966 First released in Issue 1. Derived from Issue 1 of the SVID.

43967 **Issue 5**

43968 The RETURN VALUE section is updated to indicate that *strcspn()* returns the length of *s1*, and  
43969 not *s1* itself as was previously stated.

43970 **Issue 6**

43971 The Open Group Corrigendum U030/1 is applied. The text of the RETURN VALUE section is  
43972 updated to indicate that the computed segment length is returned, not the *s1* length.

43973 **NAME**

43974            strdup — duplicate a string

43975 **SYNOPSIS**

43976 XSI        #include &lt;string.h&gt;

43977            char \*strdup(const char \*s1);

43978

43979 **DESCRIPTION**

43980            The *strdup()* function shall return a pointer to a new string, which is a duplicate of the string pointed to by *s1*. The returned pointer can be passed to *free()*. A null pointer is returned if the new string cannot be created.

43983 **RETURN VALUE**

43984            The *strdup()* function shall return a pointer to a new string on success. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

43986 **ERRORS**43987            The *strdup()* function may fail if:

43988            [ENOMEM]       Storage space available is insufficient.

43989 **EXAMPLES**

43990            None.

43991 **APPLICATION USAGE**

43992            None.

43993 **RATIONALE**

43994            None.

43995 **FUTURE DIRECTIONS**

43996            None.

43997 **SEE ALSO**43998            *free()*, *malloc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <string.h>43999 **CHANGE HISTORY**

44000            First released in Issue 4, Version 2.

44001 **Issue 5**

44002            Moved from X/OPEN UNIX extension to BASE.

## 44003 NAME

44004            strerror, strerror\_r — get error message string

## 44005 SYNOPSIS

44006            #include &lt;string.h&gt;

44007            char \*strerror(int *errnum*);44008 TSF        int strerror\_r(int *errnum*, char \**strerrbuf*, size\_t *buflen*);

44009

## 44010 DESCRIPTION

44011 CX        For *strerror()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44014        The *strerror()* function shall map the error number in *errnum* to a locale-dependent error message string and shall return a pointer to it. Typically, the values for *errnum* come from *errno*, but *strerror()* shall map any value of type **int** to a message.

44017        The string pointed to shall not be modified by the application, but may be overwritten by a subsequent call to *strerror()* or *perror()*.

44019 CX        The contents of the error message strings returned by *strerror()* should be determined by the setting of the *LC\_MESSAGES* category in the current locale.

44021        The implementation shall behave as if no function defined in this volume of IEEE Std 1003.1-2001 calls *strerror()*.

44023 CX        The *strerror()* function shall not change the setting of *errno* if successful.

44024        Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *strerror()*, then check *errno*.

44026        The *strerror()* function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

44028 TSF        The *strerror\_r()* function shall map the error number in *errnum* to a locale-dependent error message string and shall return the string in the buffer pointed to by *strerrbuf*, with length *buflen*.

44030

## 44031 RETURN VALUE

44032        Upon successful completion, *strerror()* shall return a pointer to the generated message string. On error *errno* may be set, but no return value is reserved to indicate an error.

44034 TSF        Upon successful completion, *strerror\_r()* shall return 0. Otherwise, an error number shall be returned to indicate the error.

44035

## 44036 ERRORS

44037        These functions may fail if:

44038        [EINVAL]        The value of *errnum* is not a valid error number.

44039        The *strerror\_r()* function may fail if:

44040 TSF        [ERANGE]        Insufficient storage was supplied via *strerrbuf* and *buflen* to contain the generated message string.

44041

44042 **EXAMPLES**

44043 None.

44044 **APPLICATION USAGE**

44045 None.

44046 **RATIONALE**

44047 None.

44048 **FUTURE DIRECTIONS**

44049 None.

44050 **SEE ALSO**44051 *perrot()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>44052 **CHANGE HISTORY**

44053 First released in Issue 3.

44054 **Issue 5**44055 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

44056 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

44057 **Issue 6**

44058 Extensions beyond the ISO C standard are marked.

44059 The following new requirements on POSIX implementations derive from alignment with the  
44060 Single UNIX Specification:

- 44061 • In the RETURN VALUE section, the fact that *errno* may be set is added.
- 44062 • The [EINVAL] optional error condition is added.

44063 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

44064 The *strerror\_r()* function is added in response to IEEE PASC Interpretation 1003.1c #39.44065 The *strerror\_r()* function is marked as part of the Thread-Safe Functions option.

## 44066 NAME

44067 strfmon — convert monetary value to a string

## 44068 SYNOPSIS

44069 XSI 

```
#include <monetary.h>
```

44070 

```
ssize_t strfmon(char *restrict s, size_t maxsize,
```

  
44071 

```
const char *restrict format, ...);
```

44072

## 44073 DESCRIPTION

44074 The *strfmon()* function shall place characters into the array pointed to by *s* as controlled by the  
44075 string pointed to by *format*. No more than *maxsize* bytes are placed into the array.44076 The format is a character string, beginning and ending in its initial state, if any, that contains two  
44077 types of objects: *plain characters*, which are simply copied to the output stream, and *conversion*  
44078 *specifications*, each of which shall result in the fetching of zero or more arguments which are  
44079 converted and formatted. The results are undefined if there are insufficient arguments for the  
44080 format. If the format is exhausted while arguments remain, the excess arguments are simply  
44081 ignored.

44082 The application shall ensure that a conversion specification consists of the following sequence:

- 44083
- A '%' character
  - 44084 • Optional flags
  - 44085 • Optional field width
  - 44086 • Optional left precision
  - 44087 • Optional right precision
  - 44088 • A required conversion specifier character that determines the conversion to be performed

44089 **Flags**

44090 One or more of the following optional flags can be specified to control the conversion:

- 44091
- =f*
- An '=' followed by a single character
- f*
- which is used as the numeric fill character. In
- 
- 44092 order to work with precision or width counts, the fill character shall be a single byte
- 
- 44093 character; if not, the behavior is undefined. The default numeric fill character is the
- 
- 44094 <space>. This flag does not affect field width filling which always uses the <space>.
- 
- 44095 This flag is ignored unless a left precision (see below) is specified.
- 
- 44096
- ^*
- Do not format the currency amount with grouping characters. The default is to insert
- 
- 44097 the grouping characters if defined for the current locale.
- 
- 44098
- + or (*
- Specify the style of representing positive and negative currency amounts. Only one of
- 
- 44099 '+' or '(' may be specified. If '+' is specified, the locale's equivalent of '+' and '-'
- 
- 44100 are used (for example, in the U.S., the empty string if positive and '-' if negative). If
- 
- 44101 '(' is specified, negative amounts are enclosed within parentheses. If neither flag is
- 
- 44102 specified, the '+' style is used.
- 
- 44103
- !*
- Suppress the currency symbol from the output conversion.
- 
- 44104
- 
- Specify the alignment. If this flag is present the result of the conversion is left-justified
- 
- 44105 (padded to the right) rather than right-justified. This flag shall be ignored unless a field
- 
- 44106 width (see below) is specified.

44107 **Field Width**

44108 *w* A decimal digit string *w* specifying a minimum field width in bytes in which the result  
 44109 of the conversion is right-justified (or left-justified if the flag '-' is specified). The  
 44110 default is 0.

44111 **Left Precision**

44112 *#n* A '#' followed by a decimal digit string *n* specifying a maximum number of digits  
 44113 expected to be formatted to the left of the radix character. This option can be used to  
 44114 keep the formatted output from multiple calls to the *strfmon()* function aligned in the  
 44115 same columns. It can also be used to fill unused positions with a special character as in  
 44116 "\$\*\*\*123.45". This option causes an amount to be formatted as if it has the number  
 44117 of digits specified by *n*. If more than *n* digit positions are required, this conversion  
 44118 specification is ignored. Digit positions in excess of those actually required are filled  
 44119 with the numeric fill character (see the *=f* flag above).

44120 If grouping has not been suppressed with the '^' flag, and it is defined for the current  
 44121 locale, grouping separators are inserted before the fill characters (if any) are added.  
 44122 Grouping separators are not applied to fill characters even if the fill character is a digit.

44123 To ensure alignment, any characters appearing before or after the number in the  
 44124 formatted output such as currency or sign symbols are padded as necessary with  
 44125 <space>s to make their positive and negative formats an equal length.

44126 **Right Precision**

44127 *.p* A period followed by a decimal digit string *p* specifying the number of digits after the  
 44128 radix character. If the value of the right precision *p* is 0, no radix character appears. If a  
 44129 right precision is not included, a default specified by the current locale is used. The  
 44130 amount being formatted is rounded to the specified number of digits prior to  
 44131 formatting.

44132 **Conversion Specifier Characters**

44133 The conversion specifier characters and their meanings are:

44134 *i* The **double** argument is formatted according to the locale's international currency  
 44135 format (for example, in the U.S.: USD 1,234.56). If the argument is  $\pm\text{Inf}$  or NaN, the  
 44136 result of the conversion is unspecified.

44137 *n* The **double** argument is formatted according to the locale's national currency format  
 44138 (for example, in the U.S.: \$1,234.56). If the argument is  $\pm\text{Inf}$  or NaN, the result of the  
 44139 conversion is unspecified.

44140 *%* Convert to a '%'; no argument is converted. The entire conversion specification shall  
 44141 be %%.

44142 **Locale Information**

44143 The *LC\_MONETARY* category of the program's locale affects the behavior of this function  
 44144 including the monetary radix character (which may be different from the numeric radix  
 44145 character affected by the *LC\_NUMERIC* category), the grouping separator, the currency  
 44146 symbols, and formats. The international currency symbol should be conformant with the  
 44147 ISO 4217:2001 standard.

44148 If the value of *maxsize* is greater than {SSIZE\_MAX}, the result is implementation-defined.

## 44149 RETURN VALUE

44150 If the total number of resulting bytes including the terminating null byte is not more than  
 44151 *maxsize*, *strfmon()* shall return the number of bytes placed into the array pointed to by *s*, not  
 44152 including the terminating null byte. Otherwise, -1 shall be returned, the contents of the array are  
 44153 unspecified, and *errno* shall be set to indicate the error.

## 44154 ERRORS

44155 The *strfmon()* function shall fail if:

44156 [E2BIG] Conversion stopped due to lack of space in the buffer.

## 44157 EXAMPLES

44158 Given a locale for the U.S. and the values 123.45, -123.45, and 3456.781, the following output  
 44159 might be produced. Square brackets (" [ ] ") are used in this example to delimit the output.

|       |           |                   |                                          |
|-------|-----------|-------------------|------------------------------------------|
| 44160 | %n        | [\$123.45]        | Default formatting                       |
| 44161 |           | [-\$123.45]       |                                          |
| 44162 |           | [\$3,456.78]      |                                          |
| 44163 | %11n      | [ \$123.45]       | Right align within an 11-character field |
| 44164 |           | [ -\$123.45]      |                                          |
| 44165 |           | [ \$3,456.78]     |                                          |
| 44166 | %#5n      | [ \$ 123.45]      | Aligned columns for values up to 99 999  |
| 44167 |           | [-\$ 123.45]      |                                          |
| 44168 |           | [ \$ 3,456.78]    |                                          |
| 44169 | %=*#5n    | [ \$***123.45]    | Specify a fill character                 |
| 44170 |           | [-\$***123.45]    |                                          |
| 44171 |           | [ \$*3,456.78]    |                                          |
| 44172 | %=0#5n    | [ \$000123.45]    | Fill characters do not use grouping      |
| 44173 |           | [-\$000123.45]    | even if the fill character is a digit    |
| 44174 |           | [ \$03,456.78]    |                                          |
| 44175 | %^#5n     | [ \$ 123.45]      | Disable the grouping separator           |
| 44176 |           | [-\$ 123.45]      |                                          |
| 44177 |           | [ \$ 3456.78]     |                                          |
| 44178 | %^#5.0n   | [ \$ 123]         | Round off to whole units                 |
| 44179 |           | [-\$ 123]         |                                          |
| 44180 |           | [ \$ 3457]        |                                          |
| 44181 | %^#5.4n   | [ \$ 123.4500]    | Increase the precision                   |
| 44182 |           | [-\$ 123.4500]    |                                          |
| 44183 |           | [ \$ 3456.7810]   |                                          |
| 44184 | %(#5n     | [\$ 123.45]       | Use an alternative pos/neg style         |
| 44185 |           | [( \$ 123.45)]    |                                          |
| 44186 |           | [\$ 3,456.78]     |                                          |
| 44187 | %!(#5n    | [ 123.45]         | Disable the currency symbol              |
| 44188 |           | [( 123.45)]       |                                          |
| 44189 |           | [ 3,456.78]       |                                          |
| 44190 | %-14#5.4n | [ \$ 123.4500 ]   | Left-justify the output                  |
| 44191 |           | [-\$ 123.4500 ]   |                                          |
| 44192 |           | [ \$ 3,456.7810 ] |                                          |

44193           %14#5.4n   [   \$   123.4500]   Corresponding right-justified output  
44194                                   [  -\$   123.4500]  
44195                                   [   \$ 3,456.7810]

44196           See also the EXAMPLES section in *fprintf()*.

44197 **APPLICATION USAGE**

44198           None.

44199 **RATIONALE**

44200           None.

44201 **FUTURE DIRECTIONS**

44202           Lowercase conversion characters are reserved for future standards use and uppercase for  
44203           implementation-defined use.

44204 **SEE ALSO**

44205           *fprintf()*, *localeconv()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**monetary.h**>

44206 **CHANGE HISTORY**

44207           First released in Issue 4.

44208 **Issue 5**

44209           Moved from ENHANCED I18N to BASE.

44210           The [ENOSYS] error is removed.

44211           A sentence is added to the DESCRIPTION warning about values of *maxsize* that are greater than  
44212           {SSIZE\_MAX}.

44213 **Issue 6**

44214           The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

44215           The **restrict** keyword is added to the *strfmon()* prototype for alignment with the  
44216           ISO/IEC 9899:1999 standard.

44217           The EXAMPLES section is reworked, clarifying the output format.

## 44218 NAME

44219 strftime — convert date and time to a string

## 44220 SYNOPSIS

44221 #include &lt;time.h&gt;

```
44222 size_t strftime(char *restrict s, size_t maxsize,
44223 const char *restrict format, const struct tm *restrict timeptr);
```

## 44224 DESCRIPTION

44225 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 44226 conflict between the requirements described here and the ISO C standard is unintentional. This  
 44227 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44228 The *strftime()* function shall place bytes into the array pointed to by *s* as controlled by the string  
 44229 pointed to by *format*. The format is a character string, beginning and ending in its initial shift  
 44230 state, if any. The *format* string consists of zero or more conversion specifications and ordinary  
 44231 characters. A conversion specification consists of a '%' character, possibly followed by an E or O  
 44232 modifier, and a terminating conversion specifier character that determines the conversion  
 44233 specification's behavior. All ordinary characters (including the terminating null byte) are copied  
 44234 unchanged into the array. If copying takes place between objects that overlap, the behavior is  
 44235 undefined. No more than *maxsize* bytes are placed into the array. Each conversion specifier is  
 44236 replaced by appropriate characters as described in the following list. The appropriate characters  
 44237 are determined using the *LC\_TIME* category of the current locale and by the values of zero or  
 44238 more members of the broken-down time structure pointed to by *timeptr*, as specified in brackets  
 44239 in the description. If any of the specified values are outside the normal range, the characters  
 44240 stored are unspecified.

44241 CX Local timezone information is used as though *strftime()* called *tzset()*.

44242 The following conversion specifications are supported:

|       |    |                                                                                                                                                             |
|-------|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 44243 | %a | Replaced by the locale's abbreviated weekday name. [ <i>tm_wday</i> ]                                                                                       |
| 44244 | %A | Replaced by the locale's full weekday name. [ <i>tm_wday</i> ]                                                                                              |
| 44245 | %b | Replaced by the locale's abbreviated month name. [ <i>tm_mon</i> ]                                                                                          |
| 44246 | %B | Replaced by the locale's full month name. [ <i>tm_mon</i> ]                                                                                                 |
| 44247 | %c | Replaced by the locale's appropriate date and time representation. (See the Base<br>44248 Definitions volume of IEEE Std 1003.1-2001, <time.h>.)            |
| 44249 | %C | Replaced by the year divided by 100 and truncated to an integer, as a decimal number<br>44250 [00,99]. [ <i>tm_year</i> ]                                   |
| 44251 | %d | Replaced by the day of the month as a decimal number [01,31]. [ <i>tm_mday</i> ]                                                                            |
| 44252 | %D | Equivalent to %m/%d/%y. [ <i>tm_mon</i> , <i>tm_mday</i> , <i>tm_year</i> ]                                                                                 |
| 44253 | %e | Replaced by the day of the month as a decimal number [1,31]; a single digit is preceded<br>44254 by a space. [ <i>tm_mday</i> ]                             |
| 44255 | %F | Equivalent to %Y-%m-%d (the ISO 8601:2000 standard date format). [ <i>tm_year</i> , <i>tm_mon</i> ,<br>44256 <i>tm_mday</i> ]                               |
| 44257 | %g | Replaced by the last 2 digits of the week-based year (see below) as a decimal number<br>44258 [00,99]. [ <i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i> ] |
| 44259 | %G | Replaced by the week-based year (see below) as a decimal number (for example, 1977).<br>44260 [ <i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i> ]          |

|          |    |                                                                                                  |
|----------|----|--------------------------------------------------------------------------------------------------|
| 44261    | %h | Equivalent to %b. [ <i>tm_mon</i> ]                                                              |
| 44262    | %H | Replaced by the hour (24-hour clock) as a decimal number [00,23]. [ <i>tm_hour</i> ]             |
| 44263    | %I | Replaced by the hour (12-hour clock) as a decimal number [01,12]. [ <i>tm_hour</i> ]             |
| 44264    | %j | Replaced by the day of the year as a decimal number [001,366]. [ <i>tm_yday</i> ]                |
| 44265    | %m | Replaced by the month as a decimal number [01,12]. [ <i>tm_mon</i> ]                             |
| 44266    | %M | Replaced by the minute as a decimal number [00,59]. [ <i>tm_min</i> ]                            |
| 44267    | %n | Replaced by a <newline>.                                                                         |
| 44268    | %p | Replaced by the locale's equivalent of either a.m. or p.m. [ <i>tm_hour</i> ]                    |
| 44269 CX | %r | Replaced by the time in a.m. and p.m. notation; in the POSIX locale this shall be                |
| 44270    |    | equivalent to %I:%M:%S %p. [ <i>tm_hour</i> , <i>tm_min</i> , <i>tm_sec</i> ]                    |
| 44271    | %R | Replaced by the time in 24-hour notation (%H:%M). [ <i>tm_hour</i> , <i>tm_min</i> ]             |
| 44272    | %S | Replaced by the second as a decimal number [00,60]. [ <i>tm_sec</i> ]                            |
| 44273    | %t | Replaced by a <tab>.                                                                             |
| 44274    | %T | Replaced by the time (%H:%M:%S). [ <i>tm_hour</i> , <i>tm_min</i> , <i>tm_sec</i> ]              |
| 44275    | %u | Replaced by the weekday as a decimal number [1,7], with 1 representing Monday.                   |
| 44276    |    | [ <i>tm_wday</i> ]                                                                               |
| 44277    | %U | Replaced by the week number of the year as a decimal number [00,53]. The first                   |
| 44278    |    | Sunday of January is the first day of week 1; days in the new year before this are in            |
| 44279    |    | week 0. [ <i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i> ]                                     |
| 44280    | %V | Replaced by the week number of the year (Monday as the first day of the week) as a               |
| 44281    |    | decimal number [01,53]. If the week containing 1 January has four or more days in the            |
| 44282    |    | new year, then it is considered week 1. Otherwise, it is the last week of the previous           |
| 44283    |    | year, and the next week is week 1. Both January 4th and the first Thursday of January            |
| 44284    |    | are always in week 1. [ <i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i> ]                       |
| 44285    | %w | Replaced by the weekday as a decimal number [0,6], with 0 representing Sunday.                   |
| 44286    |    | [ <i>tm_wday</i> ]                                                                               |
| 44287    | %W | Replaced by the week number of the year as a decimal number [00,53]. The first                   |
| 44288    |    | Monday of January is the first day of week 1; days in the new year before this are in            |
| 44289    |    | week 0. [ <i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i> ]                                     |
| 44290    | %x | Replaced by the locale's appropriate date representation. (See the Base Definitions              |
| 44291    |    | volume of IEEE Std 1003.1-2001, <time.h>.)                                                       |
| 44292    | %X | Replaced by the locale's appropriate time representation. (See the Base Definitions              |
| 44293    |    | volume of IEEE Std 1003.1-2001, <time.h>.)                                                       |
| 44294    | %y | Replaced by the last two digits of the year as a decimal number [00,99]. [ <i>tm_year</i> ]      |
| 44295    | %Y | Replaced by the year as a decimal number (for example, 1997). [ <i>tm_year</i> ]                 |
| 44296    | %z | Replaced by the offset from UTC in the ISO 8601:2000 standard format (+hhmm or                   |
| 44297    |    | -hhmm), or by no characters if no timezone is determinable. For example, "-0430"                 |
| 44298 CX |    | means 4 hours 30 minutes behind UTC (west of Greenwich). If <i>tm_isdst</i> is zero, the         |
| 44299    |    | standard time offset is used. If <i>tm_isdst</i> is greater than zero, the daylight savings time |
| 44300    |    | offset is used. If <i>tm_isdst</i> is negative, no characters are returned. [ <i>tm_isdst</i> ]  |

|          |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 44301    | %Z  | Replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists. [ <i>tm_isdst</i> ]                                                                                                                                                                                                                                                                                                                                                                    |
| 44302    |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 44303    | %%  | Replaced by %.                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 44304    |     | If a conversion specification does not correspond to any of the above, the behavior is undefined.                                                                                                                                                                                                                                                                                                                                                                                       |
| 44305 CX |     | If a <b>struct tm</b> broken-down time structure is created by <i>localtime()</i> or <i>localtime_r()</i> , or modified by <i>mktime()</i> , and the value of <i>TZ</i> is subsequently modified, the results of the %Z and %z <i>strftime()</i> conversion specifiers are undefined, when <i>strftime()</i> is called with such a broken-down time structure.                                                                                                                          |
| 44306    |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 44307    |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 44308    |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 44309    |     | If a <b>struct tm</b> broken-down time structure is created or modified by <i>gmtime()</i> or <i>gmtime_r()</i> , it is unspecified whether the result of the %Z and %z conversion specifiers shall refer to UTC or the current local timezone, when <i>strftime()</i> is called with such a broken-down time structure.                                                                                                                                                                |
| 44310    |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 44311    |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 44312    |     | <b>Modified Conversion Specifiers</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 44313    |     | Some conversion specifiers can be modified by the E or O modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist for the current locale (see ERA in the Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3.5, LC_TIME), the behavior shall be as if the unmodified conversion specification were used. |
| 44314    |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 44315    |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 44316    |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 44317    |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 44318    | %Ec | Replaced by the locale's alternative appropriate date and time representation.                                                                                                                                                                                                                                                                                                                                                                                                          |
| 44319    | %EC | Replaced by the name of the base year (period) in the locale's alternative representation.                                                                                                                                                                                                                                                                                                                                                                                              |
| 44320    |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 44321    | %Ex | Replaced by the locale's alternative date representation.                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 44322    | %EX | Replaced by the locale's alternative time representation.                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 44323    | %Ey | Replaced by the offset from %EC (year only) in the locale's alternative representation.                                                                                                                                                                                                                                                                                                                                                                                                 |
| 44324    | %EY | Replaced by the full alternative year representation.                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 44325    | %Od | Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading zeros if there is any alternative symbol for zero; otherwise, with leading spaces.                                                                                                                                                                                                                                                                                      |
| 44326    |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 44327    |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 44328    | %Oe | Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading spaces.                                                                                                                                                                                                                                                                                                                                                                 |
| 44329    |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 44330    | %OH | Replaced by the hour (24-hour clock) using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                    |
| 44331    | %OI | Replaced by the hour (12-hour clock) using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                    |
| 44332    | %Om | Replaced by the month using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 44333    | %OM | Replaced by the minutes using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 44334    | %OS | Replaced by the seconds using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 44335    | %Ou | Replaced by the weekday as a number in the locale's alternative representation (Monday=1).                                                                                                                                                                                                                                                                                                                                                                                              |
| 44336    |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 44337    | %OU | Replaced by the week number of the year (Sunday as the first day of the week, rules corresponding to %U) using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                |
| 44338    |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 44339    | %OV | Replaced by the week number of the year (Monday as the first day of the week, rules corresponding to %V) using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                |
| 44340    |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

44341        %ow     Replaced by the number of the weekday (Sunday=0) using the locale's alternative  
44342                    numeric symbols.

44343        %OW     Replaced by the week number of the year (Monday as the first day of the week) using  
44344                    the locale's alternative numeric symbols.

44345        %Oy     Replaced by the year (offset from %C) using the locale's alternative numeric symbols.

44346        %g, %G, and %V give values according to the ISO 8601:2000 standard week-based year. In this  
44347                    system, weeks begin on a Monday and week 1 of the year is the week that includes January 4th,  
44348                    which is also the week that includes the first Thursday of the year, and is also the first week that  
44349                    contains at least four days in the year. If the first Monday of January is the 2nd, 3rd, or 4th, the  
44350                    preceding days are part of the last week of the preceding year; thus, for Saturday 2nd January  
44351                    1999, %G is replaced by 1998 and %V is replaced by 53. If December 29th, 30th, or 31st is a  
44352                    Monday, it and any following days are part of week 1 of the following year. Thus, for Tuesday  
44353                    30th December 1997, %G is replaced by 1998 and %V is replaced by 01.

44354        If a conversion specifier is not one of the above, the behavior is undefined.

#### 44355 RETURN VALUE

44356        If the total number of resulting bytes including the terminating null byte is not more than  
44357                    *maxsize*, *strptime()* shall return the number of bytes placed into the array pointed to by *s*, not  
44358                    including the terminating null byte. Otherwise, 0 shall be returned and the contents of the array  
44359                    are unspecified.

#### 44360 ERRORS

44361        No errors are defined.

#### 44362 EXAMPLES

##### 44363        Getting a Localized Date String

44364        The following example first sets the locale to the user's default. The locale information will be  
44365                    used in the *nl\_langinfo()* and *strptime()* functions. The *nl\_langinfo()* function returns the localized  
44366                    date string which specifies how the date is laid out. The *strptime()* function takes this information  
44367                    and, using the *tm* structure for values, places the date and time information into *datestring*.

```
44368 #include <time.h>
44369 #include <locale.h>
44370 #include <langinfo.h>
44371 ...
44372 struct tm *tm;
44373 char datestring[256];
44374 ...
44375 setlocale (LC_ALL, "");
44376 ...
44377 strptime (datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
44378 ...
```

#### 44379 APPLICATION USAGE

44380        The range of values for %S is [00,60] rather than [00,59] to allow for the occasional leap second.

44381        Some of the conversion specifications are duplicates of others. They are included for  
44382                    compatibility with *nl\_cxtime()* and *nl\_ascxtime()*, which were published in Issue 2.

44383        Applications should use %Y (4-digit years) in preference to %y (2-digit years).

44384        In the C locale, the E and O modifiers are ignored and the replacement strings for the following  
44385                    specifiers are:

|       |                                                                                                                                                                                          |                                             |
|-------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------|
| 44386 | %a                                                                                                                                                                                       | The first three characters of %A.           |
| 44387 | %A                                                                                                                                                                                       | One of Sunday, Monday, . . . , Saturday.    |
| 44388 | %b                                                                                                                                                                                       | The first three characters of %B.           |
| 44389 | %B                                                                                                                                                                                       | One of January, February, . . . , December. |
| 44390 | %c                                                                                                                                                                                       | Equivalent to %a %b %e %T %Y.               |
| 44391 | %p                                                                                                                                                                                       | One of AM or PM.                            |
| 44392 | %r                                                                                                                                                                                       | Equivalent to %I:%M:%S %p.                  |
| 44393 | %x                                                                                                                                                                                       | Equivalent to %m/%d/%Y.                     |
| 44394 | %X                                                                                                                                                                                       | Equivalent to %T.                           |
| 44395 | %Z                                                                                                                                                                                       | Implementation-defined.                     |
| 44396 | <b>RATIONALE</b>                                                                                                                                                                         |                                             |
| 44397 | None.                                                                                                                                                                                    |                                             |
| 44398 | <b>FUTURE DIRECTIONS</b>                                                                                                                                                                 |                                             |
| 44399 | None.                                                                                                                                                                                    |                                             |
| 44400 | <b>SEE ALSO</b>                                                                                                                                                                          |                                             |
| 44401 | <i>asctime()</i> , <i>clock()</i> , <i>ctime()</i> , <i>difftime()</i> , <i>getdate()</i> , <i>gmtime()</i> , <i>localtime()</i> , <i>mktime()</i> , <i>strptime()</i> , <i>time()</i> , |                                             |
| 44402 | <i>tzset()</i> , <i>utime()</i> , Base Definitions volume of IEEE Std 1003.1-2001, Section 7.3.5, LC_TIME,                                                                               |                                             |
| 44403 | < <b>time.h</b> >                                                                                                                                                                        |                                             |
| 44404 | <b>CHANGE HISTORY</b>                                                                                                                                                                    |                                             |
| 44405 | First released in Issue 3.                                                                                                                                                               |                                             |
| 44406 | <b>Issue 5</b>                                                                                                                                                                           |                                             |
| 44407 | The description of %OV is changed to be consistent with %V and defines Monday as the first day                                                                                           |                                             |
| 44408 | of the week.                                                                                                                                                                             |                                             |
| 44409 | The description of %Oy is clarified.                                                                                                                                                     |                                             |
| 44410 | <b>Issue 6</b>                                                                                                                                                                           |                                             |
| 44411 | Extensions beyond the ISO C standard are marked.                                                                                                                                         |                                             |
| 44412 | The Open Group Corrigendum U033/8 is applied. The %V conversion specifier is changed from                                                                                                |                                             |
| 44413 | “Otherwise, it is week 53 of the previous year, and the next week is week 1” to “Otherwise, it is                                                                                        |                                             |
| 44414 | the last week of the previous year, and the next week is week 1”.                                                                                                                        |                                             |
| 44415 | The following new requirements on POSIX implementations derive from alignment with the                                                                                                   |                                             |
| 44416 | Single UNIX Specification:                                                                                                                                                               |                                             |
| 44417 | <ul style="list-style-type: none"> <li>• The %C, %D, %e, %h, %n, %r, %R, %t, and %T conversion specifiers are added.</li> </ul>                                                          |                                             |
| 44418 | <ul style="list-style-type: none"> <li>• The modified conversion specifiers are added for consistency with the ISO POSIX-2 standard</li> </ul>                                           |                                             |
| 44419 | <i>date</i> utility.                                                                                                                                                                     |                                             |
| 44420 | The following changes are made for alignment with the ISO/IEC 9899:1999 standard:                                                                                                        |                                             |
| 44421 | <ul style="list-style-type: none"> <li>• The <i>strptime()</i> prototype is updated.</li> </ul>                                                                                          |                                             |
| 44422 | <ul style="list-style-type: none"> <li>• The DESCRIPTION is extensively revised.</li> </ul>                                                                                              |                                             |
| 44423 | <ul style="list-style-type: none"> <li>• The %z conversion specifier is added.</li> </ul>                                                                                                |                                             |
| 44424 | A new example is added.                                                                                                                                                                  |                                             |

44425

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/60 is applied.

44426 **NAME**

44427            strlen — get string length

44428 **SYNOPSIS**

44429            #include &lt;string.h&gt;

44430            size\_t strlen(const char \*s);

44431 **DESCRIPTION**

44432 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
 44433 conflict between the requirements described here and the ISO C standard is unintentional. This  
 44434 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44435        The *strlen()* function shall compute the number of bytes in the string to which *s* points, not  
 44436 including the terminating null byte.

44437 **RETURN VALUE**

44438        The *strlen()* function shall return the length of *s*; no return value shall be reserved to indicate an  
 44439 error.

44440 **ERRORS**

44441        No errors are defined.

44442 **EXAMPLES**44443            **Getting String Lengths**

44444        The following example sets the maximum length of *key* and *data* by using *strlen()* to get the  
 44445 lengths of those strings.

```

44446 #include <string.h>
44447 ...
44448 struct element {
44449 char *key;
44450 char *data;
44451 };
44452 ...
44453 char *key, *data;
44454 int len;

44455 *keylength = *datalength = 0;
44456 ...
44457 if ((len = strlen(key)) > *keylength)
44458 *keylength = len;
44459 if ((len = strlen(data)) > *datalength)
44460 *datalength = len;
44461 ...
```

44462 **APPLICATION USAGE**

44463        None.

44464 **RATIONALE**

44465        None.

44466 **FUTURE DIRECTIONS**

44467        None.

44468 **SEE ALSO**

44469 The Base Definitions volume of IEEE Std 1003.1-2001, <string.h>

44470 **CHANGE HISTORY**

44471 First released in Issue 1. Derived from Issue 1 of the SVID.

44472 **Issue 5**

44473 The RETURN VALUE section is updated to indicate that *strlen()* returns the length of *s*, and not  
44474 *s* itself as was previously stated.

44475 **NAME**

44476 strncasecmp — case-insensitive string comparison

44477 **SYNOPSIS**

44478 XSI `#include <strings.h>`

44479 `int strncasecmp(const char *s1, const char *s2, size_t n);`

44480

44481 **DESCRIPTION**

44482 Refer to *strcasecmp()*.

44483 **NAME**

44484           strncat — concatenate a string with part of another

44485 **SYNOPSIS**

44486           #include &lt;string.h&gt;

44487           char \*strncat(char \*restrict *s1*, const char \*restrict *s2*, size\_t *n*);44488 **DESCRIPTION**

44489 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
44490 conflict between the requirements described here and the ISO C standard is unintentional. This  
44491 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44492       The *strncat()* function shall append not more than *n* bytes (a null byte and bytes that follow it  
44493 are not appended) from the array pointed to by *s2* to the end of the string pointed to by *s1*. The  
44494 initial byte of *s2* overwrites the null byte at the end of *s1*. A terminating null byte is always  
44495 appended to the result. If copying takes place between objects that overlap, the behavior is  
44496 undefined.

44497 **RETURN VALUE**44498       The *strncat()* function shall return *s1*; no return value shall be reserved to indicate an error.44499 **ERRORS**

44500       No errors are defined.

44501 **EXAMPLES**

44502       None.

44503 **APPLICATION USAGE**

44504       None.

44505 **RATIONALE**

44506       None.

44507 **FUTURE DIRECTIONS**

44508       None.

44509 **SEE ALSO**44510       *strcat()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>44511 **CHANGE HISTORY**

44512       First released in Issue 1. Derived from Issue 1 of the SVID.

44513 **Issue 6**44514       The *strncat()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

44515 **NAME**

44516 strncmp — compare part of two strings

44517 **SYNOPSIS**

44518 #include &lt;string.h&gt;

44519 int strncmp(const char \*s1, const char \*s2, size\_t n);

44520 **DESCRIPTION**

44521 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
44522 conflict between the requirements described here and the ISO C standard is unintentional. This  
44523 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44524 The *strncmp()* function shall compare not more than *n* bytes (bytes that follow a null byte are not  
44525 compared) from the array pointed to by *s1* to the array pointed to by *s2*.

44526 The sign of a non-zero return value is determined by the sign of the difference between the  
44527 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings  
44528 being compared.

44529 **RETURN VALUE**

44530 Upon successful completion, *strncmp()* shall return an integer greater than, equal to, or less than  
44531 0, if the possibly null-terminated array pointed to by *s1* is greater than, equal to, or less than the  
44532 possibly null-terminated array pointed to by *s2* respectively.

44533 **ERRORS**

44534 No errors are defined.

44535 **EXAMPLES**

44536 None.

44537 **APPLICATION USAGE**

44538 None.

44539 **RATIONALE**

44540 None.

44541 **FUTURE DIRECTIONS**

44542 None.

44543 **SEE ALSO**44544 *strcmp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <string.h>44545 **CHANGE HISTORY**

44546 First released in Issue 1. Derived from Issue 1 of the SVID.

44547 **Issue 6**

44548 Extensions beyond the ISO C standard are marked.

44549 **NAME**

44550           strncpy — copy part of a string

44551 **SYNOPSIS**

44552           #include &lt;string.h&gt;

44553           char \*strncpy(char \*restrict *s1*, const char \*restrict *s2*, size\_t *n*);44554 **DESCRIPTION**

44555 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
44556 conflict between the requirements described here and the ISO C standard is unintentional. This  
44557 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44558       The *strncpy()* function shall copy not more than *n* bytes (bytes that follow a null byte are not  
44559 copied) from the array pointed to by *s2* to the array pointed to by *s1*. If copying takes place  
44560 between objects that overlap, the behavior is undefined.

44561       If the array pointed to by *s2* is a string that is shorter than *n* bytes, null bytes shall be appended  
44562 to the copy in the array pointed to by *s1*, until *n* bytes in all are written.

44563 **RETURN VALUE**44564       The *strncpy()* function shall return *s1*; no return value is reserved to indicate an error.44565 **ERRORS**

44566       No errors are defined.

44567 **EXAMPLES**

44568       None.

44569 **APPLICATION USAGE**

44570       Character movement is performed differently in different implementations. Thus, overlapping  
44571 moves may yield surprises.

44572       If there is no null byte in the first *n* bytes of the array pointed to by *s2*, the result is not null-  
44573 terminated.

44574 **RATIONALE**

44575       None.

44576 **FUTURE DIRECTIONS**

44577       None.

44578 **SEE ALSO**44579       *strcpy()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>44580 **CHANGE HISTORY**

44581       First released in Issue 1. Derived from Issue 1 of the SVID.

44582 **Issue 6**44583       The *strncpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

44584 **NAME**

44585 strpbrk — scan a string for a byte

44586 **SYNOPSIS**

44587 #include &lt;string.h&gt;

44588 char \*strpbrk(const char \*s1, const char \*s2);

44589 **DESCRIPTION**

44590 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
44591 conflict between the requirements described here and the ISO C standard is unintentional. This  
44592 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44593 The *strpbrk()* function shall locate the first occurrence in the string pointed to by *s1* of any byte  
44594 from the string pointed to by *s2*.

44595 **RETURN VALUE**

44596 Upon successful completion, *strpbrk()* shall return a pointer to the byte or a null pointer if no  
44597 byte from *s2* occurs in *s1*.

44598 **ERRORS**

44599 No errors are defined.

44600 **EXAMPLES**

44601 None.

44602 **APPLICATION USAGE**

44603 None.

44604 **RATIONALE**

44605 None.

44606 **FUTURE DIRECTIONS**

44607 None.

44608 **SEE ALSO**44609 *strchr()*, *strchr()*, the Base Definitions volume of IEEE Std 1003.1-2001, <string.h>44610 **CHANGE HISTORY**

44611 First released in Issue 1. Derived from Issue 1 of the SVID.

## 44612 NAME

44613 strptime — date and time conversion

## 44614 SYNOPSIS

44615 XSI 

```
#include <time.h>
```

44616 

```
char *strptime(const char *restrict buf, const char *restrict format,
```

  
44617 

```
struct tm *restrict tm);
```

44618

## 44619 DESCRIPTION

44620 The *strptime()* function shall convert the character string pointed to by *buf* to values which are  
44621 stored in the **tm** structure pointed to by *tm*, using the format specified by *format*.44622 The *format* is composed of zero or more directives. Each directive is composed of one of the  
44623 following: one or more white-space characters (as specified by *isspace()*); an ordinary character  
44624 (neither '%' nor a white-space character); or a conversion specification. Each conversion  
44625 specification is composed of a '%' character followed by a conversion character which specifies  
44626 the replacement required. The application shall ensure that there is white-space or other non-  
44627 alphanumeric characters between any two conversion specifications. The following conversion  
44628 specifications are supported:44629 %a The day of the week, using the locale's weekday names; either the abbreviated or full  
44630 name may be specified.

44631 %A Equivalent to %a.

44632 %b The month, using the locale's month names; either the abbreviated or full name may be  
44633 specified.

44634 %B Equivalent to %b.

44635 %c Replaced by the locale's appropriate date and time representation.

44636 %C The century number [00,99]; leading zeros are permitted but not required.

44637 %d The day of the month [01,31]; leading zeros are permitted but not required.

44638 %D The date as %m/%d/%y.

44639 %e Equivalent to %d.

44640 %h Equivalent to %b.

44641 %H The hour (24-hour clock) [00,23]; leading zeros are permitted but not required.

44642 %I The hour (12-hour clock) [01,12]; leading zeros are permitted but not required.

44643 %j The day number of the year [001,366]; leading zeros are permitted but not required.

44644 %m The month number [01,12]; leading zeros are permitted but not required.

44645 %M The minute [00,59]; leading zeros are permitted but not required.

44646 %n Any white space.

44647 %p The locale's equivalent of a.m or p.m.

44648 %r 12-hour clock time using the AM/PM notation if **t\_fmt\_ampm** is not an empty string in  
44649 the LC\_TIME portion of the current locale; in the POSIX locale, this shall be equivalent  
44650 to %I:%M:%S %p.

44651 %R The time as %H:%M.

|       |              |                                                                                                                                                                                                                                                                                       |
|-------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 44652 | %S           | The seconds [00,60]; leading zeros are permitted but not required.                                                                                                                                                                                                                    |
| 44653 | %t           | Any white space.                                                                                                                                                                                                                                                                      |
| 44654 | %T           | The time as %H:%M:%S.                                                                                                                                                                                                                                                                 |
| 44655 | %U           | The week number of the year (Sunday as the first day of the week) as a decimal number [00,53]; leading zeros are permitted but not required.                                                                                                                                          |
| 44656 |              |                                                                                                                                                                                                                                                                                       |
| 44657 | %w           | The weekday as a decimal number [0,6], with 0 representing Sunday; leading zeros are permitted but not required.                                                                                                                                                                      |
| 44658 |              |                                                                                                                                                                                                                                                                                       |
| 44659 | %W           | The week number of the year (Monday as the first day of the week) as a decimal number [00,53]; leading zeros are permitted but not required.                                                                                                                                          |
| 44660 |              |                                                                                                                                                                                                                                                                                       |
| 44661 | %x           | The date, using the locale's date format.                                                                                                                                                                                                                                             |
| 44662 | %X           | The time, using the locale's time format.                                                                                                                                                                                                                                             |
| 44663 | %y           | The year within century. When a century is not otherwise specified, values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive; leading zeros shall be permitted but shall not be required. |
| 44664 |              |                                                                                                                                                                                                                                                                                       |
| 44665 |              |                                                                                                                                                                                                                                                                                       |
| 44666 |              |                                                                                                                                                                                                                                                                                       |
| 44667 | <b>Note:</b> | It is expected that in a future version of IEEE Std 1003.1-2001 the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)                                                                                   |
| 44668 |              |                                                                                                                                                                                                                                                                                       |
| 44669 |              |                                                                                                                                                                                                                                                                                       |
| 44670 | %Y           | The year, including the century (for example, 1988).                                                                                                                                                                                                                                  |
| 44671 | %%           | Replaced by %.                                                                                                                                                                                                                                                                        |

#### 44672 **Modified Conversion Specifiers**

Some conversion specifiers can be modified by the **E** and **O** modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist in the current locale, the behavior shall be as if the unmodified conversion specification were used.

|       |     |                                                                                                                    |
|-------|-----|--------------------------------------------------------------------------------------------------------------------|
| 44673 |     |                                                                                                                    |
| 44674 |     |                                                                                                                    |
| 44675 |     |                                                                                                                    |
| 44676 |     |                                                                                                                    |
| 44677 | %Ec | The locale's alternative appropriate date and time representation.                                                 |
| 44678 | %EC | The name of the base year (period) in the locale's alternative representation.                                     |
| 44679 | %Ex | The locale's alternative date representation.                                                                      |
| 44680 | %EX | The locale's alternative time representation.                                                                      |
| 44681 | %Ey | The offset from %EC (year only) in the locale's alternative representation.                                        |
| 44682 | %EY | The full alternative year representation.                                                                          |
| 44683 | %Od | The day of the month using the locale's alternative numeric symbols; leading zeros are permitted but not required. |
| 44684 |     |                                                                                                                    |
| 44685 | %Oe | Equivalent to %Od.                                                                                                 |
| 44686 | %OH | The hour (24-hour clock) using the locale's alternative numeric symbols.                                           |
| 44687 | %OI | The hour (12-hour clock) using the locale's alternative numeric symbols.                                           |
| 44688 | %Om | The month using the locale's alternative numeric symbols.                                                          |
| 44689 | %OM | The minutes using the locale's alternative numeric symbols.                                                        |

- 44690        %OS        The seconds using the locale's alternative numeric symbols.
- 44691        %OU        The week number of the year (Sunday as the first day of the week) using the locale's  
44692 alternative numeric symbols.
- 44693        %Ow        The number of the weekday (Sunday=0) using the locale's alternative numeric symbols.
- 44694        %OW        The week number of the year (Monday as the first day of the week) using the locale's  
44695 alternative numeric symbols.
- 44696        %Oy        The year (offset from %C) using the locale's alternative numeric symbols.
- 44697        A conversion specification composed of white-space characters is executed by scanning input  
44698 up to the first character that is not white-space (which remains unscanned), or until no more  
44699 characters can be scanned.
- 44700        A conversion specification that is an ordinary character is executed by scanning the next  
44701 character from the buffer. If the character scanned from the buffer differs from the one  
44702 comprising the directive, the directive fails, and the differing and subsequent characters remain  
44703 unscanned.
- 44704        A series of conversion specifications composed of %n, %t, white-space characters, or any  
44705 combination is executed by scanning up to the first character that is not white space (which  
44706 remains unscanned), or until no more characters can be scanned.
- 44707        Any other conversion specification is executed by scanning characters until a character matching  
44708 the next directive is scanned, or until no more characters can be scanned. These characters,  
44709 except the one matching the next directive, are then compared to the locale values associated  
44710 with the conversion specifier. If a match is found, values for the appropriate **tm** structure  
44711 members are set to values corresponding to the locale information. Case is ignored when  
44712 matching items in *buf* such as month or weekday names. If no match is found, *strptime()* fails  
44713 and no more characters are scanned.
- 44714 **RETURN VALUE**
- 44715        Upon successful completion, *strptime()* shall return a pointer to the character following the last  
44716 character parsed. Otherwise, a null pointer shall be returned.
- 44717 **ERRORS**
- 44718        No errors are defined.
- 44719 **EXAMPLES**
- 44720        None.
- 44721 **APPLICATION USAGE**
- 44722        Several "equivalent to" formats and the special processing of white-space characters are  
44723 provided in order to ease the use of identical *format* strings for *strftime()* and *strptime()*.
- 44724        Applications should use %Y (4-digit years) in preference to %y (2-digit years).
- 44725        It is unspecified whether multiple calls to *strptime()* using the same **tm** structure will update the  
44726 current contents of the structure or overwrite all contents of the structure. Conforming  
44727 applications should make a single call to *strptime()* with a format and all data needed to  
44728 completely specify the date and time being converted.
- 44729 **RATIONALE**
- 44730        None.

44731 **FUTURE DIRECTIONS**

44732           The *strptime()* function is expected to be mandatory in the next version of this volume of  
44733           IEEE Std 1003.1-2001.

44734 **SEE ALSO**

44735           *scanf()*, *strptime()*, *time()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**time.h**>

44736 **CHANGE HISTORY**

44737           First released in Issue 4.

44738 **Issue 5**

44739           Moved from ENHANCED I18N to BASE.

44740           The [ENOSYS] error is removed.

44741           The exact meaning of the %y and %Oy specifiers is clarified in the DESCRIPTION.

44742 **Issue 6**

44743           The Open Group Corrigendum U033/5 is applied. The %r specifier description is reworded.

44744           The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

44745           The **restrict** keyword is added to the *strptime()* prototype for alignment with the  
44746           ISO/IEC 9899:1999 standard.

44747           The Open Group Corrigendum U047/2 is applied.

44748           The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion  
44749           specification” for consistency with *strptime()*.

44750 **NAME**

44751 strrchr — string scanning operation

44752 **SYNOPSIS**

44753 #include &lt;string.h&gt;

44754 char \*strrchr(const char \*s, int c);

44755 **DESCRIPTION**

44756 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
44757 conflict between the requirements described here and the ISO C standard is unintentional. This  
44758 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44759 The *strrchr()* function shall locate the last occurrence of *c* (converted to a **char**) in the string  
44760 pointed to by *s*. The terminating null byte is considered to be part of the string.

44761 **RETURN VALUE**

44762 Upon successful completion, *strrchr()* shall return a pointer to the byte or a null pointer if *c* does  
44763 not occur in the string.

44764 **ERRORS**

44765 No errors are defined.

44766 **EXAMPLES**44767 **Finding the Base Name of a File**

44768 The following example uses *strrchr()* to get a pointer to the base name of a file. The *strrchr()*  
44769 function searches backwards through the name of the file to find the last '/' character in *name*.  
44770 This pointer (plus one) will point to the base name of the file.

```
44771 #include <string.h>
44772 ...
44773 const char *name;
44774 char *basename;
44775 ...
44776 basename = strrchr(name, '/') + 1;
44777 ...
```

44778 **APPLICATION USAGE**

44779 None.

44780 **RATIONALE**

44781 None.

44782 **FUTURE DIRECTIONS**

44783 None.

44784 **SEE ALSO**44785 *strchr()*, the Base Definitions volume of IEEE Std 1003.1-2001, <string.h>44786 **CHANGE HISTORY**

44787 First released in Issue 1. Derived from Issue 1 of the SVID.

44788 **NAME**

44789 strspn — get length of a substring

44790 **SYNOPSIS**

44791 #include &lt;string.h&gt;

44792 size\_t strspn(const char \*s1, const char \*s2);

44793 **DESCRIPTION**

44794 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
44795 conflict between the requirements described here and the ISO C standard is unintentional. This  
44796 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44797 The *strspn()* function shall compute the length (in bytes) of the maximum initial segment of the  
44798 string pointed to by *s1* which consists entirely of bytes from the string pointed to by *s2*.

44799 **RETURN VALUE**

44800 The *strspn()* function shall return the length of *s1*; no return value is reserved to indicate an  
44801 error.

44802 **ERRORS**

44803 No errors are defined.

44804 **EXAMPLES**

44805 None.

44806 **APPLICATION USAGE**

44807 None.

44808 **RATIONALE**

44809 None.

44810 **FUTURE DIRECTIONS**

44811 None.

44812 **SEE ALSO**44813 *strcspn()*, the Base Definitions volume of IEEE Std 1003.1-2001, <string.h>44814 **CHANGE HISTORY**

44815 First released in Issue 1. Derived from Issue 1 of the SVID.

44816 **Issue 5**

44817 The RETURN VALUE section is updated to indicate that *strspn()* returns the length of *s*, and not  
44818 *s* itself as was previously stated.

44819 **NAME**

44820        strstr — find a substring

44821 **SYNOPSIS**

44822        #include &lt;string.h&gt;

44823        char \*strstr(const char \*s1, const char \*s2);

44824 **DESCRIPTION**

44825 cx        The functionality described on this reference page is aligned with the ISO C standard. Any  
44826        conflict between the requirements described here and the ISO C standard is unintentional. This  
44827        volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44828        The *strstr()* function shall locate the first occurrence in the string pointed to by *s1* of the  
44829        sequence of bytes (excluding the terminating null byte) in the string pointed to by *s2*.

44830 **RETURN VALUE**

44831        Upon successful completion, *strstr()* shall return a pointer to the located string or a null pointer  
44832        if the string is not found.

44833        If *s2* points to a string with zero length, the function shall return *s1*.

44834 **ERRORS**

44835        No errors are defined.

44836 **EXAMPLES**

44837        None.

44838 **APPLICATION USAGE**

44839        None.

44840 **RATIONALE**

44841        None.

44842 **FUTURE DIRECTIONS**

44843        None.

44844 **SEE ALSO**44845        *strchr()*, the Base Definitions volume of IEEE Std 1003.1-2001, <string.h>44846 **CHANGE HISTORY**

44847        First released in Issue 3. Included for alignment with the ANSI C standard.

44848 **NAME**

44849 strtod, strtodf, strtold — convert a string to a double-precision number

44850 **SYNOPSIS**

44851 #include &lt;stdlib.h&gt;

44852 double strtod(const char \*restrict *nptr*, char \*\*restrict *endptr*);44853 float strtodf(const char \*restrict *nptr*, char \*\*restrict *endptr*);44854 long double strtold(const char \*restrict *nptr*, char \*\*restrict *endptr*);44855 **DESCRIPTION**

44856 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 44857 conflict between the requirements described here and the ISO C standard is unintentional. This  
 44858 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

44859 These functions shall convert the initial portion of the string pointed to by *nptr* to **double**, **float**,  
 44860 and **long double** representation, respectively. First, they decompose the input string into three  
 44861 parts:

- 44862 1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
- 44863 2. A subject sequence interpreted as a floating-point constant or representing infinity or NaN
- 44864 3. A final string of one or more unrecognized characters, including the terminating null byte  
 44865 of the input string

44866 Then they shall attempt to convert the subject sequence to a floating-point number, and return  
 44867 the result.

44868 The expected form of the subject sequence is an optional plus or minus sign, then one of the  
 44869 following:

- 44870 • A non-empty sequence of decimal digits optionally containing a radix character, then an  
 44871 optional exponent part
- 44872 • A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix  
 44873 character, then an optional binary exponent part
- 44874 • One of INF or INFINITY, ignoring case
- 44875 • One of NAN or NAN(*n-char-sequence<sub>opt</sub>*), ignoring case in the NAN part, where:

```

44876 n-char-sequence:
44877 digit
44878 nondigit
44879 n-char-sequence digit
44880 n-char-sequence nondigit

```

44881 The subject sequence is defined as the longest initial subsequence of the input string, starting  
 44882 with the first non-white-space character, that is of the expected form. The subject sequence  
 44883 contains no characters if the input string is not of the expected form.

44884 If the subject sequence has the expected form for a floating-point number, the sequence of  
 44885 characters starting with the first digit or the decimal-point character (whichever occurs first)  
 44886 shall be interpreted as a floating constant of the C language, except that the radix character shall  
 44887 be used in place of a period, and that if neither an exponent part nor a radix character appears in  
 44888 a decimal floating-point number, or if a binary exponent part does not appear in a hexadecimal  
 44889 floating-point number, an exponent part of the appropriate type with value zero is assumed to  
 44890 follow the last digit in the string. If the subject sequence begins with a minus sign, the sequence  
 44891 shall be interpreted as negated. A character sequence INF or INFINITY shall be interpreted as an

44892 infinity, if representable in the return type, else as if it were a floating constant that is too large  
 44893 for the range of the return type. A character sequence NAN or NAN(*n-char-sequence<sub>opt</sub>*) shall be  
 44894 interpreted as a quiet NaN, if supported in the return type, else as if it were a subject sequence  
 44895 part that does not have the expected form; the meaning of the *n-char* sequences is  
 44896 implementation-defined. A pointer to the final string is stored in the object pointed to by *endptr*,  
 44897 provided that *endptr* is not a null pointer.

44898 If the subject sequence has the hexadecimal form and FLT\_RADIX is a power of 2, the value  
 44899 resulting from the conversion is correctly rounded.

44900 CX The radix character is defined in the program's locale (category *LC\_NUMERIC*). In the POSIX  
 44901 locale, or in a locale where the radix character is not defined, the radix character shall default to a  
 44902 period ('.').

44903 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be  
 44904 accepted.

44905 If the subject sequence is empty or does not have the expected form, no conversion shall be  
 44906 performed; the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not  
 44907 a null pointer.

44908 CX The *strtod()* function shall not change the setting of *errno* if successful.

44909 Since 0 is returned on error and is also a valid return on success, an application wishing to check  
 44910 for error situations should set *errno* to 0, then call *strtod()*, *strtof()*, or *strtold()*, then check *errno*.

#### 44911 RETURN VALUE

44912 Upon successful completion, these functions shall return the converted value. If no conversion  
 44913 could be performed, 0 shall be returned, and *errno* may be set to [EINVAL].

44914 If the correct value is outside the range of representable values, ±HUGE\_VAL, ±HUGE\_VALF, or |  
 44915 ±HUGE\_VALL shall be returned (according to the sign of the value), and *errno* shall be set to |  
 44916 [ERANGE].

44917 If the correct value would cause an underflow, a value whose magnitude is no greater than the  
 44918 smallest normalized positive number in the return type shall be returned and *errno* set to  
 44919 [ERANGE].

#### 44920 ERRORS

44921 These functions shall fail if:

44922 CX [ERANGE] The value to be returned would cause overflow or underflow.

44923 These functions may fail if:

44924 CX [EINVAL] No conversion could be performed.

#### 44925 EXAMPLES

44926 None.

#### 44927 APPLICATION USAGE

44928 If the subject sequence has the hexadecimal form and FLT\_RADIX is not a power of 2, and the  
 44929 result is not exactly representable, the result should be one of the two numbers in the  
 44930 appropriate internal format that are adjacent to the hexadecimal floating source value, with the  
 44931 extra stipulation that the error should have a correct sign for the current rounding direction.

44932 If the subject sequence has the decimal form and at most DECIMAL\_DIG (defined in <float.h>)  
 44933 significant digits, the result should be correctly rounded. If the subject sequence *D* has the  
 44934 decimal form and more than DECIMAL\_DIG significant digits, consider the two bounding,  
 44935 adjacent decimal strings *L* and *U*, both having DECIMAL\_DIG significant digits, such that the

44936 values of  $L$ ,  $D$ , and  $U$  satisfy  $L \leq D \leq U$ . The result should be one of the (equal or adjacent)  
 44937 values that would be obtained by correctly rounding  $L$  and  $U$  according to the current rounding  
 44938 direction, with the extra stipulation that the error with respect to  $D$  should have a correct sign  
 44939 for the current rounding direction.

44940 The changes to `strtod()` introduced by the ISO/IEC 9899:1999 standard can alter the behavior of  
 44941 well-formed applications complying with the ISO/IEC 9899:1990 standard and thus earlier  
 44942 versions of the base documents. One such example would be:

```

44943 int
44944 what_kind_of_number (char *s)
44945 {
44946 char *endp;
44947 double d;
44948 long l;

44949 d = strtod(s, &endp);
44950 if (s != endp && *endp == '\0')
44951 printf("It's a float with value %g\n", d);
44952 else
44953 {
44954 l = strtol(s, &endp, 0);
44955 if (s != endp && *endp == '\0')
44956 printf("It's an integer with value %ld\n", l);
44957 else
44958 return 1;
44959 }
44960 return 0;
44961 }

```

44962 If the function is called with:

```
44963 what_kind_of_number ("0x10")
```

44964 an ISO/IEC 9899:1990 standard-compliant library will result in the function printing:

```
44965 It's an integer with value 16
```

44966 With the ISO/IEC 9899:1999 standard, the result is:

```
44967 It's a float with value 16
```

44968 The change in behavior is due to the inclusion of floating-point numbers in hexadecimal  
 44969 notation without requiring that either a decimal point or the binary exponent be present.

#### 44970 RATIONALE

44971 None.

#### 44972 FUTURE DIRECTIONS

44973 None.

#### 44974 SEE ALSO

44975 `isspace()`, `localeconv()`, `scanf()`, `setlocale()`, `strtol()`, the Base Definitions volume of  
 44976 IEEE Std 1003.1-2001, Chapter 7, Locale, `<float.h>`, `<stdlib.h>`

#### 44977 CHANGE HISTORY

44978 First released in Issue 1. Derived from Issue 1 of the SVID.

44979 **Issue 5**

44980 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

44981 **Issue 6**

44982 Extensions beyond the ISO C standard are marked.

44983 The following new requirements on POSIX implementations derive from alignment with the  
44984 Single UNIX Specification:

- 44985 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
44986 added if no conversion could be performed.

44987 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 44988 • The *strtod()* function is updated.
- 44989 • The *strtof()* and *strtold()* functions are added.
- 44990 • The DESCRIPTION is extensively revised.

44991 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

44992 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/61 is applied, correcting the second |  
44993 paragraph in the RETURN VALUE section. This change clarifies the sign of the return value. |

44994 **NAME**

44995 strtoimax, strtoumax — convert string to integer type

44996 **SYNOPSIS**

44997 #include <inttypes.h>

44998 intmax\_t strtoimax(const char \*restrict nptr, char \*\*restrict endptr,  
44999 int base);

45000 uintmax\_t strtoumax(const char \*restrict nptr, char \*\*restrict endptr,  
45001 int base);

45002 **DESCRIPTION**

45003 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
45004 conflict between the requirements described here and the ISO C standard is unintentional. This  
45005 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

45006 These functions shall be equivalent to the *strtol()*, *strtoll()*, *strtoul()*, and *strtoull()* functions,  
45007 except that the initial portion of the string shall be converted to **intmax\_t** and **uintmax\_t**  
45008 representation, respectively.

45009 **RETURN VALUE**

45010 These functions shall return the converted value, if any.

45011 If no conversion could be performed, zero shall be returned.

45012 If the correct value is outside the range of representable values, {INTMAX\_MAX},  
45013 {INTMAX\_MIN}, or {UINTMAX\_MAX} shall be returned (according to the return type and sign  
45014 of the value, if any), and *errno* shall be set to [ERANGE].

45015 **ERRORS**

45016 These functions shall fail if:

45017 [ERANGE] The value to be returned is not representable.

45018 These functions may fail if:

45019 [EINVAL] The value of *base* is not supported.

45020 **EXAMPLES**

45021 None.

45022 **APPLICATION USAGE**

45023 None.

45024 **RATIONALE**

45025 None.

45026 **FUTURE DIRECTIONS**

45027 None.

45028 **SEE ALSO**

45029 *strtol()*, *strtoul()*, the Base Definitions volume of IEEE Std 1003.1-2001, <inttypes.h>

45030 **CHANGE HISTORY**

45031 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

## 45032 NAME

45033 strtok, strtok\_r — split string into tokens

## 45034 SYNOPSIS

45035 #include &lt;string.h&gt;

```
45036 char *strtok(char *restrict s1, const char *restrict s2);
45037 TSF char *strtok_r(char *restrict s, const char *restrict sep,
45038 char **restrict lasts);
45039
```

## 45040 DESCRIPTION

45041 CX For *strtok()*: The functionality described on this reference page is aligned with the ISO C  
 45042 standard. Any conflict between the requirements described here and the ISO C standard is  
 45043 unintentional. This volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

45044 A sequence of calls to *strtok()* breaks the string pointed to by *s1* into a sequence of tokens, each  
 45045 of which is delimited by a byte from the string pointed to by *s2*. The first call in the sequence has  
 45046 *s1* as its first argument, and is followed by calls with a null pointer as their first argument. The  
 45047 separator string pointed to by *s2* may be different from call to call.

45048 The first call in the sequence searches the string pointed to by *s1* for the first byte that is *not*  
 45049 contained in the current separator string pointed to by *s2*. If no such byte is found, then there  
 45050 are no tokens in the string pointed to by *s1* and *strtok()* shall return a null pointer. If such a byte  
 45051 is found, it is the start of the first token.

45052 The *strtok()* function then searches from there for a byte that *is* contained in the current  
 45053 separator string. If no such byte is found, the current token extends to the end of the string  
 45054 pointed to by *s1*, and subsequent searches for a token shall return a null pointer. If such a byte  
 45055 is found, it is overwritten by a null byte, which terminates the current token. The *strtok()* function  
 45056 saves a pointer to the following byte, from which the next search for a token shall start.

45057 Each subsequent call, with a null pointer as the value of the first argument, starts searching from  
 45058 the saved pointer and behaves as described above.

45059 The implementation shall behave as if no function defined in this volume of  
 45060 IEEE Std 1003.1-2001 calls *strtok()*.

45061 CX The *strtok()* function need not be reentrant. A function that is not required to be reentrant is not  
 45062 required to be thread-safe.

45063 TSF The *strtok\_r()* function considers the null-terminated string *s* as a sequence of zero or more text  
 45064 tokens separated by spans of one or more characters from the separator string *sep*. The  
 45065 argument *lasts* points to a user-provided pointer which points to stored information necessary  
 45066 for *strtok\_r()* to continue scanning the same string.

45067 In the first call to *strtok\_r()*, *s* points to a null-terminated string, *sep* to a null-terminated string of  
 45068 separator characters, and the value pointed to by *lasts* is ignored. The *strtok\_r()* function shall  
 45069 return a pointer to the first character of the first token, write a null character into *s* immediately  
 45070 following the returned token, and update the pointer to which *lasts* points.

45071 In subsequent calls, *s* is a NULL pointer and *lasts* shall be unchanged from the previous call so  
 45072 that subsequent calls shall move through the string *s*, returning successive tokens until no  
 45073 tokens remain. The separator string *sep* may be different from call to call. When no token  
 45074 remains in *s*, a NULL pointer shall be returned.

45075 **RETURN VALUE**

45076 Upon successful completion, *strtok()* shall return a pointer to the first byte of a token. Otherwise,  
45077 if there is no token, *strtok()* shall return a null pointer.

45078 TSF The *strtok\_r()* function shall return a pointer to the token found, or a NULL pointer when no  
45079 token is found.

45080 **ERRORS**

45081 No errors are defined.

45082 **EXAMPLES**45083 **Searching for Word Separators**

45084 The following example searches for tokens separated by <space>s.

```
45085 #include <string.h>
45086 ...
45087 char *token;
45088 char *line = "LINE TO BE SEPARATED";
45089 char *search = " ";

45090 /* Token will point to "LINE". */
45091 token = strtok(line, search);

45092 /* Token will point to "TO". */
45093 token = strtok(NULL, search);
```

45094 **Breaking a Line**

45095 The following example uses *strtok()* to break a line into two character strings separated by any  
45096 combination of <space>s, <tab>s, or <newline>s.

```
45097 #include <string.h>
45098 ...
45099 struct element {
45100 char *key;
45101 char *data;
45102 };
45103 ...
45104 char line[LINE_MAX];
45105 char *key, *data;
45106 ...
45107 key = strtok(line, " \n");
45108 data = strtok(NULL, " \n");
45109 ...
```

45110 **APPLICATION USAGE**

45111 The *strtok\_r()* function is thread-safe and stores its state in a user-supplied buffer instead of  
45112 possibly using a static data area that may be overwritten by an unrelated call from another  
45113 thread.

45114 **RATIONALE**

45115 The *strtok()* function searches for a separator string within a larger string. It returns a pointer to  
45116 the last substring between separator strings. This function uses static storage to keep track of  
45117 the current string position between calls. The new function, *strtok\_r()*, takes an additional  
45118 argument, *lasts*, to keep track of the current position in the string.

45119 **FUTURE DIRECTIONS**

45120 None.

45121 **SEE ALSO**45122 The Base Definitions volume of IEEE Std 1003.1-2001, <**string.h**>45123 **CHANGE HISTORY**

45124 First released in Issue 1. Derived from Issue 1 of the SVID.

45125 **Issue 5**45126 The *strtok\_r()* function is included for alignment with the POSIX Threads Extension.45127 A note indicating that the *strtok()* function need not be reentrant is added to the DESCRIPTION.45128 **Issue 6**

45129 Extensions beyond the ISO C standard are marked.

45130 The *strtok\_r()* function is marked as part of the Thread-Safe Functions option.

45131 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

45132 The APPLICATION USAGE section is updated to include a note on the thread-safe function and

45133 its avoidance of possibly using a static data area.

45134 The **restrict** keyword is added to the *strtok()* and *strtok\_r()* prototypes for alignment with the

45135 ISO/IEC 9899:1999 standard.

## 45136 NAME

45137 strtol, strtoll — convert a string to a long integer

## 45138 SYNOPSIS

45139 #include &lt;stdlib.h&gt;

45140 long strtol(const char \*restrict *str*, char \*\*restrict *endptr*, int *base*);45141 long long strtoll(const char \*restrict *str*, char \*\*restrict *endptr*,45142 int *base*)

## 45143 DESCRIPTION

45144 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 45145 conflict between the requirements described here and the ISO C standard is unintentional. This  
 45146 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

45147 These functions shall convert the initial portion of the string pointed to by *str* to a type **long** and  
 45148 **long long** representation, respectively. First, they decompose the input string into three parts:

- 45149 1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
- 45150 2. A subject sequence interpreted as an integer represented in some radix determined by the  
 45151 value of *base*
- 45152 3. A final string of one or more unrecognized characters, including the terminating null byte  
 45153 of the input string.

45154 Then they shall attempt to convert the subject sequence to an integer, and return the result.

45155 If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant,  
 45156 octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A  
 45157 decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An  
 45158 octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to  
 45159 '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the  
 45160 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

45161 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence  
 45162 of letters and digits representing an integer with the radix specified by *base*, optionally preceded  
 45163 by a '+' or '-' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the  
 45164 values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the  
 45165 value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and  
 45166 digits, following the sign if present.

45167 The subject sequence is defined as the longest initial subsequence of the input string, starting  
 45168 with the first non-white-space character that is of the expected form. The subject sequence shall  
 45169 contain no characters if the input string is empty or consists entirely of white-space characters,  
 45170 or if the first non-white-space character is other than a sign or a permissible letter or digit.

45171 If the subject sequence has the expected form and the value of *base* is 0, the sequence of  
 45172 characters starting with the first digit shall be interpreted as an integer constant. If the subject  
 45173 sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the  
 45174 base for conversion, ascribing to each letter its value as given above. If the subject sequence  
 45175 begins with a minus sign, the value resulting from the conversion shall be negated. A pointer to  
 45176 the final string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null  
 45177 pointer.

45178 cx In other than the C or POSIX locales, other implementation-defined subject sequences may be  
 45179 accepted.

45180 If the subject sequence is empty or does not have the expected form, no conversion is performed;  
 45181 the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null  
 45182 pointer.

45183 CX The *strtol()* function shall not change the setting of *errno* if successful.

45184 Since 0, {LONG\_MIN} or {LLONG\_MIN}, and {LONG\_MAX} or {LLONG\_MAX} are returned on  
 45185 error and are also valid returns on success, an application wishing to check for error situations  
 45186 should set *errno* to 0, then call *strtol()* or *strtoll()*, then check *errno*.

#### 45187 RETURN VALUE

45188 Upon successful completion, these functions shall return the converted value, if any. If no  
 45189 CX conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

45190 If the correct value is outside the range of representable values, {LONG\_MIN}, {LONG\_MAX},  
 45191 {LLONG\_MIN}, or {LLONG\_MAX} shall be returned (according to the sign of the value), and  
 45192 *errno* set to [ERANGE].

#### 45193 ERRORS

45194 These functions shall fail if:

45195 [ERANGE] The value to be returned is not representable.

45196 These functions may fail if:

45197 CX [EINVAL] The value of *base* is not supported.

#### 45198 EXAMPLES

45199 None.

#### 45200 APPLICATION USAGE

45201 None.

#### 45202 RATIONALE

45203 None.

#### 45204 FUTURE DIRECTIONS

45205 None.

#### 45206 SEE ALSO

45207 *isalpha()*, *scanf()*, *strtod()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>

#### 45208 CHANGE HISTORY

45209 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 45210 Issue 5

45211 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

#### 45212 Issue 6

45213 Extensions beyond the ISO C standard are marked.

45214 The following new requirements on POSIX implementations derive from alignment with the  
 45215 Single UNIX Specification:

45216 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
 45217 added if no conversion could be performed.

45218 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

45219 • The *strtol()* prototype is updated.

45220 • The *strtoll()* function is added.

45221 **NAME**

45222            **strtold** — convert a string to a double-precision number

45223 **SYNOPSIS**

45224            #include <stdlib.h>

45225            long double strtold(const char \*restrict *nptr*, char \*\*restrict *endptr*);

45226 **DESCRIPTION**

45227            Refer to *strtod*().

45228 **NAME**

45229            **strtoll** — convert a string to a long integer

45230 **SYNOPSIS**

45231            #include <stdlib.h>

45232            long long strtoll(const char \*restrict *str*, char \*\*restrict *endptr*,  
45233                            int *base*);

45234 **DESCRIPTION**

45235            Refer to *strtol*().

## 45236 NAME

45237 strtoul, strtoull — convert a string to an unsigned long

## 45238 SYNOPSIS

45239 #include &lt;stdlib.h&gt;

45240 unsigned long strtoul(const char \*restrict *str*,  
45241 char \*\*restrict *endptr*, int *base*);45242 unsigned long long strtoull(const char \*restrict *str*,  
45243 char \*\*restrict *endptr*, int *base*);

## 45244 DESCRIPTION

45245 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
45246 conflict between the requirements described here and the ISO C standard is unintentional. This  
45247 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.45248 These functions shall convert the initial portion of the string pointed to by *str* to a type **unsigned**  
45249 **long** and **unsigned long long** representation, respectively. First, they decompose the input  
45250 string into three parts:

- 45251 1. An initial, possibly empty, sequence of white-space characters (as specified by
- isspace()*
- )
- 
- 45252 2. A subject sequence interpreted as an integer represented in some radix determined by the
- 
- 45253 value of
- base*
- 
- 45254 3. A final string of one or more unrecognized characters, including the terminating null byte
- 
- 45255 of the input string

45256 Then they shall attempt to convert the subject sequence to an unsigned integer, and return the  
45257 result.45258 If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant,  
45259 octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A  
45260 decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An  
45261 octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to  
45262 '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the  
45263 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.45264 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence  
45265 of letters and digits representing an integer with the radix specified by *base*, optionally preceded  
45266 by a '+' or '-' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the  
45267 values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the  
45268 value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and  
45269 digits, following the sign if present.45270 The subject sequence is defined as the longest initial subsequence of the input string, starting  
45271 with the first non-white-space character that is of the expected form. The subject sequence shall  
45272 contain no characters if the input string is empty or consists entirely of white-space characters,  
45273 or if the first non-white-space character is other than a sign or a permissible letter or digit.45274 If the subject sequence has the expected form and the value of *base* is 0, the sequence of  
45275 characters starting with the first digit shall be interpreted as an integer constant. If the subject  
45276 sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the  
45277 base for conversion, ascribing to each letter its value as given above. If the subject sequence  
45278 begins with a minus sign, the value resulting from the conversion shall be negated. A pointer to  
45279 the final string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null  
45280 pointer.

45281 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be  
45282 accepted.

45283 If the subject sequence is empty or does not have the expected form, no conversion shall be  
45284 performed; the value of *str* shall be stored in the object pointed to by *endptr*, provided that *endptr*  
45285 is not a null pointer.

45286 CX The *strtoul()* function shall not change the setting of *errno* if successful.

45287 Since 0, {ULONG\_MAX}, and {ULLONG\_MAX} are returned on error and are also valid returns  
45288 on success, an application wishing to check for error situations should set *errno* to 0, then call  
45289 *strtoul()* or *strtoull()*, then check *errno*.

#### 45290 RETURN VALUE

45291 Upon successful completion, these functions shall return the converted value, if any. If no  
45292 CX conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL]. If the  
45293 correct value is outside the range of representable values, {ULONG\_MAX} or {ULLONG\_MAX}  
45294 shall be returned and *errno* set to [ERANGE].

#### 45295 ERRORS

45296 These functions shall fail if:

45297 CX [EINVAL] The value of *base* is not supported.

45298 [ERANGE] The value to be returned is not representable.

45299 These functions may fail if:

45300 CX [EINVAL] No conversion could be performed.

#### 45301 EXAMPLES

45302 None.

#### 45303 APPLICATION USAGE

45304 None.

#### 45305 RATIONALE

45306 None.

#### 45307 FUTURE DIRECTIONS

45308 None.

#### 45309 SEE ALSO

45310 *isalpha()*, *scanf()*, *strtod()*, *strtol()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
45311 <stdlib.h>

#### 45312 CHANGE HISTORY

45313 First released in Issue 4. Derived from the ANSI C standard.

#### 45314 Issue 5

45315 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

#### 45316 Issue 6

45317 Extensions beyond the ISO C standard are marked.

45318 The following new requirements on POSIX implementations derive from alignment with the  
45319 Single UNIX Specification:

- 45320 • The [EINVAL] error condition is added for when the value of *base* is not supported.

45321 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
45322 added if no conversion could be performed.

45323

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

45324

- The *strtoul()* prototype is updated.

45325

- The *strtoull()* function is added.

45326 **NAME**

45327        strtoumax — convert a string to an integer type

45328 **SYNOPSIS**

45329        #include &lt;inttypes.h&gt;

45330        uintmax\_t strtoumax(const char \*restrict *nptr*, char \*\*restrict *endptr*,  
45331                            int *base*);45332 **DESCRIPTION**45333        Refer to *strtoimax()*.

45334 **NAME**

45335 strxfrm — string transformation

45336 **SYNOPSIS**

45337 #include &lt;string.h&gt;

45338 size\_t strxfrm(char \*restrict *s1*, const char \*restrict *s2*, size\_t *n*);45339 **DESCRIPTION**45340 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
45341 conflict between the requirements described here and the ISO C standard is unintentional. This  
45342 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.45343 The *strxfrm()* function shall transform the string pointed to by *s2* and place the resulting string  
45344 into the array pointed to by *s1*. The transformation is such that if *strcmp()* is applied to two  
45345 transformed strings, it shall return a value greater than, equal to, or less than 0, corresponding to  
45346 the result of *strcoll()* applied to the same two original strings. No more than *n* bytes are placed  
45347 into the resulting array pointed to by *s1*, including the terminating null byte. If *n* is 0, *s1* is  
45348 permitted to be a null pointer. If copying takes place between objects that overlap, the behavior  
45349 is undefined.45350 CX The *strxfrm()* function shall not change the setting of *errno* if successful.45351 Since no return value is reserved to indicate an error, an application wishing to check for error  
45352 situations should set *errno* to 0, then call *strxfrm()*, then check *errno*.45353 **RETURN VALUE**45354 Upon successful completion, *strxfrm()* shall return the length of the transformed string (not  
45355 including the terminating null byte). If the value returned is *n* or more, the contents of the array  
45356 pointed to by *s1* are unspecified.45357 CX On error, *strxfrm()* may set *errno* but no return value is reserved to indicate an error.45358 **ERRORS**45359 The *strxfrm()* function may fail if:45360 CX [EINVAL] The string pointed to by the *s2* argument contains characters outside the  
45361 domain of the collating sequence.45362 **EXAMPLES**

45363 None.

45364 **APPLICATION USAGE**45365 The transformation function is such that two transformed strings can be ordered by *strcmp()* as  
45366 appropriate to collating sequence information in the program's locale (category *LC\_COLLATE*).45367 The fact that when *n* is 0 *s1* is permitted to be a null pointer is useful to determine the size of the  
45368 *s1* array prior to making the transformation.45369 **RATIONALE**

45370 None.

45371 **FUTURE DIRECTIONS**

45372 None.

45373 **SEE ALSO**45374 *strcmp()*, *strcoll()*, the Base Definitions volume of IEEE Std 1003.1-2001, <string.h>

45375 **CHANGE HISTORY**

45376 First released in Issue 3. Included for alignment with the ISO C standard.

45377 **Issue 5**

45378 The DESCRIPTION is updated to indicate that *errno* does not change if the function is  
45379 successful.

45380 **Issue 6**

45381 Extensions beyond the ISO C standard are marked.

45382 The following new requirements on POSIX implementations derive from alignment with the  
45383 Single UNIX Specification:

- 45384 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
45385 added if no conversion could be performed.

45386 The *strxfrm()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

45387 **NAME**

45388 swab — swap bytes

45389 **SYNOPSIS**45390 XSI 

```
#include <unistd.h>
```

45391 

```
void swab(const void *restrict src, void *restrict dest,
```

  
45392 

```
 ssize_t nbytes);
```

45393

45394 **DESCRIPTION**45395 The *swab()* function shall copy *nbytes* bytes, which are pointed to by *src*, to the object pointed to  
45396 by *dest*, exchanging adjacent bytes. The *nbytes* argument should be even. If *nbytes* is odd, *swab()*  
45397 copies and exchanges *nbytes*-1 bytes and the disposition of the last byte is unspecified. If  
45398 copying takes place between objects that overlap, the behavior is undefined. If *nbytes* is  
45399 negative, *swab()* does nothing.45400 **RETURN VALUE**

45401 None.

45402 **ERRORS**

45403 No errors are defined.

45404 **EXAMPLES**

45405 None.

45406 **APPLICATION USAGE**

45407 None.

45408 **RATIONALE**

45409 None.

45410 **FUTURE DIRECTIONS**

45411 None.

45412 **SEE ALSO**45413 The Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>45414 **CHANGE HISTORY**

45415 First released in Issue 1. Derived from Issue 1 of the SVID.

45416 **Issue 6**45417 The **restrict** keyword is added to the *swab()* prototype for alignment with the  
45418 ISO/IEC 9899:1999 standard.

45419 **NAME**

45420 swapcontext — swap user context

45421 **SYNOPSIS**

45422 XSI #include &lt;ucontext.h&gt;

45423 int swapcontext(ucontext\_t \*restrict oucp,  
45424 const ucontext\_t \*restrict ucp);

45425

45426 **DESCRIPTION**45427 Refer to *makecontext()*.

45428 **NAME**

45429       swprintf — print formatted wide-character output

45430 **SYNOPSIS**

45431       #include <stdio.h>

45432       #include <wchar.h>

45433       int swprintf(wchar\_t \*restrict *ws*, size\_t *n*,

45434               const wchar\_t \*restrict *format*, ...);

45435 **DESCRIPTION**

45436       Refer to *fwprintf()*.

45437 **NAME**

45438 swscanf — convert formatted wide-character input

45439 **SYNOPSIS**

45440 #include &lt;stdio.h&gt;

45441 #include &lt;wchar.h&gt;

45442 int swscanf(const wchar\_t \*restrict *ws*,45443 const wchar\_t \*restrict *format*, ... );45444 **DESCRIPTION**45445 Refer to *fwscanf()*.

45446 **NAME**45447 `symlink` — make a symbolic link to a file45448 **SYNOPSIS**45449 `#include <unistd.h>`45450 `int symlink(const char *path1, const char *path2);`45451 **DESCRIPTION**

45452 The `symlink()` function shall create a symbolic link called `path2` that contains the string pointed  
 45453 to by `path1` (`path2` is the name of the symbolic link created, `path1` is the string contained in the  
 45454 symbolic link).

45455 The string pointed to by `path1` shall be treated only as a character string and shall not be  
 45456 validated as a pathname.

45457 If the `symlink()` function fails for any reason other than [EIO], any file named by `path2` shall be  
 45458 unaffected.

45459 **RETURN VALUE**

45460 Upon successful completion, `symlink()` shall return 0; otherwise, it shall return -1 and set `errno` to  
 45461 indicate the error.

45462 **ERRORS**45463 The `symlink()` function shall fail if:

45464 [EACCES] Write permission is denied in the directory where the symbolic link is being  
 45465 created, or search permission is denied for a component of the path prefix of  
 45466 `path2`.

45467 [EEXIST] The `path2` argument names an existing file or symbolic link.

45468 [EIO] An I/O error occurs while reading from or writing to the file system.

45469 [ELOOP] A loop exists in symbolic links encountered during resolution of the `path2`  
 45470 argument.

45471 [ENAMETOOLONG]

45472 The length of the `path2` argument exceeds {PATH\_MAX} or a pathname  
 45473 component is longer than {NAME\_MAX} or the length of the `path1` argument  
 45474 is longer than {SYMLINK\_MAX}.

45475 [ENOENT] A component of `path2` does not name an existing file or `path2` is an empty  
 45476 string.

45477 [ENOSPC] The directory in which the entry for the new symbolic link is being placed  
 45478 cannot be extended because no space is left on the file system containing the  
 45479 directory, or the new symbolic link cannot be created because no space is left  
 45480 on the file system which shall contain the link, or the file system is out of file-  
 45481 allocation resources.

45482 [ENOTDIR] A component of the path prefix of `path2` is not a directory.

45483 [EROFS] The new symbolic link would reside on a read-only file system.

45484 The `symlink()` function may fail if:

45485 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 45486 resolution of the `path2` argument.

45487 [ENAMETOOLONG]

45488 As a result of encountering a symbolic link in resolution of the `path2`

45489 argument, the length of the substituted pathname string exceeded  
45490 {PATH\_MAX} bytes (including the terminating null byte), or the length of the  
45491 string pointed to by *path1* exceeded {SYMLINK\_MAX}.

**45492 EXAMPLES**

45493 None.

**45494 APPLICATION USAGE**

45495 Like a hard link, a symbolic link allows a file to have multiple logical names. The presence of a  
45496 hard link guarantees the existence of a file, even after the original name has been removed. A  
45497 symbolic link provides no such assurance; in fact, the file named by the *path1* argument need not  
45498 exist when the link is created. A symbolic link can cross file system boundaries.

45499 Normal permission checks are made on each component of the symbolic link pathname during  
45500 its resolution.

**45501 RATIONALE**

45502 Since IEEE Std 1003.1-2001 does not require any association of file times with symbolic links,  
45503 there is no requirement that file times be updated by *symlink()*.

**45504 FUTURE DIRECTIONS**

45505 None.

**45506 SEE ALSO**

45507 *lchown()*, *link()*, *lstat()*, *open()*, *readlink()*, *unlink()*, the Base Definitions volume of  
45508 IEEE Std 1003.1-2001, <**unistd.h**>

**45509 CHANGE HISTORY**

45510 First released in Issue 4, Version 2.

**45511 Issue 5**

45512 Moved from X/OPEN UNIX extension to BASE.

**45513 Issue 6**

45514 The following changes were made to align with the IEEE P1003.1a draft standard:

- 45515 • The DESCRIPTION text is updated.
- 45516 • The [ELOOP] optional error condition is added.

45517 **NAME**

45518           sync — schedule file system updates

45519 **SYNOPSIS**

45520 XSI       #include &lt;unistd.h&gt;

45521           void sync(void);

45522

45523 **DESCRIPTION**45524           The *sync()* function shall cause all information in memory that updates file systems to be  
45525           scheduled for writing out to all file systems.45526           The writing, although scheduled, is not necessarily complete upon return from *sync()*.45527 **RETURN VALUE**45528           The *sync()* function shall not return a value.45529 **ERRORS**

45530           No errors are defined.

45531 **EXAMPLES**

45532           None.

45533 **APPLICATION USAGE**

45534           None.

45535 **RATIONALE**

45536           None.

45537 **FUTURE DIRECTIONS**

45538           None.

45539 **SEE ALSO**45540           *fsync()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>45541 **CHANGE HISTORY**

45542           First released in Issue 4, Version 2.

45543 **Issue 5**

45544           Moved from X/OPEN UNIX extension to BASE.

45545 **NAME**

45546 sysconf — get configurable system variables

45547 **SYNOPSIS**

45548 #include &lt;unistd.h&gt;

45549 long sysconf(int name);

45550 **DESCRIPTION**

45551 The *sysconf()* function provides a method for the application to determine the current value of a  
 45552 configurable system limit or option (*variable*). The implementation shall support all of the  
 45553 variables listed in the following table and may support others.

45554 The *name* argument represents the system variable to be queried. The following table lists the  
 45555 minimal set of system variables from <limits.h> or <unistd.h> that can be returned by *sysconf()*,  
 45556 and the symbolic constants defined in <unistd.h> that are the corresponding values used for  
 45557 *name*.

45558

45559

45560

45561

45562

45563

45564

45565

45566

45567

45568

45569

45570

45571

45572

45573

45574

45575

45576

45577

45578

45579

45580

45581

45582

45583

45584

45585

45586

45587

45588

45589

45590

| Variable                                                                    | Value of Name          |
|-----------------------------------------------------------------------------|------------------------|
| {AIO_LISTIO_MAX}                                                            | _SC_AIO_LISTIO_MAX     |
| {AIO_MAX}                                                                   | _SC_AIO_MAX            |
| {AIO_PRIO_DELTA_MAX}                                                        | _SC_AIO_PRIO_DELTA_MAX |
| {ARG_MAX}                                                                   | _SC_ARG_MAX            |
| {ATEXIT_MAX}                                                                | _SC_ATEXIT_MAX         |
| {BC_BASE_MAX}                                                               | _SC_BC_BASE_MAX        |
| {BC_DIM_MAX}                                                                | _SC_BC_DIM_MAX         |
| {BC_SCALE_MAX}                                                              | _SC_BC_SCALE_MAX       |
| {BC_STRING_MAX}                                                             | _SC_BC_STRING_MAX      |
| {CHILD_MAX}                                                                 | _SC_CHILD_MAX          |
| Clock ticks/second                                                          | _SC_CLK_TCK            |
| {COLL_WEIGHTS_MAX}                                                          | _SC_COLL_WEIGHTS_MAX   |
| {DELAYTIMER_MAX}                                                            | _SC_DELAYTIMER_MAX     |
| {EXPR_NEST_MAX}                                                             | _SC_EXPR_NEST_MAX      |
| {HOST_NAME_MAX}                                                             | _SC_HOST_NAME_MAX      |
| {IOV_MAX}                                                                   | _SC_IOV_MAX            |
| {LINE_MAX}                                                                  | _SC_LINE_MAX           |
| {LOGIN_NAME_MAX}                                                            | _SC_LOGIN_NAME_MAX     |
| {NGROUPS_MAX}                                                               | _SC_NGROUPS_MAX        |
| Maximum size of <i>getgrgid_r()</i> and<br><i>getgrnam_r()</i> data buffers | _SC_GETGR_R_SIZE_MAX   |
| Maximum size of <i>getpwuid_r()</i> and<br><i>getpwnam_r()</i> data buffers | _SC_GETPW_R_SIZE_MAX   |
| {MQ_OPEN_MAX}                                                               | _SC_MQ_OPEN_MAX        |
| {MQ_PRIO_MAX}                                                               | _SC_MQ_PRIO_MAX        |
| {OPEN_MAX}                                                                  | _SC_OPEN_MAX           |
| _POSIX_ADVISORY_INFO                                                        | _SC_ADVISORY_INFO      |
| _POSIX_BARRIERS                                                             | _SC_BARRIERS           |
| _POSIX_ASYNCHRONOUS_IO                                                      | _SC_ASYNCHRONOUS_IO    |
| _POSIX_CLOCK_SELECTION                                                      | _SC_CLOCK_SELECTION    |
| _POSIX_CPUTIME                                                              | _SC_CPUTIME            |

|       | Variable                          | Value of Name                  |
|-------|-----------------------------------|--------------------------------|
| 45591 |                                   |                                |
| 45592 |                                   |                                |
| 45593 | _POSIX_FILE_LOCKING               | _SC_FILE_LOCKING               |
| 45594 | _POSIX_FSYNC                      | _SC_FSYNC                      |
| 45595 | _POSIX_IPV6                       | _SC_IPV6                       |
| 45596 | _POSIX_JOB_CONTROL                | _SC_JOB_CONTROL                |
| 45597 | _POSIX_MAPPED_FILES               | _SC_MAPPED_FILES               |
| 45598 | _POSIX_MEMLOCK                    | _SC_MEMLOCK                    |
| 45599 | _POSIX_MEMLOCK_RANGE              | _SC_MEMLOCK_RANGE              |
| 45600 | _POSIX_MEMORY_PROTECTION          | _SC_MEMORY_PROTECTION          |
| 45601 | _POSIX_MESSAGE_PASSING            | _SC_MESSAGE_PASSING            |
| 45602 | _POSIX_MONOTONIC_CLOCK            | _SC_MONOTONIC_CLOCK            |
| 45603 | _POSIX_MULTI_PROCESS              | _SC_MULTI_PROCESS              |
| 45604 | _POSIX_PRIORITIZED_IO             | _SC_PRIORITIZED_IO             |
| 45605 | _POSIX_PRIORITY_SCHEDULING        | _SC_PRIORITY_SCHEDULING        |
| 45606 | _POSIX_RAW_SOCKETS                | _SC_RAW_SOCKETS                |
| 45607 | _POSIX_READER_WRITER_LOCKS        | _SC_READER_WRITER_LOCKS        |
| 45608 | _POSIX_REALTIME_SIGNALS           | _SC_REALTIME_SIGNALS           |
| 45609 | _POSIX_REGEX                      | _SC_REGEX                      |
| 45610 | _POSIX_SAVED_IDS                  | _SC_SAVED_IDS                  |
| 45611 | _POSIX_SEMAPHORES                 | _SC_SEMAPHORES                 |
| 45612 | _POSIX_SHARED_MEMORY_OBJECTS      | _SC_SHARED_MEMORY_OBJECTS      |
| 45613 | _POSIX_SHELL                      | _SC_SHELL                      |
| 45614 | _POSIX_SPAWN                      | _SC_SPAWN                      |
| 45615 | _POSIX_SPIN_LOCKS                 | _SC_SPIN_LOCKS                 |
| 45616 | _POSIX_SPORADIC_SERVER            | _SC_SPORADIC_SERVER            |
| 45617 | _POSIX_SYMLINK_MAX                | _SC_SYMLINK_MAX                |
| 45618 | _POSIX_SYNCHRONIZED_IO            | _SC_SYNCHRONIZED_IO            |
| 45619 | _POSIX_THREAD_ATTR_STACKADDR      | _SC_THREAD_ATTR_STACKADDR      |
| 45620 | _POSIX_THREAD_ATTR_STACKSIZE      | _SC_THREAD_ATTR_STACKSIZE      |
| 45621 | _POSIX_THREAD_CPUTIME             | _SC_THREAD_CPUTIME             |
| 45622 | _POSIX_THREAD_PRIO_INHERIT        | _SC_THREAD_PRIO_INHERIT        |
| 45623 | _POSIX_THREAD_PRIO_PROTECT        | _SC_THREAD_PRIO_PROTECT        |
| 45624 | _POSIX_THREAD_PRIORITY_SCHEDULING | _SC_THREAD_PRIORITY_SCHEDULING |
| 45625 | _POSIX_THREAD_PROCESS_SHARED      | _SC_THREAD_PROCESS_SHARED      |
| 45626 | _POSIX_THREAD_SAFE_FUNCTIONS      | _SC_THREAD_SAFE_FUNCTIONS      |
| 45627 | _POSIX_THREAD_SPAWN               | _SC_THREAD_SPAWN               |
| 45628 | _POSIX_THREADS                    | _SC_THREADS                    |
| 45629 | _POSIX_TIMEOUTS                   | _SC_TIMEOUTS                   |
| 45630 | _POSIX_TIMERS                     | _SC_TIMERS                     |
| 45631 | _POSIX_TRACE                      | _SC_TRACE                      |
| 45632 | _POSIX_TRACE_EVENT_FILTER         | _SC_TRACE_EVENT_FILTER         |
| 45633 | _POSIX_TRACE_INHERIT              | _SC_TRACE_INHERIT              |
| 45634 | _POSIX_TRACE_LOG                  | _SC_TRACE_LOG                  |
| 45635 | _POSIX_TYPED_MEMORY_OBJECTS       | _SC_TYPED_MEMORY_OBJECTS       |
| 45636 | _POSIX_VERSION                    | _SC_VERSION                    |
| 45637 | _POSIX_V6_ILP32_OFF32             | _SC_V6_ILP32_OFF32             |
| 45638 | _POSIX_V6_ILP32_OFFBIG            | _SC_V6_ILP32_OFFBIG            |
| 45639 | _POSIX_V6_LP64_OFF64              | _SC_V6_LP64_OFF64              |

|       | Variable                        | Value of Name                    |
|-------|---------------------------------|----------------------------------|
| 45640 |                                 |                                  |
| 45641 |                                 |                                  |
| 45642 | _POSIX_V6_LPBIG_OFFBIG          | _SC_V6_LPBIG_OFFBIG              |
| 45643 | _POSIX2_C_BIND                  | _SC_2_C_BIND                     |
| 45644 | _POSIX2_C_DEV                   | _SC_2_C_DEV                      |
| 45645 | _POSIX2_C_VERSION               | _SC_2_C_VERSION                  |
| 45646 | _POSIX2_CHAR_TERM               | _SC_2_CHAR_TERM                  |
| 45647 | _POSIX2_FORT_DEV                | _SC_2_FORT_DEV                   |
| 45648 | _POSIX2_FORT_RUN                | _SC_2_FORT_RUN                   |
| 45649 | _POSIX2_LOCALEDEF               | _SC_2_LOCALEDEF                  |
| 45650 | _POSIX2_PBS                     | _SC_2_PBS                        |
| 45651 | _POSIX2_PBS_ACCOUNTING          | _SC_2_PBS_ACCOUNTING             |
| 45652 | _POSIX2_PBS_CHECKPOINT          | _SC_2_PBS_CHECKPOINT             |
| 45653 | _POSIX2_PBS_LOCATE              | _SC_2_PBS_LOCATE                 |
| 45654 | _POSIX2_PBS_MESSAGE             | _SC_2_PBS_MESSAGE                |
| 45655 | _POSIX2_PBS_TRACK               | _SC_2_PBS_TRACK                  |
| 45656 | _POSIX2_SW_DEV                  | _SC_2_SW_DEV                     |
| 45657 | _POSIX2_UPE                     | _SC_2_UPE                        |
| 45658 | _POSIX2_VERSION                 | _SC_2_VERSION                    |
| 45659 | _REGEX_VERSION                  | _SC_REGEX_VERSION                |
| 45660 | {PAGE_SIZE}                     | _SC_PAGE_SIZE                    |
| 45661 | {PAGESIZE}                      | _SC_PAGESIZE                     |
| 45662 | {PTHREAD_DESTRUCTOR_ITERATIONS} | _SC_THREAD_DESTRUCTOR_ITERATIONS |
| 45663 | {PTHREAD_KEYS_MAX}              | _SC_THREAD_KEYS_MAX              |
| 45664 | {PTHREAD_STACK_MIN}             | _SC_THREAD_STACK_MIN             |
| 45665 | {PTHREAD_THREADS_MAX}           | _SC_THREAD_THREADS_MAX           |
| 45666 | {RE_DUP_MAX}                    | _SC_RE_DUP_MAX                   |
| 45667 | {RTSIG_MAX}                     | _SC_RTSIG_MAX                    |
| 45668 | {SEM_NSEMS_MAX}                 | _SC_SEM_NSEMS_MAX                |
| 45669 | {SEM_VALUE_MAX}                 | _SC_SEM_VALUE_MAX                |
| 45670 | {SIGQUEUE_MAX}                  | _SC_SIGQUEUE_MAX                 |
| 45671 | {STREAM_MAX}                    | _SC_STREAM_MAX                   |
| 45672 | {SYMLOOP_MAX}                   | _SC_SYMLOOP_MAX                  |
| 45673 | {TIMER_MAX}                     | _SC_TIMER_MAX                    |
| 45674 | {TTY_NAME_MAX}                  | _SC_TTY_NAME_MAX                 |
| 45675 | {TZNAME_MAX}                    | _SC_TZNAME_MAX                   |
| 45676 | _XBS5_ILP32_OFF32 (LEGACY)      | _SC_XBS5_ILP32_OFF32 (LEGACY)    |
| 45677 | _XBS5_ILP32_OFFBIG (LEGACY)     | _SC_XBS5_ILP32_OFFBIG (LEGACY)   |
| 45678 | _XBS5_LP64_OFF64 (LEGACY)       | _SC_XBS5_LP64_OFF64 (LEGACY)     |
| 45679 | _XBS5_LPBIG_OFFBIG (LEGACY)     | _SC_XBS5_LPBIG_OFFBIG (LEGACY)   |
| 45680 | _XOPEN_CRYPT                    | _SC_XOPEN_CRYPT                  |
| 45681 | _XOPEN_ENH_I18N                 | _SC_XOPEN_ENH_I18N               |
| 45682 | _XOPEN_LEGACY                   | _SC_XOPEN_LEGACY                 |
| 45683 | _XOPEN_REALTIME                 | _SC_XOPEN_REALTIME               |
| 45684 | _XOPEN_REALTIME_THREADS         | _SC_XOPEN_REALTIME_THREADS       |
| 45685 | _XOPEN_SHM                      | _SC_XOPEN_SHM                    |
| 45686 | _XOPEN_STREAMS                  | _SC_XOPEN_STREAMS                |
| 45687 | _XOPEN_UNIX                     | _SC_XOPEN_UNIX                   |
| 45688 | _XOPEN_VERSION                  | _SC_XOPEN_VERSION                |

45689

45690

45691

| Variable           | Value of Name         |
|--------------------|-----------------------|
| _XOPEN_XCU_VERSION | _SC_XOPEN_XCU_VERSION |

45692 **RETURN VALUE**

45693

45694

45695

If *name* is an invalid value, *sysconf()* shall return  $-1$  and set *errno* to indicate the error. If the variable corresponding to *name* has no limit, *sysconf()* shall return  $-1$  without changing the value of *errno*. Note that indefinite limits do not imply infinite limits; see `<limits.h>`.

45696

45697

45698

45699 XSI

45700

45701

Otherwise, *sysconf()* shall return the current variable value on the system. The value returned shall not be more restrictive than the corresponding value described to the application when it was compiled with the implementation's `<limits.h>` or `<unistd.h>`. The value shall not change during the lifetime of the calling process, except that *sysconf*(\_SC\_OPEN\_MAX) may return different values before and after a call to *setrlimit()* which changes the RLIMIT\_NOFILE soft limit.

45702 **ERRORS**

45703

45704

The *sysconf()* function shall fail if:

[EINVAL] The value of the *name* argument is invalid.

45705 **EXAMPLES**

45706

None.

45707 **APPLICATION USAGE**

45708

45709

45710

As  $-1$  is a permissible return value in a successful situation, an application wishing to check for error situations should set *errno* to 0, then call *sysconf()*, and, if it returns  $-1$ , check to see if *errno* is non-zero.

45711

45712

45713

45714

45715

45716

45717

45718

If the value of *sysconf*(\_SC\_2\_VERSION) is not equal to the value of the \_POSIX2\_VERSION symbolic constant, the utilities available via *system()* or *popen()* might not behave as described in the Shell and Utilities volume of IEEE Std 1003.1-2001. This would mean that the application is not running in an environment that conforms to the Shell and Utilities volume of IEEE Std 1003.1-2001. Some applications might be able to deal with this, others might not. However, the functions defined in this volume of IEEE Std 1003.1-2001 continue to operate as specified, even if *sysconf*(\_SC\_2\_VERSION) reports that the utilities no longer perform as specified.

45719 **RATIONALE**

45720

45721

45722

This functionality was added in response to requirements of application developers and of system vendors who deal with many international system configurations. It is closely related to *pathconf()* and *fpathconf()*.

45723

45724

45725

45726

45727

Although a conforming application can run on all systems by never demanding more resources than the minimum values published in this volume of IEEE Std 1003.1-2001, it is useful for that application to be able to use the actual value for the quantity of a resource available on any given system. To do this, the application makes use of the value of a symbolic constant in `<limits.h>` or `<unistd.h>`.

45728

45729

45730

However, once compiled, the application must still be able to cope if the amount of resource available is increased. To that end, an application may need a means of determining the quantity of a resource, or the presence of an option, at execution time.

45731

Two examples are offered:

45732

45733

45734

1. Applications may wish to act differently on systems with or without job control. Applications vendors who wish to distribute only a single binary package to all instances of a computer architecture would be forced to assume job control is never available if it

45735 were to rely solely on the `<unistd.h>` value published in this volume of  
45736 IEEE Std 1003.1-2001.

45737 2. International applications vendors occasionally require knowledge of the number of clock  
45738 ticks per second. Without these facilities, they would be required to either distribute their  
45739 applications partially in source form or to have 50 Hz and 60 Hz versions for the various  
45740 countries in which they operate.

45741 It is the knowledge that many applications are actually distributed widely in executable form  
45742 that leads to this facility. If limited to the most restrictive values in the headers, such  
45743 applications would have to be prepared to accept the most limited environments offered by the  
45744 smallest microcomputers. Although this is entirely portable, there was a consensus that they  
45745 should be able to take advantage of the facilities offered by large systems, without the  
45746 restrictions associated with source and object distributions.

45747 During the discussions of this feature, it was pointed out that it is almost always possible for an  
45748 application to discern what a value might be at runtime by suitably testing the various functions  
45749 themselves. And, in any event, it could always be written to adequately deal with error returns  
45750 from the various functions. In the end, it was felt that this imposed an unreasonable level of  
45751 complication and sophistication on the application writer.

45752 This runtime facility is not meant to provide ever-changing values that applications have to  
45753 check multiple times. The values are seen as changing no more frequently than once per system  
45754 initialization, such as by a system administrator or operator with an automatic configuration  
45755 program. This volume of IEEE Std 1003.1-2001 specifies that they shall not change within the  
45756 lifetime of the process.

45757 Some values apply to the system overall and others vary at the file system or directory level. The  
45758 latter are described in *pathconf()*.

45759 Note that all values returned must be expressible as integers. String values were considered, but  
45760 the additional flexibility of this approach was rejected due to its added complexity of  
45761 implementation and use.

45762 Some values, such as `{PATH_MAX}`, are sometimes so large that they must not be used to, say,  
45763 allocate arrays. The *sysconf()* function returns a negative value to show that this symbolic  
45764 constant is not even defined in this case.

45765 Similar to *pathconf()*, this permits the implementation not to have a limit. When one resource is  
45766 infinite, returning an error indicating that some other resource limit has been reached is  
45767 conforming behavior.

#### 45768 **FUTURE DIRECTIONS**

45769 None.

#### 45770 **SEE ALSO**

45771 *confstr()*, *pathconf()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<limits.h>`,  
45772 `<unistd.h>`, the Shell and Utilities volume of IEEE Std 1003.1-2001, *getconf*

#### 45773 **CHANGE HISTORY**

45774 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

#### 45775 **Issue 5**

45776 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
45777 Threads Extension.

45778 The `_XBS_` variables and name values are added to the table of system variables in the  
45779 DESCRIPTION. These are all marked EX.

45780 **Issue 6**

45781 The symbol CLK\_TCK is obsolescent and removed. It is replaced with the phrase “clock ticks  
45782 per second”.

45783 The symbol {PASS\_MAX} is removed.

45784 The following changes were made to align with the IEEE P1003.1a draft standard:

- 45785 • Table entries are added for the following variables: \_SC\_REGEX, \_SC\_SHELL,  
45786 \_SC\_REGEX\_VERSION, \_SC\_SYMLOOP\_MAX.

45787 The following *sysconf()* variables and their associated names are added for alignment with  
45788 IEEE Std 1003.1d-1999:

```
45789 _POSIX_ADVISORY_INFO
45790 _POSIX_CPUTIME
45791 _POSIX_SPAWN
45792 _POSIX_SPORADIC_SERVER
45793 _POSIX_THREAD_CPUTIME
45794 _POSIX_THREAD_SPORADIC_SERVER
45795 _POSIX_TIMEOUTS
```

45796 The following changes are made to the DESCRIPTION for alignment with IEEE Std 1003.1j-2000:

- 45797 • A statement expressing the dependency of support for some system variables on  
45798 implementation options is added.
- 45799 • The following system variables are added:

```
45800 _POSIX_BARRIERS
45801 _POSIX_CLOCK_SELECTION
45802 _POSIX_MONOTONIC_CLOCK
45803 _POSIX_READER_WRITER_LOCKS
45804 _POSIX_SPIN_LOCKS
45805 _POSIX_TYPED_MEMORY_OBJECTS
```

45806 The following system variables are added for alignment with IEEE Std 1003.2d-1994:

```
45807 _POSIX2_PBS
45808 _POSIX2_PBS_ACCOUNTING
45809 _POSIX2_PBS_LOCATE
45810 _POSIX2_PBS_MESSAGE
45811 _POSIX2_PBS_TRACK
```

45812 The following *sysconf()* variables and their associated names are added for alignment with  
45813 IEEE Std 1003.1q-2000:

```
45814 _POSIX_TRACE
45815 _POSIX_TRACE_EVENT_FILTER
45816 _POSIX_TRACE_INHERIT
45817 _POSIX_TRACE_LOG
```

45818 The macros associated with the *c89* programming models are marked LEGACY, and new  
45819 equivalent macros associated with *c99* are introduced.

45820 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/62 is applied, updating the  
45821 DESCRIPTION to denote that the \_PC\* and \_SC\* symbols are now required to be supported. A  
45822 corresponding change has been made in the Base Definitions volume of IEEE Std 1003.1-2001.  
45823 The deletion in the second paragraph removes some duplicated text. Additional symbols that  
45824 were erroneously omitted from this reference page have been added.

45825 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/63 is applied, making it clear in the  
45826 RETURN VALUE section that the value returned for `sysconf(_SC_OPEN_MAX)` may change if a  
45827 call to `setrlimit()` adjusts the `RLIMIT_NOFILE` soft limit.

45828 **NAME**

45829            syslog — log a message

45830 **SYNOPSIS**

45831 XSI        #include <syslog.h>

45832            void syslog(int *priority*, const char \**message*, ... /\* *argument* \*/);

45833

45834 **DESCRIPTION**

45835            Refer to *closelog()*.

45836 **NAME**

45837 system — issue a command

45838 **SYNOPSIS**

45839 #include &lt;stdlib.h&gt;

45840 int system(const char \**command*);45841 **DESCRIPTION**

45842 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 45843 conflict between the requirements described here and the ISO C standard is unintentional. This  
 45844 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

45845 If *command* is a null pointer, the *system()* function shall determine whether the host environment  
 45846 has a command processor. If *command* is not a null pointer, the *system()* function shall pass the  
 45847 string pointed to by *command* to that command processor to be executed in an implementation-  
 45848 defined manner; this might then cause the program calling *system()* to behave in a non-  
 45849 conforming manner or to terminate.

45850 CX The environment of the executed command shall be as if a child process were created using  
 45851 *fork()*, and the child process invoked the *sh* utility using *execl()* as follows:

```
45852 execl(<shell path>, "sh", "-c", command, (char *)0);
```

45853 where <shell path> is an unspecified pathname for the *sh* utility.

45854 The *system()* function shall ignore the SIGINT and SIGQUIT signals, and shall block the  
 45855 SIGCHLD signal, while waiting for the command to terminate. If this might cause the  
 45856 application to miss a signal that would have killed it, then the application should examine the  
 45857 return value from *system()* and take whatever action is appropriate to the application if the  
 45858 command terminated due to receipt of a signal.

45859 The *system()* function shall not affect the termination status of any child of the calling processes  
 45860 other than the process or processes it itself creates.

45861 The *system()* function shall not return until the child process has terminated.

45862 **RETURN VALUE**

45863 If *command* is a null pointer, *system()* shall return non-zero to indicate that a command processor  
 45864 CX is available, or zero if none is available. The *system()* function shall always return non-zero when  
 45865 *command* is NULL.

45866 CX If *command* is not a null pointer, *system()* shall return the termination status of the command  
 45867 language interpreter in the format specified by *waitpid()*. The termination status shall be as  
 45868 defined for the *sh* utility; otherwise, the termination status is unspecified. If some error prevents  
 45869 the command language interpreter from executing after the child process is created, the return  
 45870 value from *system()* shall be as if the command language interpreter had terminated using  
 45871 *exit(127)* or *\_exit(127)*. If a child process cannot be created, or if the termination status for the  
 45872 command language interpreter cannot be obtained, *system()* shall return -1 and set *errno* to  
 45873 indicate the error.

45874 **ERRORS**

45875 CX The *system()* function may set *errno* values as described by *fork()*.

45876 In addition, *system()* may fail if:

45877 CX [ECHILD] The status of the child process created by *system()* is no longer available.

45878 **EXAMPLES**

45879 None.

45880 **APPLICATION USAGE**

45881 If the return value of *system()* is not `-1`, its value can be decoded through the use of the macros  
45882 described in `<sys/wait.h>`. For convenience, these macros are also provided in `<stdlib.h>`.

45883 Note that, while *system()* must ignore `SIGINT` and `SIGQUIT` and block `SIGCHLD` while waiting  
45884 for the child to terminate, the handling of signals in the executed command is as specified by  
45885 *fork()* and *exec*. For example, if `SIGINT` is being caught or is set to `SIG_DFL` when *system()* is  
45886 called, then the child is started with `SIGINT` handling set to `SIG_DFL`.

45887 Ignoring `SIGINT` and `SIGQUIT` in the parent process prevents coordination problems (two  
45888 processes reading from the same terminal, for example) when the executed command ignores or  
45889 catches one of the signals. It is also usually the correct action when the user has given a  
45890 command to the application to be executed synchronously (as in the `'!'` command in many  
45891 interactive applications). In either case, the signal should be delivered only to the child process,  
45892 not to the application itself. There is one situation where ignoring the signals might have less  
45893 than the desired effect. This is when the application uses *system()* to perform some task invisible  
45894 to the user. If the user typed the interrupt character ("`^C`", for example) while *system()* is being  
45895 used in this way, one would expect the application to be killed, but only the executed command  
45896 is killed. Applications that use *system()* in this way should carefully check the return status from  
45897 *system()* to see if the executed command was successful, and should take appropriate action  
45898 when the command fails.

45899 Blocking `SIGCHLD` while waiting for the child to terminate prevents the application from  
45900 catching the signal and obtaining status from *system()*'s child process before *system()* can get the  
45901 status itself.

45902 The context in which the utility is ultimately executed may differ from that in which *system()*  
45903 was called. For example, file descriptors that have the `FD_CLOEXEC` flag set are closed, and the  
45904 process ID and parent process ID are different. Also, if the executed utility changes its  
45905 environment variables or its current working directory, that change is not reflected in the caller's  
45906 context.

45907 There is no defined way for an application to find the specific path for the shell. However,  
45908 *confstr()* can provide a value for *PATH* that is guaranteed to find the *sh* utility.

45909 **RATIONALE**

45910 The *system()* function should not be used by programs that have set user (or group) ID  
45911 privileges. The *fork()* and *exec* family of functions (except *execlp()* and *execvp()*), should be used  
45912 instead. This prevents any unforeseen manipulation of the environment of the user that could  
45913 cause execution of commands not anticipated by the calling program.

45914 There are three levels of specification for the *system()* function. The ISO C standard gives the  
45915 most basic. It requires that the function exists, and defines a way for an application to query  
45916 whether a command language interpreter exists. It says nothing about the command language or  
45917 the environment in which the command is interpreted.

45918 IEEE Std 1003.1-2001 places additional restrictions on *system()*. It requires that if there is a  
45919 command language interpreter, the environment must be as specified by *fork()* and *exec*. This  
45920 ensures, for example, that close-on-exec works, that file locks are not inherited, and that the  
45921 process ID is different. It also specifies the return value from *system()* when the command line  
45922 can be run, thus giving the application some information about the command's completion  
45923 status.

45924 Finally, IEEE Std 1003.1-2001 requires the command to be interpreted as in the shell command  
45925 language defined in the Shell and Utilities volume of IEEE Std 1003.1-2001.

45926 Note that, *system*(NULL) is required to return non-zero, indicating that there is a command  
45927 language interpreter. At first glance, this would seem to conflict with the ISO C standard which  
45928 allows *system*(NULL) to return zero. There is no conflict, however. A system must have a  
45929 command language interpreter, and is non-conforming if none is present. It is therefore  
45930 permissible for the *system*() function on such a system to implement the behavior specified by  
45931 the ISO C standard as long as it is understood that the implementation does not conform to  
45932 IEEE Std 1003.1-2001 if *system*(NULL) returns zero.

45933 It was explicitly decided that when *command* is NULL, *system*() should not be required to check  
45934 to make sure that the command language interpreter actually exists with the correct mode, that  
45935 there are enough processes to execute it, and so on. The call *system*(NULL) could, theoretically,  
45936 check for such problems as too many existing child processes, and return zero. However, it  
45937 would be inappropriate to return zero due to such a (presumably) transient condition. If some  
45938 condition exists that is not under the control of this application and that would cause any  
45939 *system*() call to fail, that system has been rendered non-conforming.

45940 Early drafts required, or allowed, *system*() to return with *errno* set to [EINTR] if it was  
45941 interrupted with a signal. This error return was removed, and a requirement that *system*() not  
45942 return until the child has terminated was added. This means that if a *waitpid*() call in *system*()  
45943 exits with *errno* set to [EINTR], *system*() must reissue the *waitpid*(). This change was made for  
45944 two reasons:

- 45945 1. There is no way for an application to clean up if *system*() returns [EINTR], short of calling  
45946 *wait*(), and that could have the undesirable effect of returning the status of children other  
45947 than the one started by *system*()).
- 45948 2. While it might require a change in some historical implementations, those  
45949 implementations already have to be changed because they use *wait*() instead of *waitpid*()).

45950 Note that if the application is catching SIGCHLD signals, it will receive such a signal before a  
45951 successful *system*() call returns.

45952 To conform to IEEE Std 1003.1-2001, *system*() must use *waitpid*(), or some similar function,  
45953 instead of *wait*()).

45954 The following code sample illustrates how *system*() might be implemented on an  
45955 implementation conforming to IEEE Std 1003.1-2001.

```
45956 #include <signal.h>
45957 int system(const char *cmd)
45958 {
45959 int stat;
45960 pid_t pid;
45961 struct sigaction sa, savintr, savequit;
45962 sigset_t saveblock;
45963 if (cmd == NULL)
45964 return(1);
45965 sa.sa_handler = SIG_IGN;
45966 sigemptyset(&sa.sa_mask);
45967 sa.sa_flags = 0;
45968 sigemptyset(&savintr.sa_mask);
45969 sigemptyset(&savequit.sa_mask);
45970 sigaction(SIGINT, &sa, &savintr);
45971 sigaction(SIGQUIT, &sa, &savequit);
```

```

45972 sigaddset(&sa.sa_mask, SIGCHLD);
45973 sigprocmask(SIG_BLOCK, &sa.sa_mask, &saveblock);
45974 if ((pid = fork()) == 0) {
45975 sigaction(SIGINT, &saveintr, (struct sigaction *)0);
45976 sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
45977 sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
45978 execl("/bin/sh", "sh", "-c", cmd, (char *)0);
45979 _exit(127);
45980 }
45981 if (pid == -1) {
45982 stat = -1; /* errno comes from fork() */
45983 } else {
45984 while (waitpid(pid, &stat, 0) == -1) {
45985 if (errno != EINTR) {
45986 stat = -1;
45987 break;
45988 }
45989 }
45990 }
45991 sigaction(SIGINT, &saveintr, (struct sigaction *)0);
45992 sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
45993 sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
45994 return(stat);
45995 }

```

45996 Note that, while a particular implementation of *system()* (such as the one above) can assume a  
45997 particular path for the shell, such a path is not necessarily valid on another system. The above  
45998 example is not portable, and is not intended to be.

45999 One reviewer suggested that an implementation of *system()* might want to use an environment  
46000 variable such as *SHELL* to determine which command interpreter to use. The supposed  
46001 implementation would use the default command interpreter if the one specified by the  
46002 environment variable was not available. This would allow a user, when using an application  
46003 that prompts for command lines to be processed using *system()*, to specify a different command  
46004 interpreter. Such an implementation is discouraged. If the alternate command interpreter did not  
46005 follow the command line syntax specified in the Shell and Utilities volume of  
46006 IEEE Std 1003.1-2001, then changing *SHELL* would render *system()* non-conforming. This would  
46007 affect applications that expected the specified behavior from *system()*, and since the Shell and  
46008 Utilities volume of IEEE Std 1003.1-2001 does not mention that *SHELL* affects *system()*, the  
46009 application would not know that it needed to unset *SHELL*.

#### 46010 FUTURE DIRECTIONS

46011 None.

#### 46012 SEE ALSO

46013 *exec*, *pipe()*, *waitpid()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<limits.h>**,  
46014 **<signal.h>**, **<stdlib.h>**, **<sys/wait.h>**, the Shell and Utilities volume of IEEE Std 1003.1-2001, *sh*

#### 46015 CHANGE HISTORY

46016 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 46017 Issue 6

46018 Extensions beyond the ISO C standard are marked.

46019 **NAME**

46020 tan, tanf, tanl — tangent function

46021 **SYNOPSIS**

46022 #include &lt;math.h&gt;

46023 double tan(double x);

46024 float tanf(float x);

46025 long double tanl(long double x);

46026 **DESCRIPTION**

46027 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 46028 conflict between the requirements described here and the ISO C standard is unintentional. This  
 46029 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

46030 These functions shall compute the tangent of their argument *x*, measured in radians.

46031 An application wishing to check for error situations should set *errno* to zero and call  
 46032 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 46033 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 46034 zero, an error has occurred.

46035 **RETURN VALUE**46036 Upon successful completion, these functions shall return the tangent of *x*.

46037 If the correct value would cause underflow, and is not representable, a range error may occur,  
 46038 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

46039 MX If *x* is NaN, a NaN shall be returned.46040 If *x* is  $\pm 0$ , *x* shall be returned.46041 If *x* is subnormal, a range error may occur and *x* should be returned.

46042 If *x* is  $\pm\text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an implementation-  
 46043 defined value shall be returned.

46044 If the correct value would cause underflow, and is representable, a range error may occur and  
 46045 the correct value shall be returned.

46046 XSI If the correct value would cause overflow, a range error shall occur and *tan()*, *tanf()*, and *tanl()*  
 46047 shall return  $\pm\text{HUGE\_VAL}$ ,  $\pm\text{HUGE\_VALF}$ , and  $\pm\text{HUGE\_VALL}$ , respectively, with the same sign  
 46048 as the correct value of the function.

46049 **ERRORS**

46050 These functions shall fail if:

46051 MX **Domain Error** The value of *x* is  $\pm\text{Inf}$ .

46052 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 46053 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling  
 46054 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 46055 shall be raised.

46056 XSI **Range Error** The result overflows

46057 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 46058 then *errno* shall be set to [ERANGE]. If the integer expression  
 46059 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow  
 46060 floating-point exception shall be raised.

46061 These functions may fail if:

46062 MX Range Error The result underflows, or the value of  $x$  is subnormal.

46063 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 46064 then *errno* shall be set to [ERANGE]. If the integer expression  
 46065 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow  
 46066 floating-point exception shall be raised.

#### 46067 EXAMPLES

##### 46068 Taking the Tangent of a 45-Degree Angle

```
46069 #include <math.h>
46070 ...
46071 double radians = 45.0 * M_PI / 180;
46072 double result;
46073 ...
46074 result = tan (radians);
```

#### 46075 APPLICATION USAGE

46076 There are no known floating-point representations such that for a normal argument,  $\tan(x)$  is  
 46077 either overflow or underflow.

46078 These functions may lose accuracy when their argument is near a multiple of  $\pi/2$  or is far from  
 46079 0.0.

46080 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 46081 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

#### 46082 RATIONALE

46083 None.

#### 46084 FUTURE DIRECTIONS

46085 None.

#### 46086 SEE ALSO

46087 *atan()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
 46088 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

#### 46089 CHANGE HISTORY

46090 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 46091 Issue 5

46092 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes  
 46093 in previous issues.

#### 46094 Issue 6

46095 The *tanf()* and *tanl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

46096 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
 46097 revised to align with the ISO/IEC 9899:1999 standard.

46098 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
 46099 marked.

46100 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/64 is applied, correcting the last  
 46101 paragraph in the RETURN VALUE section. |

46102 **NAME**

46103 tanh, tanhf, tanhl — hyperbolic tangent functions

46104 **SYNOPSIS**

46105 #include &lt;math.h&gt;

46106 double tanh(double x);

46107 float tanhf(float x);

46108 long double tanhl(long double x);

46109 **DESCRIPTION**

46110 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 46111 conflict between the requirements described here and the ISO C standard is unintentional. This  
 46112 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

46113 These functions shall compute the hyperbolic tangent of their argument *x*.

46114 An application wishing to check for error situations should set *errno* to zero and call  
 46115 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 46116 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 46117 zero, an error has occurred.

46118 **RETURN VALUE**46119 Upon successful completion, these functions shall return the hyperbolic tangent of *x*.46120 **MX** If *x* is NaN, a NaN shall be returned.46121 If *x* is  $\pm 0$ , *x* shall be returned.46122 If *x* is  $\pm\text{Inf}$ ,  $\pm 1$  shall be returned.46123 If *x* is subnormal, a range error may occur and *x* should be returned.46124 **ERRORS**

46125 These functions may fail if:

46126 **MX** **Range Error** The value of *x* is subnormal.

46127 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 46128 then *errno* shall be set to [ERANGE]. If the integer expression  
 46129 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow  
 46130 floating-point exception shall be raised.

46131 **EXAMPLES**

46132 None.

46133 **APPLICATION USAGE**

46134 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 46135 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

46136 **RATIONALE**

46137 None.

46138 **FUTURE DIRECTIONS**

46139 None.

46140 **SEE ALSO**

46141 *atanh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tan()*, the Base Definitions volume of  
 46142 IEEE Std 1003.1-2001, Section 4.18, Treatment of Error Conditions for Mathematical Functions,  
 46143 <math.h>

46144 **CHANGE HISTORY**

46145 First released in Issue 1. Derived from Issue 1 of the SVID.

46146 **Issue 5**

46147 The DESCRIPTION is updated to indicate how an application should check for an error. This  
46148 text was previously published in the APPLICATION USAGE section.

46149 **Issue 6**

46150 The *tanhf()* and *tanhl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

46151 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
46152 revised to align with the ISO/IEC 9899:1999 standard.

46153 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
46154 marked.

46155 **NAME**

46156           tanl — tangent function

46157 **SYNOPSIS**

46158           #include &lt;math.h&gt;

46159           long double tanl(long double x);

46160 **DESCRIPTION**46161           Refer to *tan()*.

46162 **NAME**

46163 tcdrain — wait for transmission of output

46164 **SYNOPSIS**

46165 #include &lt;termios.h&gt;

46166 int tcdrain(int *fildev*);46167 **DESCRIPTION**46168 The *tcdrain()* function shall block until all output written to the object referred to by *fildev* is  
46169 transmitted. The *fildev* argument is an open file descriptor associated with a terminal.46170 Any attempts to use *tcdrain()* from a process which is a member of a background process group  
46171 on a *fildev* associated with its controlling terminal, shall cause the process group to be sent a  
46172 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process  
46173 shall be allowed to perform the operation, and no signal is sent.46174 **RETURN VALUE**46175 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
46176 indicate the error.46177 **ERRORS**46178 The *tcdrain()* function shall fail if:46179 [EBADF] The *fildev* argument is not a valid file descriptor.46180 [EINTR] A signal interrupted *tcdrain()*.46181 [ENOTTY] The file associated with *fildev* is not a terminal.46182 The *tcdrain()* function may fail if:46183 [EIO] The process group of the writing process is orphaned, and the writing process  
46184 is not ignoring or blocking SIGTTOU.46185 **EXAMPLES**

46186 None.

46187 **APPLICATION USAGE**

46188 None.

46189 **RATIONALE**

46190 None.

46191 **FUTURE DIRECTIONS**

46192 None.

46193 **SEE ALSO**46194 *tcfldsh()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal  
46195 Interface, <termios.h>, <unistd.h>46196 **CHANGE HISTORY**

46197 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

46198 **Issue 6**

46199 The following new requirements on POSIX implementations derive from alignment with the  
46200 Single UNIX Specification:

- 46201 • In the DESCRIPTION, the final paragraph is no longer conditional on
- 46202 `_POSIX_JOB_CONTROL`. This is a FIPS requirement.
- 46203 • The [EIO] error is added.

46204 **NAME**

46205 tcfLOW — suspend or restart the transmission or reception of data

46206 **SYNOPSIS**

46207 #include &lt;termios.h&gt;

46208 int tcfLOW(int *filDES*, int *actiON*);46209 **DESCRIPTION**46210 The *tcfLOW()* function shall suspend or restart transmission or reception of data on the object  
46211 referred to by *filDES*, depending on the value of *actiON*. The *filDES* argument is an open file  
46212 descriptor associated with a terminal.

- 46213 • If *actiON* is TCOFF, output shall be suspended.
- 46214 • If *actiON* is TCOON, suspended output shall be restarted.
- 46215 • If *actiON* is TCIOFF, the system shall transmit a STOP character, which is intended to cause  
46216 the terminal device to stop transmitting data to the system.
- 46217 • If *actiON* is TCION, the system shall transmit a START character, which is intended to cause  
46218 the terminal device to start transmitting data to the system.

46219 The default on the opening of a terminal file is that neither its input nor its output are  
46220 suspended.46221 Attempts to use *tcfLOW()* from a process which is a member of a background process group on a  
46222 *filDES* associated with its controlling terminal, shall cause the process group to be sent a  
46223 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process  
46224 shall be allowed to perform the operation, and no signal is sent.46225 **RETURN VALUE**46226 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
46227 indicate the error.46228 **ERRORS**46229 The *tcfLOW()* function shall fail if:

- 46230 [EBADF] The *filDES* argument is not a valid file descriptor.
- 46231 [EINVAL] The *actiON* argument is not a supported value.
- 46232 [ENOTTY] The file associated with *filDES* is not a terminal.

46233 The *tcfLOW()* function may fail if:

- 46234 [EIO] The process group of the writing process is orphaned, and the writing process  
46235 is not ignoring or blocking SIGTTOU.

46236 **EXAMPLES**

46237 None.

46238 **APPLICATION USAGE**

46239 None.

46240 **RATIONALE**

46241 None.

46242 **FUTURE DIRECTIONS**

46243 None.

46244 **SEE ALSO**

46245 *tcsendbreak()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface, <termios.h>, <unistd.h>

46247 **CHANGE HISTORY**

46248 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

46249 **Issue 6**

46250 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 46251
- 46252 • The [EIO] error is added.

46253 **NAME**

46254 tcflush — flush non-transmitted output data, non-read input data, or both

46255 **SYNOPSIS**

46256 #include &lt;termios.h&gt;

46257 int tcflush(int *fildev*, int *queue\_selector*);46258 **DESCRIPTION**46259 Upon successful completion, *tcflush()* shall discard data written to the object referred to by *fildev*  
46260 (an open file descriptor associated with a terminal) but not transmitted, or data received but not  
46261 read, depending on the value of *queue\_selector*:

- 46262 • If *queue\_selector* is TCIFLUSH, it shall flush data received but not read.
- 46263 • If *queue\_selector* is TCOFLUSH, it shall flush data written but not transmitted.
- 46264 • If *queue\_selector* is TCIOFLUSH, it shall flush both data received but not read and data  
46265 written but not transmitted.

46266 Attempts to use *tcflush()* from a process which is a member of a background process group on a  
46267 *fildev* associated with its controlling terminal shall cause the process group to be sent a SIGTTOU  
46268 signal. If the calling process is blocking or ignoring SIGTTOU signals, the process shall be  
46269 allowed to perform the operation, and no signal is sent.

46270 **RETURN VALUE**

46271 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
46272 indicate the error.

46273 **ERRORS**46274 The *tcflush()* function shall fail if:

- 46275 [EBADF] The *fildev* argument is not a valid file descriptor.
- 46276 [EINVAL] The *queue\_selector* argument is not a supported value.
- 46277 [ENOTTY] The file associated with *fildev* is not a terminal.

46278 The *tcflush()* function may fail if:

- 46279 [EIO] The process group of the writing process is orphaned, and the writing process  
46280 is not ignoring or blocking SIGTTOU.

46281 **EXAMPLES**

46282 None.

46283 **APPLICATION USAGE**

46284 None.

46285 **RATIONALE**

46286 None.

46287 **FUTURE DIRECTIONS**

46288 None.

46289 **SEE ALSO**

46290 *tcdrain()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal  
46291 Interface, <termios.h>, <unistd.h>

46292 **CHANGE HISTORY**

46293 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

46294 **Issue 6**

46295 The Open Group Corrigendum U035/1 is applied. In the ERRORS and APPLICATION USAGE  
46296 sections, references to *tcfow()* are replaced with *tcflush()*.

46297 The following new requirements on POSIX implementations derive from alignment with the  
46298 Single UNIX Specification:

- 46299 • In the DESCRIPTION, the final paragraph is no longer conditional on  
46300 `_POSIX_JOB_CONTROL`. This is a FIPS requirement.
- 46301 • The [EIO] error is added.

## 46302 NAME

46303 tcgetattr — get the parameters associated with the terminal

## 46304 SYNOPSIS

46305 #include <termios.h>

46306 int tcgetattr(int *fildev*, struct termios \**termios\_p*);

## 46307 DESCRIPTION

46308 The *tcgetattr()* function shall get the parameters associated with the terminal referred to by *fildev*  
46309 and store them in the **termios** structure referenced by *termios\_p*. The *fildev* argument is an open  
46310 file descriptor associated with a terminal.

46311 The *termios\_p* argument is a pointer to a **termios** structure.

46312 The *tcgetattr()* operation is allowed from any process.

46313 If the terminal device supports different input and output baud rates, the baud rates stored in  
46314 the **termios** structure returned by *tcgetattr()* shall reflect the actual baud rates, even if they are  
46315 equal. If differing baud rates are not supported, the rate returned as the output baud rate shall be  
46316 the actual baud rate. If the terminal device does not support split baud rates, the input baud rate  
46317 stored in the **termios** structure shall be the output rate (as one of the symbolic values).

## 46318 RETURN VALUE

46319 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
46320 indicate the error.

## 46321 ERRORS

46322 The *tcgetattr()* function shall fail if:

46323 [EBADF] The *fildev* argument is not a valid file descriptor.

46324 [ENOTTY] The file associated with *fildev* is not a terminal.

## 46325 EXAMPLES

46326 None.

## 46327 APPLICATION USAGE

46328 None.

## 46329 RATIONALE

46330 Care must be taken when changing the terminal attributes. Applications should always do a  
46331 *tcgetattr()*, save the **termios** structure values returned, and then do a *tcsetattr()*, changing only  
46332 the necessary fields. The application should use the values saved from the *tcgetattr()* to reset the  
46333 terminal state whenever it is done with the terminal. This is necessary because terminal  
46334 attributes apply to the underlying port and not to each individual open instance; that is, all  
46335 processes that have used the terminal see the latest attribute changes.

46336 A program that uses these functions should be written to catch all signals and take other  
46337 appropriate actions to ensure that when the program terminates, whether planned or not, the  
46338 terminal device's state is restored to its original state.

46339 Existing practice dealing with error returns when only part of a request can be honored is based  
46340 on calls to the *ioctl()* function. In historical BSD and System V implementations, the  
46341 corresponding *ioctl()* returns zero if the requested actions were semantically correct, even if  
46342 some of the requested changes could not be made. Many existing applications assume this  
46343 behavior and would no longer work correctly if the return value were changed from zero to -1  
46344 in this case.

46345 Note that either specification has a problem. When zero is returned, it implies everything  
46346 succeeded even if some of the changes were not made. When -1 is returned, it implies  
46347 everything failed even though some of the changes were made.

46348 Applications that need all of the requested changes made to work properly should follow  
46349 *tcsetattr()* with a call to *tcgetattr()* and compare the appropriate field values.

46350 **FUTURE DIRECTIONS**

46351 None.

46352 **SEE ALSO**

46353 *tcsetattr()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal  
46354 Interface, <**termios.h**>

46355 **CHANGE HISTORY**

46356 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

46357 **Issue 6**

46358 In the DESCRIPTION, the rate returned as the input baud rate shall be the output rate.  
46359 Previously, the number zero was also allowed but was obsolescent.

46360 **NAME**

46361 tcgetpgrp — get the foreground process group ID

46362 **SYNOPSIS**

46363 #include &lt;unistd.h&gt;

46364 pid\_t tcgetpgrp(int *fildev*);46365 **DESCRIPTION**46366 The *tcgetpgrp()* function shall return the value of the process group ID of the foreground process  
46367 group associated with the terminal.46368 If there is no foreground process group, *tcgetpgrp()* shall return a value greater than 1 that does  
46369 not match the process group ID of any existing process group.46370 The *tcgetpgrp()* function is allowed from a process that is a member of a background process  
46371 group; however, the information may be subsequently changed by a process that is a member of  
46372 a foreground process group.46373 **RETURN VALUE**46374 Upon successful completion, *tcgetpgrp()* shall return the value of the process group ID of the  
46375 foreground process associated with the terminal. Otherwise, -1 shall be returned and *errno* set to  
46376 indicate the error.46377 **ERRORS**46378 The *tcgetpgrp()* function shall fail if:46379 [EBADF] The *fildev* argument is not a valid file descriptor.46380 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the  
46381 controlling terminal.46382 **EXAMPLES**

46383 None.

46384 **APPLICATION USAGE**

46385 None.

46386 **RATIONALE**

46387 None.

46388 **FUTURE DIRECTIONS**

46389 None.

46390 **SEE ALSO**46391 *setsid()*, *setpgid()*, *tcsetpgrp()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
46392 <sys/types.h>, <unistd.h>46393 **CHANGE HISTORY**

46394 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

46395 **Issue 6**

46396 In the SYNOPSIS, the optional include of the &lt;sys/types.h&gt; header is removed.

46397 The following new requirements on POSIX implementations derive from alignment with the  
46398 Single UNIX Specification:

- 46399
- The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
46400 required for conforming implementations of previous POSIX specifications, it was not  
46401 required for UNIX applications.

46402  
46403

- In the DESCRIPTION, text previously conditional on support for `_POSIX_JOB_CONTROL` is now mandatory. This is a FIPS requirement.

46404 **NAME**

46405 tcgetsid — get the process group ID for the session leader for the controlling terminal

46406 **SYNOPSIS**

46407 XSI #include <termios.h>

46408 pid\_t tcgetsid(int *fildes*);

46409

46410 **DESCRIPTION**

46411 The *tcgetsid()* function shall obtain the process group ID of the session for which the terminal  
46412 specified by *fildes* is the controlling terminal.

46413 **RETURN VALUE**

46414 Upon successful completion, *tcgetsid()* shall return the process group ID associated with the  
46415 terminal. Otherwise, a value of (**pid\_t**)-1 shall be returned and *errno* set to indicate the error.

46416 **ERRORS**

46417 The *tcgetsid()* function shall fail if:

46418 [EBADF] The *fildes* argument is not a valid file descriptor.

46419 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the  
46420 controlling terminal.

46421 **EXAMPLES**

46422 None.

46423 **APPLICATION USAGE**

46424 None.

46425 **RATIONALE**

46426 None.

46427 **FUTURE DIRECTIONS**

46428 None.

46429 **SEE ALSO**

46430 The Base Definitions volume of IEEE Std 1003.1-2001, <**termios.h**>

46431 **CHANGE HISTORY**

46432 First released in Issue 4, Version 2.

46433 **Issue 5**

46434 Moved from X/OPEN UNIX extension to BASE.

46435 The [EACCES] error has been removed from the list of mandatory errors, and the description of  
46436 [ENOTTY] has been reworded.

46437 **NAME**

46438 tcsendbreak — send a break for a specific duration

46439 **SYNOPSIS**

46440 #include <termios.h>

46441 int tcsendbreak(int *fildev*, int *duration*);

46442 **DESCRIPTION**

46443 If the terminal is using asynchronous serial data transmission, *tcsendbreak()* shall cause  
46444 transmission of a continuous stream of zero-valued bits for a specific duration. If *duration* is 0, it  
46445 shall cause transmission of zero-valued bits for at least 0.25 seconds, and not more than 0.5  
46446 seconds. If *duration* is not 0, it shall send zero-valued bits for an implementation-defined period  
46447 of time.

46448 The *fildev* argument is an open file descriptor associated with a terminal.

46449 If the terminal is not using asynchronous serial data transmission, it is implementation-defined  
46450 whether *tcsendbreak()* sends data to generate a break condition or returns without taking any  
46451 action.

46452 Attempts to use *tcsendbreak()* from a process which is a member of a background process group  
46453 on a *fildev* associated with its controlling terminal shall cause the process group to be sent a  
46454 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process  
46455 shall be allowed to perform the operation, and no signal is sent.

46456 **RETURN VALUE**

46457 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
46458 indicate the error.

46459 **ERRORS**

46460 The *tcsendbreak()* function shall fail if:

46461 [EBADF] The *fildev* argument is not a valid file descriptor.

46462 [ENOTTY] The file associated with *fildev* is not a terminal.

46463 The *tcsendbreak()* function may fail if:

46464 [EIO] The process group of the writing process is orphaned, and the writing process  
46465 is not ignoring or blocking SIGTTOU.

46466 **EXAMPLES**

46467 None.

46468 **APPLICATION USAGE**

46469 None.

46470 **RATIONALE**

46471 None.

46472 **FUTURE DIRECTIONS**

46473 None.

46474 **SEE ALSO**

46475 The Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11, General Terminal Interface,  
46476 <termios.h>, <unistd.h>

46477 **CHANGE HISTORY**

46478 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

46479 **Issue 6**

46480 The following new requirements on POSIX implementations derive from alignment with the  
46481 Single UNIX Specification:

- 46482 • In the DESCRIPTION, text previously conditional on `_POSIX_JOB_CONTROL` is now
- 46483 mandated. This is a FIPS requirement.
- 46484 • The [EIO] error is added.

46485 **NAME**

46486 tcsetattr — set the parameters associated with the terminal

46487 **SYNOPSIS**

46488 #include &lt;termios.h&gt;

46489 int tcsetattr(int *fildev*, int *optional\_actions*,  
46490 const struct termios \**termios\_p*);46491 **DESCRIPTION**46492 The *tcsetattr()* function shall set the parameters associated with the terminal referred to by the  
46493 open file descriptor *fildev* (an open file descriptor associated with a terminal) from the **termios**  
46494 structure referenced by *termios\_p* as follows:

- 46495
- If *optional\_actions* is TCSANOW, the change shall occur immediately.
  - 46496 • If *optional\_actions* is TCSADRAIN, the change shall occur after all output written to *fildev* is  
46497 transmitted. This function should be used when changing parameters that affect output.
  - 46498 • If *optional\_actions* is TCSAFLUSH, the change shall occur after all output written to *fildev* is  
46499 transmitted, and all input so far received but not read shall be discarded before the change is  
46500 made.

46501 If the output baud rate stored in the **termios** structure pointed to by *termios\_p* is the zero baud  
46502 rate, B0, the modem control lines shall no longer be asserted. Normally, this shall disconnect the  
46503 line.46504 If the input baud rate stored in the **termios** structure pointed to by *termios\_p* is 0, the input baud  
46505 rate given to the hardware is the same as the output baud rate stored in the **termios** structure.46506 The *tcsetattr()* function shall return successfully if it was able to perform any of the requested  
46507 actions, even if some of the requested actions could not be performed. It shall set all the  
46508 attributes that the implementation supports as requested and leave all the attributes not  
46509 supported by the implementation unchanged. If no part of the request can be honored, it shall  
46510 return `-1` and set *errno* to `[EINVAL]`. If the input and output baud rates differ and are a  
46511 combination that is not supported, neither baud rate shall be changed. A subsequent call to  
46512 *tcgetattr()* shall return the actual state of the terminal device (reflecting both the changes made  
46513 and not made in the previous *tcsetattr()* call). The *tcsetattr()* function shall not change the values  
46514 found in the **termios** structure under any circumstances.46515 The effect of *tcsetattr()* is undefined if the value of the **termios** structure pointed to by *termios\_p*  
46516 was not derived from the result of a call to *tcgetattr()* on *fildev*; an application should modify  
46517 only fields and flags defined by this volume of IEEE Std 1003.1-2001 between the call to  
46518 *tcgetattr()* and *tcsetattr()*, leaving all other fields and flags unmodified.46519 No actions defined by this volume of IEEE Std 1003.1-2001, other than a call to *tcsetattr()* or a  
46520 close of the last file descriptor in the system associated with this terminal device, shall cause any  
46521 of the terminal attributes defined by this volume of IEEE Std 1003.1-2001 to change.46522 If *tcsetattr()* is called from a process which is a member of a background process group on a  
46523 *fildev* associated with its controlling terminal:

- 46524
- If the calling process is blocking or ignoring SIGTTOU signals, the operation completes  
46525 normally and no signal is sent.
  - 46526 • Otherwise, a SIGTTOU signal shall be sent to the process group.

46527 **RETURN VALUE**

46528       Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
46529       indicate the error.

46530 **ERRORS**

46531       The *tcsetattr()* function shall fail if:

46532       [EBADF]       The *fildev* argument is not a valid file descriptor.

46533       [EINTR]       A signal interrupted *tcsetattr()*.

46534       [EINVAL]       The *optional\_actions* argument is not a supported value, or an attempt was  
46535       made to change an attribute represented in the **termios** structure to an  
46536       unsupported value.

46537       [ENOTTY]       The file associated with *fildev* is not a terminal.

46538       The *tcsetattr()* function may fail if:

46539       [EIO]       The process group of the writing process is orphaned, and the writing process  
46540       is not ignoring or blocking SIGTTOU.

46541 **EXAMPLES**

46542       None.

46543 **APPLICATION USAGE**

46544       If trying to change baud rates, applications should call *tcsetattr()* then call *tcgetattr()* in order to  
46545       determine what baud rates were actually selected.

46546 **RATIONALE**

46547       The *tcsetattr()* function can be interrupted in the following situations:

- 46548       • It is interrupted while waiting for output to drain.
- 46549       • It is called from a process in a background process group and SIGTTOU is caught.

46550       See also the RATIONALE section in *tcgetattr()*.

46551 **FUTURE DIRECTIONS**

46552       Using an input baud rate of 0 to set the input rate equal to the output rate may not necessarily be  
46553       supported in a future version of this volume of IEEE Std 1003.1-2001.

46554 **SEE ALSO**

46555       *cfgetispeed()*, *tcgetattr()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 11,  
46556       General Terminal Interface, <**termios.h**>, <**unistd.h**>

46557 **CHANGE HISTORY**

46558       First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

46559 **Issue 6**

46560       The following new requirements on POSIX implementations derive from alignment with the  
46561       Single UNIX Specification:

- 46562       • In the DESCRIPTION, text previously conditional on `_POSIX_JOB_CONTROL` is now  
46563       mandated. This is a FIPS requirement.
- 46564       • The [EIO] error is added.

46565  
46566

In the DESCRIPTION, the text describing use of *tcsetattr()* from a process which is a member of a background process group is clarified.

46567 **NAME**

46568 tcsetpgrp — set the foreground process group ID

46569 **SYNOPSIS**

46570 #include &lt;unistd.h&gt;

46571 int tcsetpgrp(int *fildev*, pid\_t *pgid\_id*);46572 **DESCRIPTION**

46573 If the process has a controlling terminal, *tcsetpgrp()* shall set the foreground process group ID  
 46574 associated with the terminal to *pgid\_id*. The application shall ensure that the file associated with  
 46575 *fildev* is the controlling terminal of the calling process and the controlling terminal is currently  
 46576 associated with the session of the calling process. The application shall ensure that the value of  
 46577 *pgid\_id* matches a process group ID of a process in the same session as the calling process.

46578 Attempts to use *tcsetpgrp()* from a process which is a member of a background process group on  
 46579 a *fildev* associated with its controlling terminal shall cause the process group to be sent a  
 46580 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process  
 46581 shall be allowed to perform the operation, and no signal is sent.

46582 **RETURN VALUE**

46583 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 46584 indicate the error.

46585 **ERRORS**46586 The *tcsetpgrp()* function shall fail if:

- |       |          |                                                                                                                                                                                                                       |
|-------|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 46587 | [EBADF]  | The <i>fildev</i> argument is not a valid file descriptor.                                                                                                                                                            |
| 46588 | [EINVAL] | This implementation does not support the value in the <i>pgid_id</i> argument.                                                                                                                                        |
| 46589 | [ENOTTY] | The calling process does not have a controlling terminal, or the file is not the<br>46590 controlling terminal, or the controlling terminal is no longer associated with<br>46591 the session of the calling process. |
| 46592 | [EPERM]  | The value of <i>pgid_id</i> is a value supported by the implementation, but does not<br>46593 match the process group ID of a process in the same session as the calling<br>46594 process.                            |

46595 **EXAMPLES**

46596 None.

46597 **APPLICATION USAGE**

46598 None.

46599 **RATIONALE**

46600 None.

46601 **FUTURE DIRECTIONS**

46602 None.

46603 **SEE ALSO**46604 *tcgetpgrp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/types.h>, <unistd.h>46605 **CHANGE HISTORY**

46606 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

46607 **Issue 6**

46608 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

46609 The following new requirements on POSIX implementations derive from alignment with the  
46610 Single UNIX Specification:

46611 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
46612 required for conforming implementations of previous POSIX specifications, it was not  
46613 required for UNIX applications.

46614 • In the DESCRIPTION and ERRORS sections, text previously conditional on  
46615 `_POSIX_JOB_CONTROL` is now mandated. This is a FIPS requirement.

46616 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

46617 The Open Group Corrigendum U047/4 is applied.

## 46618 NAME

46619 tdelete, tfind, tsearch, twalk — manage a binary search tree

## 46620 SYNOPSIS

```

46621 XSI #include <search.h>
46622 void *tdelete(const void *restrict key, void **restrict rootp,
46623 int(*compar)(const void *, const void *));
46624 void *tfind(const void *key, void *const *rootp,
46625 int(*compar)(const void *, const void *));
46626 void *tsearch(const void *key, void **rootp,
46627 int (*compar)(const void *, const void *));
46628 void twalk(const void *root,
46629 void (*action)(const void *, VISIT, int));
46630

```

## 46631 DESCRIPTION

46632 The *tdelete()*, *tfind()*, *tsearch()*, and *twalk()* functions manipulate binary search trees.  
 46633 Comparisons are made with a user-supplied routine, the address of which is passed as the  
 46634 *compar* argument. This routine is called with two arguments, which are the pointers to the  
 46635 elements being compared. The application shall ensure that the user-supplied routine returns an  
 46636 integer less than, equal to, or greater than 0, according to whether the first argument is to be  
 46637 considered less than, equal to, or greater than the second argument. The comparison function  
 46638 need not compare every byte, so arbitrary data may be contained in the elements in addition to  
 46639 the values being compared.

46640 The *tsearch()* function shall build and access the tree. The *key* argument is a pointer to an element  
 46641 to be accessed or stored. If there is a node in the tree whose element is equal to the value pointed  
 46642 to by *key*, a pointer to this found node shall be returned. Otherwise, the value pointed to by *key*  
 46643 shall be inserted (that is, a new node is created and the value of *key* is copied to this node), and a  
 46644 pointer to this node returned. Only pointers are copied, so the application shall ensure that the  
 46645 calling routine stores the data. The *rootp* argument points to a variable that points to the root  
 46646 node of the tree. A null pointer value for the variable pointed to by *rootp* denotes an empty tree;  
 46647 in this case, the variable shall be set to point to the node which shall be at the root of the new  
 46648 tree.

46649 Like *tsearch()*, *tfind()* shall search for a node in the tree, returning a pointer to it if found.  
 46650 However, if it is not found, *tfind()* shall return a null pointer. The arguments for *tfind()* are the  
 46651 same as for *tsearch()*.

46652 The *tdelete()* function shall delete a node from a binary search tree. The arguments are the same  
 46653 as for *tsearch()*. The variable pointed to by *rootp* shall be changed if the deleted node was the  
 46654 root of the tree. The *tdelete()* function shall return a pointer to the parent of the deleted node, or a  
 46655 null pointer if the node is not found.

46656 The *twalk()* function shall traverse a binary search tree. The *root* argument is a pointer to the root  
 46657 node of the tree to be traversed. (Any node in a tree may be used as the root for a walk below  
 46658 that node.) The argument *action* is the name of a routine to be invoked at each node. This routine  
 46659 is, in turn, called with three arguments. The first argument shall be the address of the node being  
 46660 visited. The structure pointed to by this argument is unspecified and shall not be modified by  
 46661 the application, but it shall be possible to cast a pointer-to-node into a pointer-to-pointer-to-  
 46662 element to access the element stored in the node. The second argument shall be a value from an  
 46663 enumeration data type:

```

46664 typedef enum { preorder, postorder, endorder, leaf } VISIT;

```

46665 (defined in `<search.h>`), depending on whether this is the first, second, or third time that the  
 46666 node is visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a  
 46667 leaf. The third argument shall be the level of the node in the tree, with the root being level 0.

46668 If the calling function alters the pointer to the root, the result is undefined.

#### 46669 RETURN VALUE

46670 If the node is found, both `tsearch()` and `tfind()` shall return a pointer to it. If not, `tfind()` shall  
 46671 return a null pointer, and `tsearch()` shall return a pointer to the inserted item.

46672 A null pointer shall be returned by `tsearch()` if there is not enough space available to create a new  
 46673 node.

46674 A null pointer shall be returned by `tdelete()`, `tfind()`, and `tsearch()` if `rootp` is a null pointer on  
 46675 entry.

46676 The `tdelete()` function shall return a pointer to the parent of the deleted node, or a null pointer if  
 46677 the node is not found.

46678 The `twalk()` function shall not return a value.

#### 46679 ERRORS

46680 No errors are defined.

#### 46681 EXAMPLES

46682 The following code reads in strings and stores structures containing a pointer to each string and  
 46683 a count of its length. It then walks the tree, printing out the stored strings and their lengths in  
 46684 alphabetical order.

```

46685 #include <search.h>
46686 #include <string.h>
46687 #include <stdio.h>

46688 #define STRSZ 10000
46689 #define NODSZ 500

46690 struct node { /* Pointers to these are stored in the tree. */
46691 char *string;
46692 int length;
46693 };

46694 char string_space[STRSZ]; /* Space to store strings. */
46695 struct node nodes[NODSZ]; /* Nodes to store. */
46696 void *root = NULL; /* This points to the root. */

46697 int main(int argc, char *argv[])
46698 {
46699 char *strptr = string_space;
46700 struct node *nodeptr = nodes;
46701 void print_node(const void *, VISIT, int);
46702 int i = 0, node_compare(const void *, const void *);

46703 while (gets(strptr) != NULL && i++ < NODSZ) {
46704 /* Set node. */
46705 nodeptr->string = strptr;
46706 nodeptr->length = strlen(strptr);
46707 /* Put node into the tree. */
46708 (void) tsearch((void *)nodeptr, (void **)&root,
46709 node_compare);

```

```

46710 /* Adjust pointers, so we do not overwrite tree. */
46711 strptr += nodeptr->length + 1;
46712 nodeptr++;
46713 }
46714 twalk(root, print_node);
46715 return 0;
46716 }

46717 /*
46718 * This routine compares two nodes, based on an
46719 * alphabetical ordering of the string field.
46720 */
46721 int
46722 node_compare(const void *node1, const void *node2)
46723 {
46724 return strcmp(((const struct node *) node1)->string,
46725 ((const struct node *) node2)->string);
46726 }

46727 /*
46728 * This routine prints out a node, the second time
46729 * twalk encounters it or if it is a leaf.
46730 */
46731 void
46732 print_node(const void *ptr, VISIT order, int level)
46733 {
46734 const struct node *p = *(const struct node **) ptr;
46735 if (order == postorder || order == leaf) {
46736 (void) printf("string = %s, length = %d\n",
46737 p->string, p->length);
46738 }
46739 }

```

#### 46740 APPLICATION USAGE

46741 The *root* argument to *twalk()* is one level of indirection less than the *rootp* arguments to *tdelete()*  
 46742 and *tsearch()*.

46743 There are two nomenclatures used to refer to the order in which tree nodes are visited. The  
 46744 *tsearch()* function uses **preorder**, **postorder**, and **endorder** to refer respectively to visiting a node  
 46745 before any of its children, after its left child and before its right, and after both its children. The  
 46746 alternative nomenclature uses **preorder**, **inorder**, and **postorder** to refer to the same visits, which  
 46747 could result in some confusion over the meaning of **postorder**.

#### 46748 RATIONALE

46749 None.

#### 46750 FUTURE DIRECTIONS

46751 None.

#### 46752 SEE ALSO

46753 *hcreate()*, *tsearch()*, the Base Definitions volume of IEEE Std 1003.1-2001, <[search.h](#)>

46754 **CHANGE HISTORY**

46755 First released in Issue 1. Derived from Issue 1 of the SVID.

46756 **Issue 5**

46757 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
46758 previous issues.

46759 **Issue 6**

46760 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

46761 The **restrict** keyword is added to the *tdelete()* prototype for alignment with the  
46762 ISO/IEC 9899:1999 standard.

46763 **NAME**

46764 tellmdir — current location of a named directory stream

46765 **SYNOPSIS**

46766 XSI #include &lt;dirent.h&gt;

46767 long tellmdir(DIR \*dirp);

46768

46769 **DESCRIPTION**46770 The *tellmdir()* function shall obtain the current location associated with the directory stream  
46771 specified by *dirp*.46772 If the most recent operation on the directory stream was a *seekdir()*, the directory position  
46773 returned from the *tellmdir()* shall be the same as that supplied as a *loc* argument for *seekdir()*.46774 **RETURN VALUE**46775 Upon successful completion, *tellmdir()* shall return the current location of the specified directory  
46776 stream.46777 **ERRORS**

46778 No errors are defined.

46779 **EXAMPLES**

46780 None.

46781 **APPLICATION USAGE**

46782 None.

46783 **RATIONALE**

46784 None.

46785 **FUTURE DIRECTIONS**

46786 None.

46787 **SEE ALSO**46788 *opendir()*, *readdir()*, *seekdir()*, the Base Definitions volume of IEEE Std 1003.1-2001, <dirent.h>46789 **CHANGE HISTORY**

46790 First released in Issue 2.

46791 **NAME**

46792 tempnam — create a name for a temporary file

46793 **SYNOPSIS**46794 XSI 

```
#include <stdio.h>
```

46795 

```
char *tempnam(const char *dir, const char *pfx);
```

46796

46797 **DESCRIPTION**46798 The *tempnam()* function shall generate a pathname that may be used for a temporary file.

46799 The *tempnam()* function allows the user to control the choice of a directory. The *dir* argument  
 46800 points to the name of the directory in which the file is to be created. If *dir* is a null pointer or  
 46801 points to a string which is not a name for an appropriate directory, the path prefix defined as  
 46802 P\_tmpdir in the <stdio.h> header shall be used. If that directory is not accessible, an  
 46803 implementation-defined directory may be used.

46804 Many applications prefer their temporary files to have certain initial letter sequences in their  
 46805 names. The *pfx* argument should be used for this. This argument may be a null pointer or point  
 46806 to a string of up to five bytes to be used as the beginning of the filename.

46807 Some implementations of *tempnam()* may use *tmpnam()* internally. On such implementations, if  
 46808 called more than {TMP\_MAX} times in a single process, the behavior is implementation-defined.

46809 **RETURN VALUE**

46810 Upon successful completion, *tempnam()* shall allocate space for a string, put the generated  
 46811 pathname in that space, and return a pointer to it. The pointer shall be suitable for use in a  
 46812 subsequent call to *free()*. Otherwise, it shall return a null pointer and set *errno* to indicate the  
 46813 error.

46814 **ERRORS**46815 The *tempnam()* function shall fail if:

46816 [ENOMEM] Insufficient storage space is available.

46817 **EXAMPLES**46818 **Generating a Pathname**

46819 The following example generates a pathname for a temporary file in directory **/tmp**, with the  
 46820 prefix *file*. After the filename has been created, the call to *free()* deallocates the space used to  
 46821 store the filename.

```
46822 #include <stdio.h>
46823 #include <stdlib.h>
46824 ...
46825 char *directory = "/tmp";
46826 char *fileprefix = "file";
46827 char *file;

46828 file = tempnam(directory, fileprefix);
46829 free(file);
```

46830 **APPLICATION USAGE**

46831 This function only creates pathnames. It is the application's responsibility to create and remove  
 46832 the files. Between the time a pathname is created and the file is opened, it is possible for some  
 46833 other process to create a file with the same name. Applications may find *tmpfile()* more useful.

46834 **RATIONALE**

46835           None.

46836 **FUTURE DIRECTIONS**

46837           None.

46838 **SEE ALSO**

46839           *fopen()*, *free()*, *open()*, *tmpfile()*, *tmpnam()*, *unlink()*, the Base Definitions volume of  
46840           IEEE Std 1003.1-2001, <**stdio.h**>

46841 **CHANGE HISTORY**

46842           First released in Issue 1. Derived from Issue 1 of the SVID.

46843 **Issue 5**

46844           The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
46845           previous issues.

46846 **NAME**

46847 tfind — search binary search tree

46848 **SYNOPSIS**

46849 XSI #include &lt;search.h&gt;

46850 void \*tfind(const void \*key, void \*const \*rootp,  
46851 int (\*compar)(const void \*, const void \*));

46852

46853 **DESCRIPTION**46854 Refer to *tdelete()*.

46855 **NAME**

46856 `tgamma`, `tgammaf`, `tgammal` — compute `gamma()` function

46857 **SYNOPSIS**

```
46858 #include <math.h>

46859 double tgamma(double x);
46860 float tgammaf(float x);
46861 long double tgammal(long double x);
```

46862 **DESCRIPTION**

46863 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 46864 conflict between the requirements described here and the ISO C standard is unintentional. This  
 46865 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

46866 These functions shall compute the *gamma()* function of *x*.

46867 An application wishing to check for error situations should set *errno* to zero and call  
 46868 *feclearexcept*(*FE\_ALL\_EXCEPT*) before calling these functions. On return, if *errno* is non-zero or  
 46869 *fetestexcept*(*FE\_INVALID* | *FE\_DIVBYZERO* | *FE\_OVERFLOW* | *FE\_UNDERFLOW*) is non-  
 46870 zero, an error has occurred.

46871 **RETURN VALUE**

46872 Upon successful completion, these functions shall return *Gamma(x)*.

46873 If *x* is a negative integer, a domain error shall occur, and either a NaN (if supported), or an  
 46874 implementation-defined value shall be returned.

46875 If the correct value would cause overflow, a range error shall occur and *tgamma()*, *tgammaf()*,  
 46876 and *tgammal()* shall return  $\pm$ HUGE\_VAL,  $\pm$ HUGE\_VALF, or  $\pm$ HUGE\_VALL, respectively, with  
 46877 the same sign as the correct value of the function.

46878 **MX** If *x* is NaN, a NaN shall be returned.

46879 If *x* is +Inf, *x* shall be returned.

46880 If *x* is  $\pm 0$ , a pole error shall occur, and *tgamma()*, *tgammaf()*, and *tgammal()* shall return  
 46881  $\pm$ HUGE\_VAL,  $\pm$ HUGE\_VALF, and  $\pm$ HUGE\_VALL, respectively.

46882 If *x* is -Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-  
 46883 defined value shall be returned.

46884 **ERRORS**

46885 These functions shall fail if:

46886 **MX** Domain Error The value of *x* is a negative integer, or *x* is -Inf.

46887 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 46888 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling  
 46889 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 46890 shall be raised.

46891 **MX** Pole Error The value of *x* is zero.

46892 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 46893 then *errno* shall be set to [ERANGE]. If the integer expression  
 46894 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the divide-by-  
 46895 zero floating-point exception shall be raised.

46896           Range Error       The value overflows.

46897                            If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
46898                            then *errno* shall be set to [ERANGE]. If the integer expression  
46899                            (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow  
46900                            floating-point exception shall be raised.

46901 **EXAMPLES**

46902           None.

46903 **APPLICATION USAGE**

46904           For IEEE Std 754-1985 **double**, overflow happens when  $0 < x < 1/\text{DBL\_MAX}$ , and  $171.7 < x$ .

46905           On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
46906           MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

46907 **RATIONALE**

46908           This function is named *tgamma()* in order to avoid conflicts with the historical *gamma()* and  
46909           *lgamma()* functions.

46910 **FUTURE DIRECTIONS**

46911           It is possible that the error response for a negative integer argument may be changed to a pole  
46912           error and a return value of  $\pm\text{Inf}$ .

46913 **SEE ALSO**

46914           *feclearexcept()*, *fetestexcept()*, *lgamma()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
46915           Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

46916 **CHANGE HISTORY**

46917           First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

46918           IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/65 is applied, correcting the third |  
46919           paragraph in the RETURN VALUE section. |

## 46920 NAME

46921 time — get time

## 46922 SYNOPSIS

46923 #include &lt;time.h&gt;

46924 time\_t time(time\_t \*tloc);

## 46925 DESCRIPTION

46926 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 46927 conflict between the requirements described here and the ISO C standard is unintentional. This  
 46928 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

46929 CX The *time()* function shall return the value of time in seconds since the Epoch.

46930 The *tloc* argument points to an area where the return value is also stored. If *tloc* is a null pointer,  
 46931 no value is stored.

## 46932 RETURN VALUE

46933 Upon successful completion, *time()* shall return the value of time. Otherwise, (**time\_t**)−1 shall be  
 46934 returned.

## 46935 ERRORS

46936 No errors are defined.

## 46937 EXAMPLES

46938 **Getting the Current Time**

46939 The following example uses the *time()* function to calculate the time elapsed, in seconds, since  
 46940 the Epoch, *localtime()* to convert that value to a broken-down time, and *asctime()* to convert the  
 46941 broken-down time values into a printable string.

```
46942 #include <stdio.h>
46943 #include <time.h>
46944 int main(void)
46945 {
46946 time_t result;
46947 result = time(NULL);
46948 printf("%s%ju secs since the Epoch\n",
46949 asctime(localtime(&result)),
46950 (uintmax_t)result);
46951 return(0);
46952 }
```

46953 This example writes the current time to *stdout* in a form like this:

```
46954 Wed Jun 26 10:32:15 1996
46955 835810335 secs since the Epoch
```

46956 **Timing an Event**

46957 The following example gets the current time, prints it out in the user's format, and prints the  
46958 number of minutes to an event being timed.

```
46959 #include <time.h>
46960 #include <stdio.h>
46961 ...
46962 time_t now;
46963 int minutes_to_event;
46964 ...
46965 time(&now);
46966 minutes_to_event = ...;
46967 printf("The time is ");
46968 puts(asctime(localtime(&now)));
46969 printf("There are %d minutes to the event.\n",
46970 minutes_to_event);
46971 ...
```

46972 **APPLICATION USAGE**

46973 None.

46974 **RATIONALE**

46975 The *time()* function returns a value in seconds (type **time\_t**) while *times()* returns a set of values  
46976 in clock ticks (type **clock\_t**). Some historical implementations, such as 4.3 BSD, have  
46977 mechanisms capable of returning more precise times (see below). A generalized timing scheme  
46978 to unify these various timing mechanisms has been proposed but not adopted.

46979 Implementations in which **time\_t** is a 32-bit signed integer (many historical implementations)  
46980 fail in the year 2038. IEEE Std 1003.1-2001 does not address this problem. However, the use of  
46981 the **time\_t** type is mandated in order to ease the eventual fix.

46982 The use of the **<time.h>** header instead of **<sys/types.h>** allows compatibility with the ISO C  
46983 standard.

46984 Many historical implementations (including Version 7) and the 1984 /usr/group standard use  
46985 **long** instead of **time\_t**. This volume of IEEE Std 1003.1-2001 uses the latter type in order to agree  
46986 with the ISO C standard.

46987 4.3 BSD includes *time()* only as an alternate function to the more flexible *gettimeofday()* function.

46988 **FUTURE DIRECTIONS**

46989 In a future version of this volume of IEEE Std 1003.1-2001, **time\_t** is likely to be required to be  
46990 capable of representing times far in the future. Whether this will be mandated as a 64-bit type or  
46991 a requirement that a specific date in the future be representable (for example, 10000 AD) is not  
46992 yet determined. Systems purchased after the approval of this volume of IEEE Std 1003.1-2001  
46993 should be evaluated to determine whether their lifetime will extend past 2038.

46994 **SEE ALSO**

46995 *asctime()*, *clock()*, *ctime()*, *difftime()*, *gettimeofday()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*,  
46996 *strptime()*, *utime()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<time.h>**

46997 **CHANGE HISTORY**

46998 First released in Issue 1. Derived from Issue 1 of the SVID.

46999 **Issue 6**

47000 Extensions beyond the ISO C standard are marked.

47001 The EXAMPLES, RATIONALE, and FUTURE DIRECTIONS sections are added.

## 47002 NAME

47003 timer\_create — create a per-process timer (**REALTIME**)

## 47004 SYNOPSIS

47005 TMR #include &lt;signal.h&gt;

47006 #include &lt;time.h&gt;

47007 int timer\_create(clockid\_t clockid, struct sigevent \*restrict evp,

47008 timer\_t \*restrict timerid);

47009

## 47010 DESCRIPTION

47011 The *timer\_create()* function shall create a per-process timer using the specified clock, *clock\_id*, as  
 47012 the timing base. The *timer\_create()* function shall return, in the location referenced by *timerid*, a  
 47013 timer ID of type **timer\_t** used to identify the timer in timer requests. This timer ID shall be  
 47014 unique within the calling process until the timer is deleted. The particular clock, *clock\_id*, is  
 47015 defined in <**time.h**>. The timer whose ID is returned shall be in a disarmed state upon return  
 47016 from *timer\_create()*.

47017 The *evp* argument, if non-NULL, points to a **sigevent** structure. This structure, allocated by the  
 47018 application, defines the asynchronous notification to occur as specified in Section 2.4.1 (on page  
 47019 28) when the timer expires. If the *evp* argument is NULL, the effect is as if the *evp* argument  
 47020 pointed to a **sigevent** structure with the *sigev\_notify* member having the value SIGEV\_SIGNAL,  
 47021 the *sigev\_signo* having a default signal number, and the *sigev\_value* member having the value of  
 47022 the timer ID.

47023 Each implementation shall define a set of clocks that can be used as timing bases for per-process  
 47024 MON timers. All implementations shall support a *clock\_id* of CLOCK\_REALTIME. If the Monotonic  
 47025 Clock option is supported, implementations shall support a *clock\_id* of CLOCK\_MONOTONIC.

47026 Per-process timers shall not be inherited by a child process across a *fork()* and shall be disarmed  
 47027 and deleted by an *exec*.

47028 CPT If \_POSIX\_CPUTIME is defined, implementations shall support *clock\_id* values representing the  
 47029 CPU-time clock of the calling process.

47030 TCT If \_POSIX\_THREAD\_CPUTIME is defined, implementations shall support *clock\_id* values  
 47031 representing the CPU-time clock of the calling thread.

47032 CPT|TCT It is implementation-defined whether a *timer\_create()* function will succeed if the value defined  
 47033 by *clock\_id* corresponds to the CPU-time clock of a process or thread different from the process  
 47034 or thread invoking the function.

## 47035 RETURN VALUE

47036 If the call succeeds, *timer\_create()* shall return zero and update the location referenced by *timerid*  
 47037 to a **timer\_t**, which can be passed to the per-process timer calls. If an error occurs, the function  
 47038 shall return a value of -1 and set *errno* to indicate the error. The value of *timerid* is undefined if  
 47039 an error occurs.

## 47040 ERRORS

47041 The *timer\_create()* function shall fail if:

47042 [EAGAIN] The system lacks sufficient signal queuing resources to honor the request.

47043 [EAGAIN] The calling process has already created all of the timers it is allowed by this  
47044 implementation.

47045 [EINVAL] The specified clock ID is not defined.

47046 CPT|TCT [ENOTSUP] The implementation does not support the creation of a timer attached to the  
47047 CPU-time clock that is specified by *clock\_id* and associated with a process or  
47048 thread different from the process or thread invoking *timer\_create()*.

#### 47049 EXAMPLES

47050 None.

#### 47051 APPLICATION USAGE

47052 None.

#### 47053 RATIONALE

##### 47054 Periodic Timer Overrun and Resource Allocation

47055 The specified timer facilities may deliver realtime signals (that is, queued signals) on  
47056 implementations that support this option. Since realtime applications cannot afford to lose  
47057 notifications of asynchronous events, like timer expirations or asynchronous I/O completions, it  
47058 must be possible to ensure that sufficient resources exist to deliver the signal when the event  
47059 occurs. In general, this is not a difficulty because there is a one-to-one correspondence between a  
47060 request and a subsequent signal generation. If the request cannot allocate the signal delivery  
47061 resources, it can fail the call with an [EAGAIN] error.

47062 Periodic timers are a special case. A single request can generate an unspecified number of  
47063 signals. This is not a problem if the requesting process can service the signals as fast as they are  
47064 generated, thus making the signal delivery resources available for delivery of subsequent  
47065 periodic timer expiration signals. But, in general, this cannot be assured—processing of periodic  
47066 timer signals may “overrun”; that is, subsequent periodic timer expirations may occur before the  
47067 currently pending signal has been delivered.

47068 Also, for signals, according to the POSIX.1-1990 standard, if subsequent occurrences of a  
47069 pending signal are generated, it is implementation-defined whether a signal is delivered for each  
47070 occurrence. This is not adequate for some realtime applications. So a mechanism is required to  
47071 allow applications to detect how many timer expirations were delayed without requiring an  
47072 indefinite amount of system resources to store the delayed expirations.

47073 The specified facilities provide for an overrun count. The overrun count is defined as the number  
47074 of extra timer expirations that occurred between the time a timer expiration signal is generated  
47075 and the time the signal is delivered. The signal-catching function, if it is concerned with  
47076 overruns, can retrieve this count on entry. With this method, a periodic timer only needs one  
47077 “signal queuing resource” that can be allocated at the time of the *timer\_create()* function call.

47078 A function is defined to retrieve the overrun count so that an application need not allocate static  
47079 storage to contain the count, and an implementation need not update this storage  
47080 asynchronously on timer expirations. But, for some high-frequency periodic applications, the  
47081 overhead of an additional system call on each timer expiration may be prohibitive. The  
47082 functions, as defined, permit an implementation to maintain the overrun count in user space,  
47083 associated with the *timerid*. The *timer\_getoverrun()* function can then be implemented as a macro  
47084 that uses the *timerid* argument (which may just be a pointer to a user space structure containing  
47085 the counter) to locate the overrun count with no system call overhead. Other implementations,  
47086 less concerned with this class of applications, can avoid the asynchronous update of user space  
47087 by maintaining the count in a system structure at the cost of the extra system call to obtain it.

47088 **Timer Expiration Signal Parameters**

47089 The Realtime Signals Extension option supports an application-specific datum that is delivered  
47090 to the extended signal handler. This value is explicitly specified by the application, along with  
47091 the signal number to be delivered, in a **sigevent** structure. The type of the application-defined  
47092 value can be either an integer constant or a pointer. This explicit specification of the value, as  
47093 opposed to always sending the timer ID, was selected based on existing practice.

47094 It is common practice for realtime applications (on non-POSIX systems or realtime extended  
47095 POSIX systems) to use the parameters of event handlers as the case label of a switch statement  
47096 or as a pointer to an application-defined data structure. Since *timer\_ids* are dynamically allocated  
47097 by the *timer\_create()* function, they can be used for neither of these functions without additional  
47098 application overhead in the signal handler; for example, to search an array of saved timer IDs to  
47099 associate the ID with a constant or application data structure.

47100 **FUTURE DIRECTIONS**

47101 None.

47102 **SEE ALSO**

47103 *clock\_getres()*, *timer\_delete()*, *timer\_getoverrun()*, the Base Definitions volume of  
47104 IEEE Std 1003.1-2001, <**time.h**>

47105 **CHANGE HISTORY**

47106 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

47107 **Issue 6**

47108 The *timer\_create()* function is marked as part of the Timers option.

47109 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
47110 implementation does not support the Timers option.

47111 CPU-time clocks are added for alignment with IEEE Std 1003.1d-1999.

47112 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding the  
47113 requirement for the CLOCK\_MONOTONIC clock under the Monotonic Clock option.

47114 The **restrict** keyword is added to the *timer\_create()* prototype for alignment with the  
47115 ISO/IEC 9899:1999 standard.

47116 **NAME**

47117 timer\_delete — delete a per-process timer (**REALTIME**)

47118 **SYNOPSIS**

47119 TMR #include <time.h>

47120 int timer\_delete(timer\_t timerid);

47121

47122 **DESCRIPTION**

47123 The *timer\_delete()* function deletes the specified timer, *timerid*, previously created by the  
47124 *timer\_create()* function. If the timer is armed when *timer\_delete()* is called, the behavior shall be  
47125 as if the timer is automatically disarmed before removal. The disposition of pending signals for  
47126 the deleted timer is unspecified.

47127 **RETURN VALUE**

47128 If successful, the *timer\_delete()* function shall return a value of zero. Otherwise, the function shall  
47129 return a value of  $-1$  and set *errno* to indicate the error.

47130 **ERRORS**

47131 The *timer\_delete()* function shall fail if:

47132 [EINVAL] The timer ID specified by *timerid* is not a valid timer ID.

47133 **EXAMPLES**

47134 None.

47135 **APPLICATION USAGE**

47136 None.

47137 **RATIONALE**

47138 None.

47139 **FUTURE DIRECTIONS**

47140 None.

47141 **SEE ALSO**

47142 *timer\_create()*, the Base Definitions volume of IEEE Std 1003.1-2001, <time.h>

47143 **CHANGE HISTORY**

47144 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

47145 **Issue 6**

47146 The *timer\_delete()* function is marked as part of the Timers option.

47147 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
47148 implementation does not support the Timers option.

## 47149 NAME

47150 timer\_getoverrun, timer\_gettime, timer\_settime — per-process timers (**REALTIME**)

## 47151 SYNOPSIS

47152 TMR 

```
#include <time.h>
```

```
47153 int timer_getoverrun(timer_t timerid);
47154 int timer_gettime(timer_t timerid, struct itimerspec *value);
47155 int timer_settime(timer_t timerid, int flags,
47156 const struct itimerspec *restrict value,
47157 struct itimerspec *restrict ovalue);
47158
```

## 47159 DESCRIPTION

47160 The *timer\_gettime()* function shall store the amount of time until the specified timer, *timerid*,  
 47161 expires and the reload value of the timer into the space pointed to by the *value* argument. The  
 47162 *it\_value* member of this structure shall contain the amount of time before the timer expires, or  
 47163 zero if the timer is disarmed. This value is returned as the interval until timer expiration, even if  
 47164 the timer was armed with absolute time. The *it\_interval* member of *value* shall contain the reload  
 47165 value last set by *timer\_settime()*.

47166 The *timer\_settime()* function shall set the time until the next expiration of the timer specified by  
 47167 *timerid* from the *it\_value* member of the *value* argument and arm the timer if the *it\_value* member  
 47168 of *value* is non-zero. If the specified timer was already armed when *timer\_settime()* is called, this  
 47169 call shall reset the time until next expiration to the *value* specified. If the *it\_value* member of *value*  
 47170 is zero, the timer shall be disarmed. The effect of disarming or resetting a timer with pending  
 47171 expiration notifications is unspecified.

47172 If the flag **TIMER\_ABSTIME** is not set in the argument *flags*, *timer\_settime()* shall behave as if the  
 47173 time until next expiration is set to be equal to the interval specified by the *it\_value* member of  
 47174 *value*. That is, the timer shall expire in *it\_value* nanoseconds from when the call is made. If the  
 47175 flag **TIMER\_ABSTIME** is set in the argument *flags*, *timer\_settime()* shall behave as if the time  
 47176 until next expiration is set to be equal to the difference between the absolute time specified by  
 47177 the *it\_value* member of *value* and the current value of the clock associated with *timerid*. That is,  
 47178 the timer shall expire when the clock reaches the value specified by the *it\_value* member of *value*.  
 47179 If the specified time has already passed, the function shall succeed and the expiration  
 47180 notification shall be made.

47181 The reload value of the timer shall be set to the value specified by the *it\_interval* member of  
 47182 *value*. When a timer is armed with a non-zero *it\_interval*, a periodic (or repetitive) timer is  
 47183 specified.

47184 Time values that are between two consecutive non-negative integer multiples of the resolution  
 47185 of the specified timer shall be rounded up to the larger multiple of the resolution. Quantization  
 47186 error shall not cause the timer to expire earlier than the rounded time value.

47187 If the argument *ovalue* is not NULL, the *timer\_settime()* function shall store, in the location  
 47188 referenced by *ovalue*, a value representing the previous amount of time before the timer would  
 47189 have expired, or zero if the timer was disarmed, together with the previous timer reload value.  
 47190 Timers shall not expire before their scheduled time.

47191 Only a single signal shall be queued to the process for a given timer at any point in time. When a  
 47192 timer for which a signal is still pending expires, no signal shall be queued, and a timer overrun  
 47193 shall occur. When a timer expiration signal is delivered to or accepted by a process, if the  
 47194 implementation supports the Realtime Signals Extension, the *timer\_getoverrun()* function shall  
 47195 return the timer expiration overrun count for the specified timer. The overrun count returned  
 47196 contains the number of extra timer expirations that occurred between the time the signal was

47197 generated (queued) and when it was delivered or accepted, up to but not including an  
47198 implementation-defined maximum of {DELAYTIMER\_MAX}. If the number of such extra  
47199 expirations is greater than or equal to {DELAYTIMER\_MAX}, then the overrun count shall be set  
47200 to {DELAYTIMER\_MAX}. The value returned by *timer\_getoverrun()* shall apply to the most  
47201 recent expiration signal delivery or acceptance for the timer. If no expiration signal has been  
47202 delivered for the timer, or if the Realtime Signals Extension is not supported, the return value of  
47203 *timer\_getoverrun()* is unspecified.

#### 47204 RETURN VALUE

47205 If the *timer\_getoverrun()* function succeeds, it shall return the timer expiration overrun count as  
47206 explained above.

47207 If the *timer\_gettime()* or *timer\_settime()* functions succeed, a value of 0 shall be returned.

47208 If an error occurs for any of these functions, the value -1 shall be returned, and *errno* set to  
47209 indicate the error.

#### 47210 ERRORS

47211 The *timer\_getoverrun()*, *timer\_gettime()*, and *timer\_settime()* functions shall fail if:

47212 [EINVAL] The *timerid* argument does not correspond to an ID returned by *timer\_create()*  
47213 but not yet deleted by *timer\_delete()*.

47214 The *timer\_settime()* function shall fail if:

47215 [EINVAL] A *value* structure specified a nanosecond value less than zero or greater than  
47216 or equal to 1 000 million, and the *it\_value* member of that structure did not  
47217 specify zero seconds and nanoseconds.

#### 47218 EXAMPLES

47219 None.

#### 47220 APPLICATION USAGE

47221 None.

#### 47222 RATIONALE

47223 Practical clocks tick at a finite rate, with rates of 100 hertz and 1 000 hertz being common. The  
47224 inverse of this tick rate is the clock resolution, also called the clock granularity, which in either  
47225 case is expressed as a time duration, being 10 milliseconds and 1 millisecond respectively for  
47226 these common rates. The granularity of practical clocks implies that if one reads a given clock  
47227 twice in rapid succession, one may get the same time value twice; and that timers must wait for  
47228 the next clock tick after the theoretical expiration time, to ensure that a timer never returns too  
47229 soon. Note also that the granularity of the clock may be significantly coarser than the resolution  
47230 of the data format used to set and get time and interval values. Also note that some  
47231 implementations may choose to adjust time and/or interval values to exactly match the ticks of  
47232 the underlying clock.

47233 This volume of IEEE Std 1003.1-2001 defines functions that allow an application to determine the  
47234 implementation-supported resolution for the clocks and requires an implementation to  
47235 document the resolution supported for timers and *nanosleep()* if they differ from the supported  
47236 clock resolution. This is more of a procurement issue than a runtime application issue.

#### 47237 FUTURE DIRECTIONS

47238 None.

47239 **SEE ALSO**

47240 *clock\_getres()*, *timer\_create()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**time.h**>

47241 **CHANGE HISTORY**

47242 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

47243 **Issue 6**

47244 The *timer\_getoverrun()*, *timer\_gettime()*, and *timer\_settime()* functions are marked as part of the  
47245 Timers option.

47246 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
47247 implementation does not support the Timers option.

47248 The [EINVAL] error condition is updated to include the following: “and the *it\_value* member of  
47249 that structure did not specify zero seconds and nanoseconds.” This change is for IEEE PASC  
47250 Interpretation 1003.1 #89.

47251 The DESCRIPTION for *timer\_getoverrun()* is updated to clarify that “If no expiration signal has  
47252 been delivered for the timer, or if the Realtime Signals Extension is not supported, the return  
47253 value of *timer\_getoverrun()* is unspecified”.

47254 The **restrict** keyword is added to the *timer\_settime()* prototype for alignment with the  
47255 ISO/IEC 9899:1999 standard.

47256 **NAME**

47257 times — get process and waited-for child process times

47258 **SYNOPSIS**

47259 #include &lt;sys/times.h&gt;

47260 clock\_t times(struct tms \*buffer);

47261 **DESCRIPTION**47262 The *times()* function shall fill the **tms** structure pointed to by *buffer* with time-accounting  
47263 information. The **tms** structure is defined in <sys/times.h>.

47264 All times are measured in terms of the number of clock ticks used.

47265 The times of a terminated child process shall be included in the *tms\_cutime* and *tms\_cstime*  
47266 elements of the parent when *wait()* or *waitpid()* returns the process ID of this terminated child. If  
47267 a child process has not waited for its children, their times shall not be included in its times.47268 • The *tms\_utime* structure member is the CPU time charged for the execution of user  
47269 instructions of the calling process.47270 • The *tms\_stime* structure member is the CPU time charged for execution by the system on  
47271 behalf of the calling process.47272 • The *tms\_cutime* structure member is the sum of the *tms\_utime* and *tms\_cutime* times of the  
47273 child processes.47274 • The *tms\_cstime* structure member is the sum of the *tms\_stime* and *tms\_cstime* times of the child  
47275 processes.47276 **RETURN VALUE**47277 Upon successful completion, *times()* shall return the elapsed real time, in clock ticks, since an  
47278 arbitrary point in the past (for example, system start-up time). This point does not change from  
47279 one invocation of *times()* within the process to another. The return value may overflow the  
47280 possible range of type **clock\_t**. If *times()* fails, (**clock\_t**)-1 shall be returned and *errno* set to  
47281 indicate the error.47282 **ERRORS**

47283 No errors are defined.

47284 **EXAMPLES**47285 **Timing a Database Lookup**47286 The following example defines two functions, *start\_clock()* and *end\_clock()*, that are used to time  
47287 a lookup. It also defines variables of type **clock\_t** and **tms** to measure the duration of  
47288 transactions. The *start\_clock()* function saves the beginning times given by the *times()* function.  
47289 The *end\_clock()* function gets the ending times and prints the difference between the two times.47290 #include <sys/times.h>  
47291 #include <stdio.h>  
47292 ...  
47293 void start\_clock(void);  
47294 void end\_clock(char \*msg);  
47295 ...  
47296 static clock\_t st\_time;  
47297 static clock\_t en\_time;  
47298 static struct tms st\_cpu;  
47299 static struct tms en\_cpu;

```

47300 ...
47301 void
47302 start_clock()
47303 {
47304 st_time = times(&st_cpu);
47305 }

47306 /* This example assumes that the result of each subtraction
47307 is within the range of values that can be represented in
47308 an integer type. */
47309 void
47310 end_clock(char *msg)
47311 {
47312 en_time = times(&en_cpu);
47313
47314 fputs(msg, stdout);
47315 printf("Real Time: %jd, User Time %jd, System Time %jd\n",
47316 (intmax_t)(en_time - st_time),
47317 (intmax_t)(en_cpu.tms_utime - st_cpu.tms_utime),
47318 (intmax_t)(en_cpu.tms_stime - st_cpu.tms_stime));
47319 }

```

**47319 APPLICATION USAGE**

47320 Applications should use `sysconf(_SC_CLK_TCK)` to determine the number of clock ticks per  
47321 second as it may vary from system to system.

**47322 RATIONALE**

47323 The accuracy of the times reported is intentionally left unspecified to allow implementations  
47324 flexibility in design, from uniprocessor to multi-processor networks.

47325 The inclusion of times of child processes is recursive, so that a parent process may collect the  
47326 total times of all of its descendants. But the times of a child are only added to those of its parent  
47327 when its parent successfully waits on the child. Thus, it is not guaranteed that a parent process  
47328 can always see the total times of all its descendants; see also the discussion of the term  
47329 “realtime” in `alarm()`.

47330 If the type `clock_t` is defined to be a signed 32-bit integer, it overflows in somewhat more than a  
47331 year if there are 60 clock ticks per second, or less than a year if there are 100. There are individual  
47332 systems that run continuously for longer than that. This volume of IEEE Std 1003.1-2001 permits  
47333 an implementation to make the reference point for the returned value be the start-up time of the  
47334 process, rather than system start-up time.

47335 The term “charge” in this context has nothing to do with billing for services. The operating  
47336 system accounts for time used in this way. That information must be correct, regardless of how  
47337 that information is used.

**47338 FUTURE DIRECTIONS**

47339 None.

**47340 SEE ALSO**

47341 `alarm()`, `exec`, `fork()`, `sysconf()`, `time()`, `wait()`, the Base Definitions volume of  
47342 IEEE Std 1003.1-2001, `<sys/times.h>`

**47343 CHANGE HISTORY**

47344 First released in Issue 1. Derived from Issue 1 of the SVID.

47345 **NAME**

47346           timezone — difference from UTC and local standard time

47347 **SYNOPSIS**

47348 XSI       #include <time.h>

47349           extern long timezone;

47350

47351 **DESCRIPTION**

47352           Refer to *tzset()*.

47353 **NAME**

47354 tmpfile — create a temporary file

47355 **SYNOPSIS**

47356 #include &lt;stdio.h&gt;

47357 FILE \*tmpfile(void);

47358 **DESCRIPTION**

47359 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 47360 conflict between the requirements described here and the ISO C standard is unintentional. This  
 47361 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

47362 The *tmpfile()* function shall create a temporary file and open a corresponding stream. The file  
 47363 shall be automatically deleted when all references to the file are closed. The file is opened as in  
 47364 *fopen()* for update (*w+*).

47365 CX In some implementations, a permanent file may be left behind if the process calling *tmpfile()* is  
 47366 killed while it is processing a call to *tmpfile()*.

47367 An error message may be written to standard error if the stream cannot be opened.

47368 **RETURN VALUE**

47369 Upon successful completion, *tmpfile()* shall return a pointer to the stream of the file that is  
 47370 CX created. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

47371 **ERRORS**47372 The *tmpfile()* function shall fail if:47373 CX [EINTR] A signal was caught during *tmpfile()*.

47374 CX [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

47375 CX [ENFILE] The maximum allowable number of files is currently open in the system.

47376 CX [ENOSPC] The directory or file system which would contain the new file cannot be  
 47377 expanded.

47378 CX [EOVERFLOW] The file is a regular file and the size of the file cannot be represented correctly  
 47379 in an object of type *off\_t*.

47380 The *tmpfile()* function may fail if:

47381 CX [EMFILE] {FOPEN\_MAX} streams are currently open in the calling process.

47382 CX [ENOMEM] Insufficient storage space is available.

47383 **EXAMPLES**47384 **Creating a Temporary File**

47385 The following example creates a temporary file for update, and returns a pointer to a stream for  
 47386 the created file in the *fp* variable.

47387 #include &lt;stdio.h&gt;

47388 ...

47389 FILE \*fp;

47390 fp = tmpfile ();

47391 **APPLICATION USAGE**

47392 It should be possible to open at least {TMP\_MAX} temporary files during the lifetime of the  
47393 program (this limit may be shared with *tmpnam()*) and there should be no limit on the number  
47394 simultaneously open other than this limit and any limit on the number of open files  
47395 ({FOPEN\_MAX}).

47396 **RATIONALE**

47397 None.

47398 **FUTURE DIRECTIONS**

47399 None.

47400 **SEE ALSO**

47401 *fopen()*, *tmpnam()*, *unlink()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdio.h>

47402 **CHANGE HISTORY**

47403 First released in Issue 1. Derived from Issue 1 of the SVID.

47404 **Issue 5**

47405 Large File Summit extensions are added.

47406 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes  
47407 in previous issues.

47408 **Issue 6**

47409 Extensions beyond the ISO C standard are marked.

47410 The following new requirements on POSIX implementations derive from alignment with the  
47411 Single UNIX Specification:

- 47412 • In the ERRORS section, the [Eoverflow] condition is added. This change is to support  
47413 large files.
- 47414 • The [EMFILE] optional error condition is added.

47415 The APPLICATION USAGE section is added for alignment with the ISO/IEC 9899:1999  
47416 standard.

47417 **NAME**

47418 tmpnam — create a name for a temporary file

47419 **SYNOPSIS**

47420 #include &lt;stdio.h&gt;

47421 char \*tmpnam(char \*s);

47422 **DESCRIPTION**

47423 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 47424 conflict between the requirements described here and the ISO C standard is unintentional. This  
 47425 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

47426 The *tmpnam()* function shall generate a string that is a valid filename and that is not the same as  
 47427 the name of an existing file. The function is potentially capable of generating {TMP\_MAX}  
 47428 different strings, but any or all of them may already be in use by existing files and thus not be  
 47429 suitable return values.

47430 The *tmpnam()* function generates a different string each time it is called from the same process,  
 47431 up to {TMP\_MAX} times. If it is called more than {TMP\_MAX} times, the behavior is  
 47432 implementation-defined.

47433 The implementation shall behave as if no function defined in this volume of  
 47434 IEEE Std 1003.1-2001 calls *tmpnam()*.

47435 **CX** If the application uses any of the functions guaranteed to be available if either  
 47436 `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS` is defined, the application shall  
 47437 ensure that the *tmpnam()* function is called with a non-NULL parameter.

47438 **RETURN VALUE**

47439 Upon successful completion, *tmpnam()* shall return a pointer to a string. If no suitable string can  
 47440 be generated, the *tmpnam()* function shall return a null pointer.

47441 If the argument *s* is a null pointer, *tmpnam()* shall leave its result in an internal static object and  
 47442 return a pointer to that object. Subsequent calls to *tmpnam()* may modify the same object. If the  
 47443 argument *s* is not a null pointer, it is presumed to point to an array of at least `L_tmpnam` **chars**;  
 47444 *tmpnam()* shall write its result in that array and shall return the argument as its value.

47445 **ERRORS**

47446 No errors are defined.

47447 **EXAMPLES**47448 **Generating a Filename**47449 The following example generates a unique filename and stores it in the array pointed to by *ptr*.

47450 #include &lt;stdio.h&gt;

47451 ...

47452 char filename[L\_tmpnam+1];

47453 char \*ptr;

47454 ptr = tmpnam(filename);

47455 **APPLICATION USAGE**

47456 This function only creates filenames. It is the application's responsibility to create and remove  
 47457 the files.

47458 Between the time a pathname is created and the file is opened, it is possible for some other  
 47459 process to create a file with the same name. Applications may find *tmpfile()* more useful.

47460 **RATIONALE**

47461           None.

47462 **FUTURE DIRECTIONS**

47463           None.

47464 **SEE ALSO**47465           *fopen()*, *open()*, *tmpnam()*, *tmpfile()*, *unlink()*, the Base Definitions volume of  
47466           IEEE Std 1003.1-2001, <**stdio.h**>47467 **CHANGE HISTORY**

47468           First released in Issue 1. Derived from Issue 1 of the SVID.

47469 **Issue 5**

47470           The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

47471 **Issue 6**

47472           Extensions beyond the ISO C standard are marked.

47473           The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

47474           The DESCRIPTION is expanded for alignment with the ISO/IEC 9899:1999 standard.

47475 **NAME**

47476 toascii — translate an integer to a 7-bit ASCII character

47477 **SYNOPSIS**

47478 xSI #include &lt;ctype.h&gt;

47479 int toascii(int c);

47480

47481 **DESCRIPTION**47482 The *toascii()* function shall convert its argument into a 7-bit ASCII character.47483 **RETURN VALUE**47484 The *toascii()* function shall return the value (*c* &0x7f).47485 **ERRORS**

47486 No errors are returned.

47487 **EXAMPLES**

47488 None.

47489 **APPLICATION USAGE**

47490 None.

47491 **RATIONALE**

47492 None.

47493 **FUTURE DIRECTIONS**

47494 None.

47495 **SEE ALSO**47496 *isascii()*, the Base Definitions volume of IEEE Std 1003.1-2001, <ctype.h>47497 **CHANGE HISTORY**

47498 First released in Issue 1. Derived from Issue 1 of the SVID.

47499 **NAME**

47500           tolower — transliterate uppercase characters to lowercase

47501 **SYNOPSIS**

47502           #include <ctype.h>

47503           int tolower(int c);

47504 **DESCRIPTION**

47505 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
47506 conflict between the requirements described here and the ISO C standard is unintentional. This  
47507 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

47508       The *tolower()* function has as a domain a type **int**, the value of which is representable as an  
47509 **unsigned char** or the value of EOF. If the argument has any other value, the behavior is  
47510 undefined. If the argument of *tolower()* represents an uppercase letter, and there exists a  
47511 **CX**       corresponding lowercase letter (as defined by character type information in the program locale  
47512 category *LC\_CTYPE*), the result shall be the corresponding lowercase letter. All other arguments  
47513 in the domain are returned unchanged.

47514 **RETURN VALUE**

47515       Upon successful completion, *tolower()* shall return the lowercase letter corresponding to the  
47516 argument passed; otherwise, it shall return the argument unchanged.

47517 **ERRORS**

47518       No errors are defined.

47519 **EXAMPLES**

47520       None.

47521 **APPLICATION USAGE**

47522       None.

47523 **RATIONALE**

47524       None.

47525 **FUTURE DIRECTIONS**

47526       None.

47527 **SEE ALSO**

47528       *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <ctype.h>

47529 **CHANGE HISTORY**

47530       First released in Issue 1. Derived from Issue 1 of the SVID.

47531 **Issue 6**

47532       Extensions beyond the ISO C standard are marked.

47533 **NAME**

47534           toupper — transliterate lowercase characters to uppercase

47535 **SYNOPSIS**

47536           #include <ctype.h>

47537           int toupper(int c);

47538 **DESCRIPTION**

47539 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
47540 conflict between the requirements described here and the ISO C standard is unintentional. This  
47541 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

47542       The *toupper()* function has as a domain a type **int**, the value of which is representable as an  
47543 **unsigned char** or the value of EOF. If the argument has any other value, the behavior is  
47544 undefined. If the argument of *toupper()* represents a lowercase letter, and there exists a  
47545 **CX**       corresponding uppercase letter (as defined by character type information in the program locale  
47546 category *LC\_CTYPE*), the result shall be the corresponding uppercase letter. All other arguments  
47547 in the domain are returned unchanged.

47548 **RETURN VALUE**

47549       Upon successful completion, *toupper()* shall return the uppercase letter corresponding to the  
47550 argument passed.

47551 **ERRORS**

47552       No errors are defined.

47553 **EXAMPLES**

47554       None.

47555 **APPLICATION USAGE**

47556       None.

47557 **RATIONALE**

47558       None.

47559 **FUTURE DIRECTIONS**

47560       None.

47561 **SEE ALSO**

47562       *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <ctype.h>

47563 **CHANGE HISTORY**

47564       First released in Issue 1. Derived from Issue 1 of the SVID.

47565 **Issue 6**

47566       Extensions beyond the ISO C standard are marked.

47567 **NAME**

47568 towctrans — wide-character transliteration

47569 **SYNOPSIS**

47570 #include &lt;wctype.h&gt;

47571 wint\_t towctrans(wint\_t *wc*, wctrans\_t *desc*);47572 **DESCRIPTION**

47573 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 47574 conflict between the requirements described here and the ISO C standard is unintentional. This  
 47575 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

47576 The *towctrans()* function shall transliterate the wide-character code *wc* using the mapping  
 47577 described by *desc*. The current setting of the *LC\_CTYPE* category should be the same as during  
 47578 CX the call to *wctrans()* that returned the value *desc*. If the value of *desc* is invalid (that is, not  
 47579 obtained by a call to *wctrans()* or *desc* is invalidated by a subsequent call to *setlocale()* that has  
 47580 affected category *LC\_CTYPE*), the result is unspecified.

47581 An application wishing to check for error situations should set *errno* to 0 before calling  
 47582 *towctrans()*. If *errno* is non-zero on return, an error has occurred.

47583 **RETURN VALUE**

47584 If successful, the *towctrans()* function shall return the mapped value of *wc* using the mapping  
 47585 described by *desc*. Otherwise, it shall return *wc* unchanged.

47586 **ERRORS**47587 The *towctrans()* function may fail if:47588 CX [EINVAL] *desc* contains an invalid transliteration descriptor.47589 **EXAMPLES**

47590 None.

47591 **APPLICATION USAGE**

47592 The strings "tolower" and "toupper" are reserved for the standard mapping names. In the  
 47593 table below, the functions in the left column are equivalent to the functions in the right column.

47594 tolower(*wc*) towctrans(*wc*, wctrans("tolower"))47595 toupper(*wc*) towctrans(*wc*, wctrans("toupper"))47596 **RATIONALE**

47597 None.

47598 **FUTURE DIRECTIONS**

47599 None.

47600 **SEE ALSO**

47601 *tolower()*, *toupper()*, *wctrans()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
 47602 <wctype.h>

47603 **CHANGE HISTORY**

47604 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

47605 **Issue 6**

47606 Extensions beyond the ISO C standard are marked.

47607 **NAME**

47608 tolower — transliterate uppercase wide-character code to lowercase

47609 **SYNOPSIS**

47610 #include <wctype.h>

47611 wint\_t tolower(wint\_t wc);

47612 **DESCRIPTION**

47613 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
47614 conflict between the requirements described here and the ISO C standard is unintentional. This  
47615 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

47616 The *tolower()* function has as a domain a type **wint\_t**, the value of which the application shall  
47617 ensure is a character representable as a **wchar\_t**, and a wide-character code corresponding to a  
47618 valid character in the current locale or the value of WEOF. If the argument has any other value,  
47619 the behavior is undefined. If the argument of *tolower()* represents an uppercase wide-character  
47620 code, and there exists a corresponding lowercase wide-character code (as defined by character  
47621 type information in the program locale category *LC\_CTYPE*), the result shall be the  
47622 corresponding lowercase wide-character code. All other arguments in the domain are returned  
47623 unchanged.

47624 **RETURN VALUE**

47625 Upon successful completion, *tolower()* shall return the lowercase letter corresponding to the  
47626 argument passed; otherwise, it shall return the argument unchanged.

47627 **ERRORS**

47628 No errors are defined.

47629 **EXAMPLES**

47630 None.

47631 **APPLICATION USAGE**

47632 None.

47633 **RATIONALE**

47634 None.

47635 **FUTURE DIRECTIONS**

47636 None.

47637 **SEE ALSO**

47638 *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <**wctype.h**>,  
47639 <**wchar.h**>

47640 **CHANGE HISTORY**

47641 First released in Issue 4.

47642 **Issue 5**

47643 The following change has been made in this issue for alignment with  
47644 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 47645 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
47646 now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

47647 **Issue 6**

47648 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

47649 **NAME**

47650 towupper — transliterate lowercase wide-character code to uppercase

47651 **SYNOPSIS**

47652 #include <wctype.h>

47653 wint\_t towupper(wint\_t wc);

47654 **DESCRIPTION**

47655 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
47656 conflict between the requirements described here and the ISO C standard is unintentional. This  
47657 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

47658 The *towupper()* function has as a domain a type **wint\_t**, the value of which the application shall  
47659 ensure is a character representable as a **wchar\_t**, and a wide-character code corresponding to a  
47660 valid character in the current locale or the value of WEOF. If the argument has any other value,  
47661 the behavior is undefined. If the argument of *towupper()* represents a lowercase wide-character  
47662 code, and there exists a corresponding uppercase wide-character code (as defined by character  
47663 type information in the program locale category *LC\_CTYPE*), the result shall be the  
47664 corresponding uppercase wide-character code. All other arguments in the domain are returned  
47665 unchanged.

47666 **RETURN VALUE**

47667 Upon successful completion, *towupper()* shall return the uppercase letter corresponding to the  
47668 argument passed. Otherwise, it shall return the argument unchanged.

47669 **ERRORS**

47670 No errors are defined.

47671 **EXAMPLES**

47672 None.

47673 **APPLICATION USAGE**

47674 None.

47675 **RATIONALE**

47676 None.

47677 **FUTURE DIRECTIONS**

47678 None.

47679 **SEE ALSO**

47680 *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 7, Locale, <**wctype.h**>,  
47681 <**wchar.h**>

47682 **CHANGE HISTORY**

47683 First released in Issue 4.

47684 **Issue 5**

47685 The following change has been made in this issue for alignment with  
47686 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 47687
- 47688 • The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

47689 **Issue 6**

47690 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

47691 **NAME**

47692 trunc, truncf, trunc1 — round to truncated integer value

47693 **SYNOPSIS**

47694 #include &lt;math.h&gt;

47695 double trunc(double x);

47696 float truncf(float x);

47697 long double trunc1(long double x);

47698 **DESCRIPTION**47699 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
47700 conflict between the requirements described here and the ISO C standard is unintentional. This  
47701 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.47702 These functions shall round their argument to the integer value, in floating format, nearest to but  
47703 no larger in magnitude than the argument.47704 **RETURN VALUE**

47705 Upon successful completion, these functions shall return the truncated integer value.

47706 **MX** If  $x$  is NaN, a NaN shall be returned.47707 If  $x$  is  $\pm 0$  or  $\pm \text{Inf}$ ,  $x$  shall be returned.47708 **ERRORS**

47709 No errors are defined.

47710 **EXAMPLES**

47711 None.

47712 **APPLICATION USAGE**

47713 None.

47714 **RATIONALE**

47715 None.

47716 **FUTURE DIRECTIONS**

47717 None.

47718 **SEE ALSO**

47719 The Base Definitions volume of IEEE Std 1003.1-2001, &lt;math.h&gt;

47720 **CHANGE HISTORY**

47721 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

47722 **NAME**

47723 truncate — truncate a file to a specified length

47724 **SYNOPSIS**47725 XSI `#include <unistd.h>`47726 `int truncate(const char *path, off_t length);`

47727

47728 **DESCRIPTION**47729 The *truncate()* function shall cause the regular file named by *path* to have a size which shall be  
47730 equal to *length* bytes.47731 If the file previously was larger than *length*, the extra data is discarded. If the file was previously  
47732 shorter than *length*, its size is increased, and the extended area appears as if it were zero-filled.

47733 The application shall ensure that the process has write permission for the file.

47734 If the request would cause the file size to exceed the soft file size limit for the process, the  
47735 request shall fail and the implementation shall generate the SIGXFSZ signal for the process.47736 This function shall not modify the file offset for any open file descriptions associated with the  
47737 file. Upon successful completion, if the file size is changed, this function shall mark for update  
47738 the *st\_ctime* and *st\_mtime* fields of the file, and the S\_ISUID and S\_ISGID bits of the file mode  
47739 may be cleared.47740 **RETURN VALUE**47741 Upon successful completion, *truncate()* shall return 0. Otherwise,  $-1$  shall be returned, and *errno*  
47742 set to indicate the error.47743 **ERRORS**47744 The *truncate()* function shall fail if:

47745 [EINTR] A signal was caught during execution.

47746 [EINVAL] The *length* argument was less than 0.

47747 [EFBIG] or [EINVAL]

47748 The *length* argument was greater than the maximum file size.

47749 [EIO] An I/O error occurred while reading from or writing to a file system.

47750 [EACCES] A component of the path prefix denies search permission, or write permission  
47751 is denied on the file.

47752 [EISDIR] The named file is a directory.

47753 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
47754 argument.

47755 [ENAMETOOLONG]

47756 The length of the *path* argument exceeds {PATH\_MAX} or a pathname  
47757 component is longer than {NAME\_MAX}.47758 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.47759 [ENOTDIR] A component of the path prefix of *path* is not a directory.

47760 [EROFS] The named file resides on a read-only file system.

- 47761 The *truncate()* function may fail if:
- 47762 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
47763 resolution of the *path* argument.
- 47764 [ENAMETOOLONG]  
47765 Pathname resolution of a symbolic link produced an intermediate result  
47766 whose length exceeds {PATH\_MAX}.
- 47767 **EXAMPLES**  
47768 None.
- 47769 **APPLICATION USAGE**  
47770 None.
- 47771 **RATIONALE**  
47772 None.
- 47773 **FUTURE DIRECTIONS**  
47774 None.
- 47775 **SEE ALSO**  
47776 *open()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>
- 47777 **CHANGE HISTORY**  
47778 First released in Issue 4, Version 2.
- 47779 **Issue 5**  
47780 Moved from X/OPEN UNIX extension to BASE.  
47781 Large File Summit extensions are added.
- 47782 **Issue 6**  
47783 This reference page is split out from the *truncate()* reference page.  
47784 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.  
47785 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
47786 [ELOOP] error condition is added.

47787 **NAME**

47788           truncf, trunc1 — round to truncated integer value

47789 **SYNOPSIS**

47790           #include &lt;math.h&gt;

47791           float truncf(float x);

47792           long double trunc1(long double x);

47793 **DESCRIPTION**47794           Refer to *trunc()*.

47795 **NAME**

47796       tsearch — search a binary search tree

47797 **SYNOPSIS**

47798 XSI       #include &lt;search.h&gt;

47799       void \*tsearch(const void \*key, void \*\*rootp,  
47800                    int (\*compar)(const void \*, const void \*));

47801

47802 **DESCRIPTION**47803       Refer to *tdelete()*.

47804 **NAME**

47805 `ttyname`, `ttyname_r` — find the pathname of a terminal

47806 **SYNOPSIS**

47807 `#include <unistd.h>`

47808 `char *ttyname(int fd);`

47809 TSF `int ttyname_r(int fd, char *name, size_t namesize);`

47810

47811 **DESCRIPTION**

47812 The `ttyname()` function shall return a pointer to a string containing a null-terminated pathname  
47813 of the terminal associated with file descriptor *fd*. The return value may point to static data  
47814 whose content is overwritten by each call.

47815 The `ttyname()` function need not be reentrant. A function that is not required to be reentrant is  
47816 not required to be thread-safe.

47817 TSF The `ttyname_r()` function shall store the null-terminated pathname of the terminal associated  
47818 with the file descriptor *fd* in the character array referenced by *name*. The array is *namesize*  
47819 characters long and should have space for the name and the terminating null character. The  
47820 maximum length of the terminal name shall be {TTY\_NAME\_MAX}.

47821 **RETURN VALUE**

47822 Upon successful completion, `ttyname()` shall return a pointer to a string. Otherwise, a null  
47823 pointer shall be returned and *errno* set to indicate the error.

47824 TSF If successful, the `ttyname_r()` function shall return zero. Otherwise, an error number shall be  
47825 returned to indicate the error.

47826 **ERRORS**

47827 The `ttyname()` function may fail if:

47828 [EBADF] The *fd* argument is not a valid file descriptor.

47829 [ENOTTY] The *fd* argument does not refer to a terminal.

47830 The `ttyname_r()` function may fail if:

47831 TSF [EBADF] The *fd* argument is not a valid file descriptor.

47832 TSF [ENOTTY] The *fd* argument does not refer to a terminal.

47833 TSF [ERANGE] The value of *namesize* is smaller than the length of the string to be returned  
47834 including the terminating null character.

47835 **EXAMPLES**

47836 None.

47837 **APPLICATION USAGE**

47838 None.

47839 **RATIONALE**

47840 The term “terminal” is used instead of the historical term “terminal device” in order to avoid a  
47841 reference to an undefined term.

47842 The thread-safe version places the terminal name in a user-supplied buffer and returns a non-  
47843 zero value if it fails. The non-thread-safe version may return the name in a static data area that  
47844 may be overwritten by each call.

47845 **FUTURE DIRECTIONS**

47846 None.

47847 **SEE ALSO**47848 The Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>47849 **CHANGE HISTORY**

47850 First released in Issue 1. Derived from Issue 1 of the SVID.

47851 **Issue 5**47852 The *ttyname\_r()* function is included for alignment with the POSIX Threads Extension.47853 A note indicating that the *ttyname()* function need not be reentrant is added to the  
47854 DESCRIPTION.47855 **Issue 6**47856 The *ttyname\_r()* function is marked as part of the Thread-Safe Functions option.47857 The following new requirements on POSIX implementations derive from alignment with the  
47858 Single UNIX Specification:

- 47859
- The statement that *errno* is set on error is added.
  - The [EBADF] and [ENOTTY] optional error conditions are added.
- 47860

47861 **NAME**

47862 twalk — traverse a binary search tree

47863 **SYNOPSIS**

```
47864 XSI #include <search.h>
```

```
47865 void twalk(const void *root,
47866 void (*action)(const void *, VISIT, int));
47867
```

47868 **DESCRIPTION**

47869 Refer to *tdelete()*.

47870 **NAME**

47871 daylight, timezone, tzname, tzset — set timezone conversion information

47872 **SYNOPSIS**

47873 #include &lt;time.h&gt;

47874 XSI extern int daylight;

47875 extern long timezone;

47876 CX extern char \*tzname[2];

47877 void tzset(void);

47878

47879 **DESCRIPTION**

47880 The *tzset()* function shall use the value of the environment variable *TZ* to set time conversion information used by *ctime()*, *localtime()*, *mktime()*, and *strftime()*. If *TZ* is absent from the environment, implementation-defined default timezone information shall be used.

47883 The *tzset()* function shall set the external variable *tzname* as follows:

47884 tzname[0] = "std";

47885 tzname[1] = "dst";

47886 where *std* and *dst* are as described in the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables.

47888 XSI The *tzset()* function also shall set the external variable *daylight* to 0 if Daylight Savings Time conversions should never be applied for the timezone in use; otherwise, non-zero. The external variable *timezone* shall be set to the difference, in seconds, between Coordinated Universal Time (UTC) and local standard time.

47892 **RETURN VALUE**47893 The *tzset()* function shall not return a value.47894 **ERRORS**

47895 No errors are defined.

47896 **EXAMPLES**47897 Example *TZ* variables and their timezone differences are given in the table below:

47898

|       | <i>TZ</i> | <i>timezone</i> |
|-------|-----------|-----------------|
| 47900 | EST5EDT   | 5*60*60         |
| 47901 | GMT0      | 0*60*60         |
| 47902 | JST-9     | -9*60*60        |
| 47903 | MET-1MEST | -1*60*60        |
| 47904 | MST7MDT   | 7*60*60         |
| 47905 | PST8PDT   | 8*60*60         |

47906 **APPLICATION USAGE**

47907 None.

47908 **RATIONALE**

47909 None.

47910 **FUTURE DIRECTIONS**

47911 None.

47912 **SEE ALSO**

47913            *ctime()*, *localtime()*, *mktime()*, *strftime()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
47914            Chapter 8, Environment Variables, <**time.h**>

47915 **CHANGE HISTORY**

47916            First released in Issue 1. Derived from Issue 1 of the SVID.

47917 **Issue 6**

47918            The example is corrected.

47919 **NAME**47920        **ualarm** — set the interval timer47921 **SYNOPSIS**

47922 OB XSI    #include &lt;unistd.h&gt;

47923        useconds\_t ualarm(useconds\_t useconds, useconds\_t interval);

47924

47925 **DESCRIPTION**

47926        The *ualarm()* function shall cause the SIGALRM signal to be generated for the calling process  
 47927        after the number of realtime microseconds specified by the *useconds* argument has elapsed.  
 47928        When the *interval* argument is non-zero, repeated timeout notification occurs with a period in  
 47929        microseconds specified by the *interval* argument. If the notification signal, SIGALRM, is not  
 47930        caught or ignored, the calling process is terminated.

47931        Implementations may place limitations on the granularity of timer values. For each interval  
 47932        timer, if the requested timer value requires a finer granularity than the implementation supports,  
 47933        the actual timer value shall be rounded up to the next supported value.

47934        Interactions between *ualarm()* and any of the following are unspecified:

47935            *alarm()*  
 47936            *nanosleep()*  
 47937            *setitimer()*  
 47938            *timer\_create()*  
 47939            *timer\_delete()*  
 47940            *timer\_getoverrun()*  
 47941            *timer\_gettime()*  
 47942            *timer\_settime()*  
 47943            *sleep()*

47944 **RETURN VALUE**

47945        The *ualarm()* function shall return the number of microseconds remaining from the previous  
 47946        *ualarm()* call. If no timeouts are pending or if *ualarm()* has not previously been called, *ualarm()*  
 47947        shall return 0.

47948 **ERRORS**

47949        No errors are defined.

47950 **EXAMPLES**

47951        None.

47952 **APPLICATION USAGE**

47953        Applications are recommended to use *nanosleep()* if the Timers option is supported, or  
 47954        *setitimer()*, *timer\_create()*, *timer\_delete()*, *timer\_getoverrun()*, *timer\_gettime()*, or *timer\_settime()*  
 47955        instead of this function.

47956 **RATIONALE**

47957        None.

47958 **FUTURE DIRECTIONS**

47959        None.

47960 **SEE ALSO**

47961        *alarm()*, *nanosleep()*, *setitimer()*, *sleep()*, *timer\_create()*, *timer\_delete()*, *timer\_getoverrun()*, the Base  
 47962        Definitions volume of IEEE Std 1003.1-2001, <unistd.h>

47963 **CHANGE HISTORY**

47964 First released in Issue 4, Version 2.

47965 **Issue 5**

47966 Moved from X/OPEN UNIX extension to BASE.

47967 **Issue 6**

47968 This function is marked obsolescent.

47969 **NAME**47970 `ulimit` — get and set process limits47971 **SYNOPSIS**47972 XSI `#include <ulimit.h>`47973 `long ulimit(int cmd, ...);`

47974

47975 **DESCRIPTION**

47976 The `ulimit()` function shall control process limits. The process limits that can be controlled by  
 47977 this function include the maximum size of a single file that can be written (this is equivalent to  
 47978 using `setrlimit()` with `RLIMIT_FSIZE`). The `cmd` values, defined in `<ulimit.h>`, include:

47979 **UL\_GETFSIZE** Return the file size limit (`RLIMIT_FSIZE`) of the process. The limit shall be in  
 47980 units of 512-byte blocks and shall be inherited by child processes. Files of any  
 47981 size can be read. The return value shall be the integer part of the soft file size  
 47982 limit divided by 512. If the result cannot be represented as a **long**, the result is  
 47983 unspecified.

47984 **UL\_SETFSIZE** Set the file size limit for output operations of the process to the value of the  
 47985 second argument, taken as a **long**, multiplied by 512. If the result would  
 47986 overflow an **rlim\_t**, the actual value set is unspecified. Any process may  
 47987 decrease its own limit, but only a process with appropriate privileges may  
 47988 increase the limit. The return value shall be the integer part of the new file size  
 47989 limit divided by 512.

47990 The `ulimit()` function shall not change the setting of `errno` if successful.

47991 As all return values are permissible in a successful situation, an application wishing to check for  
 47992 error situations should set `errno` to 0, then call `ulimit()`, and, if it returns `-1`, check to see if `errno` is  
 47993 non-zero.

47994 **RETURN VALUE**

47995 Upon successful completion, `ulimit()` shall return the value of the requested limit. Otherwise, `-1`  
 47996 shall be returned and `errno` set to indicate the error.

47997 **ERRORS**

47998 The `ulimit()` function shall fail and the limit shall be unchanged if:

47999 [EINVAL] The `cmd` argument is not valid.

48000 [EPERM] A process not having appropriate privileges attempts to increase its file size  
 48001 limit.

48002 **EXAMPLES**

48003 None.

48004 **APPLICATION USAGE**

48005 None.

48006 **RATIONALE**

48007 None.

48008 **FUTURE DIRECTIONS**

48009 None.

48010 **SEE ALSO**

48011 *getrlimit()*, *setrlimit()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**ulimit.h**>

48012 **CHANGE HISTORY**

48013 First released in Issue 1. Derived from Issue 1 of the SVID.

48014 **Issue 5**

48015 In the description of UL\_SETFSIZE, the text is corrected to refer to **rlim\_t** rather than the spurious **rlimit\_t**.

48017 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

48018 **NAME**

48019           umask — set and get the file mode creation mask

48020 **SYNOPSIS**

48021           #include <sys/stat.h>

48022           mode\_t umask(mode\_t *cmask*);

48023 **DESCRIPTION**

48024           The *umask()* function shall set the process' file mode creation mask to *cmask* and return the  
48025           previous value of the mask. Only the file permission bits of *cmask* (see <sys/stat.h>) are used; the  
48026           meaning of the other bits is implementation-defined.

48027           The process' file mode creation mask is used during *open()*, *creat()*, *mkdir()*, and *mkfifo()* to turn  
48028           off permission bits in the *mode* argument supplied. Bit positions that are set in *cmask* are cleared  
48029           in the mode of the created file.

48030 **RETURN VALUE**

48031           The file permission bits in the value returned by *umask()* shall be the previous value of the file  
48032           mode creation mask. The state of any other bits in that value is unspecified, except that a  
48033           subsequent call to *umask()* with the returned value as *cmask* shall leave the state of the mask the  
48034           same as its state before the first call, including any unspecified use of those bits.

48035 **ERRORS**

48036           No errors are defined.

48037 **EXAMPLES**

48038           None.

48039 **APPLICATION USAGE**

48040           None.

48041 **RATIONALE**

48042           Unsigned argument and return types for *umask()* were proposed. The return type and the  
48043           argument were both changed to **mode\_t**.

48044           Historical implementations have made use of additional bits in *cmask* for their implementation-  
48045           defined purposes. The addition of the text that the meaning of other bits of the field is  
48046           implementation-defined permits these implementations to conform to this volume of  
48047           IEEE Std 1003.1-2001.

48048 **FUTURE DIRECTIONS**

48049           None.

48050 **SEE ALSO**

48051           *creat()*, *mkdir()*, *mkfifo()*, *open()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
48052           <sys/stat.h>, <sys/types.h>

48053 **CHANGE HISTORY**

48054           First released in Issue 1. Derived from Issue 1 of the SVID.

48055 **Issue 6**

48056           In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

48057           The following new requirements on POSIX implementations derive from alignment with the  
48058           Single UNIX Specification:

- 48059           • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
48060           required for conforming implementations of previous POSIX specifications, it was not  
48061           required for UNIX applications.

48062 **NAME**

48063        **uname** — get the name of the current system

48064 **SYNOPSIS**

48065        #include <sys/utsname.h>

48066        int uname(struct utsname \*name);

48067 **DESCRIPTION**

48068        The *uname()* function shall store information identifying the current system in the structure pointed to by *name*.

48070        The *uname()* function uses the **utsname** structure defined in <sys/utsname.h>.

48071        The *uname()* function shall return a string naming the current system in the character array *sysname*. Similarly, *nodename* shall contain the name of this node within an implementation-defined communications network. The arrays *release* and *version* shall further identify the operating system. The array *machine* shall contain a name that identifies the hardware that the system is running on.

48076        The format of each member is implementation-defined.

48077 **RETURN VALUE**

48078        Upon successful completion, a non-negative value shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

48080 **ERRORS**

48081        No errors are defined.

48082 **EXAMPLES**

48083        None.

48084 **APPLICATION USAGE**

48085        The inclusion of the *nodename* member in this structure does not imply that it is sufficient information for interfacing to communications networks.

48087 **RATIONALE**

48088        The values of the structure members are not constrained to have any relation to the version of this volume of IEEE Std 1003.1-2001 implemented in the operating system. An application should instead depend on `_POSIX_VERSION` and related constants defined in <unistd.h>.

48091        This volume of IEEE Std 1003.1-2001 does not define the sizes of the members of the structure and permits them to be of different sizes, although most implementations define them all to be the same size: eight bytes plus one byte for the string terminator. That size for *nodename* is not enough for use with many networks.

48095        The *uname()* function originated in System III, System V, and related implementations, and it does not exist in Version 7 or 4.3 BSD. The values it returns are set at system compile time in those historical implementations.

48098        4.3 BSD has *gethostname()* and *gethostid()*, which return a symbolic name and a numeric value, respectively. There are related *sethostname()* and *sethostid()* functions that are used to set the values the other two functions return. The former functions are included in this specification, the latter are not.

48102 **FUTURE DIRECTIONS**

48103        None.

48104 **SEE ALSO**

48105           The Base Definitions volume of IEEE Std 1003.1-2001, <sys/utsname.h>

48106 **CHANGE HISTORY**

48107           First released in Issue 1. Derived from Issue 1 of the SVID.

48108 **NAME**

48109 ungetc — push byte back into input stream

48110 **SYNOPSIS**

48111 #include &lt;stdio.h&gt;

48112 int ungetc(int *c*, FILE \**stream*);48113 **DESCRIPTION**

48114 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
48115 conflict between the requirements described here and the ISO C standard is unintentional. This  
48116 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

48117 The *ungetc()* function shall push the byte specified by *c* (converted to an **unsigned char**) back  
48118 onto the input stream pointed to by *stream*. The pushed-back bytes shall be returned by  
48119 subsequent reads on that stream in the reverse order of their pushing. A successful intervening  
48120 call (with the stream pointed to by *stream*) to a file-positioning function (*fseek()*, *fsetpos()*, or  
48121 *rewind()*) shall discard any pushed-back bytes for the stream. The external storage  
48122 corresponding to the stream shall be unchanged.

48123 One byte of push-back shall be provided. If *ungetc()* is called too many times on the same stream  
48124 without an intervening read or file-positioning operation on that stream, the operation may fail.

48125 If the value of *c* equals that of the macro EOF, the operation shall fail and the input stream shall  
48126 be left unchanged.

48127 A successful call to *ungetc()* shall clear the end-of-file indicator for the stream. The value of the  
48128 file-position indicator for the stream after reading or discarding all pushed-back bytes shall be  
48129 the same as it was before the bytes were pushed back. The file-position indicator is decremented  
48130 by each successful call to *ungetc()*; if its value was 0 before a call, its value is unspecified after  
48131 the call.

48132 **RETURN VALUE**

48133 Upon successful completion, *ungetc()* shall return the byte pushed back after conversion.  
48134 Otherwise, it shall return EOF.

48135 **ERRORS**

48136 No errors are defined.

48137 **EXAMPLES**

48138 None.

48139 **APPLICATION USAGE**

48140 None.

48141 **RATIONALE**

48142 None.

48143 **FUTURE DIRECTIONS**

48144 None.

48145 **SEE ALSO**

48146 *fseek()*, *getc()*, *fsetpos()*, *read()*, *rewind()*, *setbuf()*, the Base Definitions volume of  
48147 IEEE Std 1003.1-2001, <stdio.h>

48148 **CHANGE HISTORY**

48149 First released in Issue 1. Derived from Issue 1 of the SVID.

48150 **NAME**

48151 ungetwc — push wide-character code back into the input stream

48152 **SYNOPSIS**

48153 #include &lt;stdio.h&gt;

48154 #include &lt;wchar.h&gt;

48155 wint\_t ungetwc(wint\_t *wc*, FILE \**stream*);48156 **DESCRIPTION**

48157 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 48158 conflict between the requirements described here and the ISO C standard is unintentional. This  
 48159 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

48160 The *ungetwc()* function shall push the character corresponding to the wide-character code  
 48161 specified by *wc* back onto the input stream pointed to by *stream*. The pushed-back characters  
 48162 shall be returned by subsequent reads on that stream in the reverse order of their pushing. A  
 48163 successful intervening call (with the stream pointed to by *stream*) to a file-positioning function  
 48164 (*fseek()*, *fsetpos()*, or *rewind()*) discards any pushed-back characters for the stream. The external  
 48165 storage corresponding to the stream is unchanged.

48166 At least one character of push-back shall be provided. If *ungetwc()* is called too many times on  
 48167 the same stream without an intervening read or file-positioning operation on that stream, the  
 48168 operation may fail.

48169 If the value of *wc* equals that of the macro WEOF, the operation shall fail and the input stream  
 48170 shall be left unchanged.

48171 A successful call to *ungetwc()* shall clear the end-of-file indicator for the stream. The value of the  
 48172 file-position indicator for the stream after reading or discarding all pushed-back characters shall  
 48173 be the same as it was before the characters were pushed back. The file-position indicator is  
 48174 decremented (by one or more) by each successful call to *ungetwc()*; if its value was 0 before a  
 48175 call, its value is unspecified after the call.

48176 **RETURN VALUE**

48177 Upon successful completion, *ungetwc()* shall return the wide-character code corresponding to  
 48178 the pushed-back character. Otherwise, it shall return WEOF.

48179 **ERRORS**48180 The *ungetwc()* function may fail if:

48181 **CX** [EILSEQ] An invalid character sequence is detected, or a wide-character code does not  
 48182 correspond to a valid character.

48183 **EXAMPLES**

48184 None.

48185 **APPLICATION USAGE**

48186 None.

48187 **RATIONALE**

48188 None.

48189 **FUTURE DIRECTIONS**

48190 None.

48191 **SEE ALSO**

48192 *fseek()*, *fsetpos()*, *read()*, *rewind()*, *setbuf()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
48193 **<stdio.h>**, **<wchar.h>**

48194 **CHANGE HISTORY**

48195 First released in Issue 4. Derived from the MSE working draft.

48196 **Issue 5**

48197 The Optional Header (OH) marking is removed from **<stdio.h>**.

48198 **Issue 6**

48199 The [EILSEQ] optional error condition is marked CX.

48200 **NAME**

48201 unlink — remove a directory entry

48202 **SYNOPSIS**

48203 #include <unistd.h>

48204 int unlink(const char \*path);

48205 **DESCRIPTION**

48206 The *unlink()* function shall remove a link to a file. If *path* names a symbolic link, *unlink()* shall  
 48207 remove the symbolic link named by *path* and shall not affect any file or directory named by the  
 48208 contents of the symbolic link. Otherwise, *unlink()* shall remove the link named by the pathname  
 48209 pointed to by *path* and shall decrement the link count of the file referenced by the link.

48210 When the file's link count becomes 0 and no process has the file open, the space occupied by the  
 48211 file shall be freed and the file shall no longer be accessible. If one or more processes have the file  
 48212 open when the last link is removed, the link shall be removed before *unlink()* returns, but the  
 48213 removal of the file contents shall be postponed until all references to the file are closed.

48214 The *path* argument shall not name a directory unless the process has appropriate privileges and  
 48215 the implementation supports using *unlink()* on directories.

48216 Upon successful completion, *unlink()* shall mark for update the *st\_ctime* and *st\_mtime* fields of  
 48217 the parent directory. Also, if the file's link count is not 0, the *st\_ctime* field of the file shall be  
 48218 marked for update.

48219 **RETURN VALUE**

48220 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 48221 indicate the error. If -1 is returned, the named file shall not be changed.

48222 **ERRORS**

48223 The *unlink()* function shall fail and shall not unlink the file if:

48224 [EACCES] Search permission is denied for a component of the path prefix, or write  
 48225 permission is denied on the directory containing the directory entry to be  
 48226 removed.

48227 [EBUSY] The file named by the *path* argument cannot be unlinked because it is being  
 48228 used by the system or another process and the implementation considers this  
 48229 an error.

48230 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 48231 argument.

48232 [ENAMETOOLONG] The length of the *path* argument exceeds {PATH\_MAX} or a pathname  
 48233 component is longer than {NAME\_MAX}.  
 48234

48235 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

48236 [ENOTDIR] A component of the path prefix is not a directory.

48237 [EPERM] The file named by *path* is a directory, and either the calling process does not  
 48238 have appropriate privileges, or the implementation prohibits using *unlink()*  
 48239 on directories.

48240 XSI [EPERM] or [EACCES]  
 48241 The S\_ISVTX flag is set on the directory containing the file referred to by the  
 48242 *path* argument and the caller is not the file owner, nor is the caller the  
 48243 directory owner, nor does the caller have appropriate privileges.

- 48244 [EROFS] The directory entry to be unlinked is part of a read-only file system.
- 48245 The *unlink()* function may fail and not unlink the file if:
- 48246 XSI [EBUSY] The file named by *path* is a named STREAM.
- 48247 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
48248 resolution of the *path* argument.
- 48249 [ENAMETOOLONG]  
48250 As a result of encountering a symbolic link in resolution of the *path* argument,  
48251 the length of the substituted pathname string exceeded {PATH\_MAX}.
- 48252 [ETXTBSY] The entry to be unlinked is the last directory entry to a pure procedure (shared  
48253 text) file that is being executed.

48254 **EXAMPLES**48255 **Removing a Link to a File**

48256 The following example shows how to remove a link to a file named `/home/cnd/mod1` by  
48257 removing the entry named `/modules/pass1`.

```
48258 #include <unistd.h>
48259 char *path = "/modules/pass1";
48260 int status;
48261 ...
48262 status = unlink(path);
```

48263 **Checking for an Error**

48264 The following example fragment creates a temporary password lock file named **LOCKFILE**,  
48265 which is defined as `/etc/ptmp`, and gets a file descriptor for it. If the file cannot be opened for  
48266 writing, *unlink()* is used to remove the link between the file descriptor and **LOCKFILE**.

```
48267 #include <sys/types.h>
48268 #include <stdio.h>
48269 #include <fcntl.h>
48270 #include <errno.h>
48271 #include <unistd.h>
48272 #include <sys/stat.h>
48273 #define LOCKFILE "/etc/ptmp"
48274 int pfd; /* Integer for file descriptor returned by open call. */
48275 FILE *fpfd; /* File pointer for use in putpwent(). */
48276 ...
48277 /* Open password Lock file. If it exists, this is an error. */
48278 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL, S_IRUSR
48279 | S_IWUSR | S_IRGRP | S_IROTH)) == -1) {
48280 fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
48281 exit(1);
48282 }
48283 /* Lock file created; proceed with fdopen of lock file so that
48284 putpwent() can be used.
48285 */
48286 if ((fpfd = fdopen(pfd, "w")) == NULL) {
```

```

48287 close(pfd);
48288 unlink(LOCKFILE);
48289 exit(1);
48290 }

```

### 48291 **Replacing Files**

48292 The following example fragment uses *unlink()* to discard links to files, so that they can be  
48293 replaced with new versions of the files. The first call removes the link to **LOCKFILE** if an error  
48294 occurs. Successive calls remove the links to **SAVEFILE** and **PASSWDFILE** so that new links can  
48295 be created, then removes the link to **LOCKFILE** when it is no longer needed.

```

48296 #include <sys/types.h>
48297 #include <stdio.h>
48298 #include <fcntl.h>
48299 #include <errno.h>
48300 #include <unistd.h>
48301 #include <sys/stat.h>
48302
48303 #define LOCKFILE "/etc/ptmp"
48304 #define PASSWDFILE "/etc/passwd"
48305 #define SAVEFILE "/etc/opasswd"
48306 ...
48307 /* If no change was made, assume error and leave passwd unchanged. */
48308 if (!valid_change) {
48309 fprintf(stderr, "Could not change password for user %s\n", user);
48310 unlink(LOCKFILE);
48311 exit(1);
48312 }
48313
48314 /* Change permissions on new password file. */
48315 chmod(LOCKFILE, S_IRUSR | S_IRGRP | S_IROTH);
48316
48317 /* Remove saved password file. */
48318 unlink(SAVEFILE);
48319
48320 /* Save current password file. */
48321 link(PASSWDFILE, SAVEFILE);
48322
48323 /* Remove current password file. */
48324 unlink(PASSWDFILE);
48325
48326 /* Save new password file as current password file. */
48327 link(LOCKFILE, PASSWDFILE);
48328
48329 /* Remove lock file. */
48330 unlink(LOCKFILE);
48331
48332 exit(0);

```

### 48325 **APPLICATION USAGE**

48326 Applications should use *rmdir()* to remove a directory.

### 48327 **RATIONALE**

48328 Unlinking a directory is restricted to the superuser in many historical implementations for  
48329 reasons given in *link()* (see also *rename()*).

48330 The meaning of [EBUSY] in historical implementations is “mount point busy”. Since this volume  
48331 of IEEE Std 1003.1-2001 does not cover the system administration concepts of mounting and  
48332 unmounting, the description of the error was changed to “resource busy”. (This meaning is used  
48333 by some device drivers when a second process tries to open an exclusive use device.) The  
48334 wording is also intended to allow implementations to refuse to remove a directory if it is the  
48335 root or current working directory of any process.

48336 **FUTURE DIRECTIONS**

48337 None.

48338 **SEE ALSO**

48339 *close()*, *link()*, *remove()*, *rmdir()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

48340 **CHANGE HISTORY**

48341 First released in Issue 1. Derived from Issue 1 of the SVID.

48342 **Issue 5**

48343 The [EBUSY] error is added to the “may fail” part of the ERRORS section.

48344 **Issue 6**

48345 The following new requirements on POSIX implementations derive from alignment with the  
48346 Single UNIX Specification:

- 48347 • In the DESCRIPTION, the effect is specified if *path* specifies a symbolic link.
- 48348 • The [ELOOP] mandatory error condition is added.
- 48349 • A second [ENAMETOOLONG] is added as an optional error condition.
- 48350 • The [ETXTBSY] optional error condition is added.

48351 The following changes were made to align with the IEEE P1003.1a draft standard:

- 48352 • The [ELOOP] optional error condition is added.

48353 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

48354 **NAME**

48355 unlockpt — unlock a pseudo-terminal master/slave pair

48356 **SYNOPSIS**48357 XSI `#include <stdlib.h>`48358 `int unlockpt(int fildev);`

48359

48360 **DESCRIPTION**48361 The *unlockpt()* function shall unlock the slave pseudo-terminal device associated with the  
48362 master to which *fildev* refers.48363 Conforming applications shall ensure that they call *unlockpt()* before opening the slave side of a  
48364 pseudo-terminal device.48365 **RETURN VALUE**48366 Upon successful completion, *unlockpt()* shall return 0. Otherwise, it shall return -1 and set *errno*  
48367 to indicate the error.48368 **ERRORS**48369 The *unlockpt()* function may fail if:48370 [EBADF] The *fildev* argument is not a file descriptor open for writing.48371 [EINVAL] The *fildev* argument is not associated with a master pseudo-terminal device.48372 **EXAMPLES**

48373 None.

48374 **APPLICATION USAGE**

48375 None.

48376 **RATIONALE**

48377 None.

48378 **FUTURE DIRECTIONS**

48379 None.

48380 **SEE ALSO**48381 *grantpt()*, *open()*, *ptsname()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<stdlib.h>`48382 **CHANGE HISTORY**

48383 First released in Issue 4, Version 2.

48384 **Issue 5**

48385 Moved from X/OPEN UNIX extension to BASE.

48386 **Issue 6**

48387 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

48388 **NAME**

48389           unsetenv — remove an environment variable

48390 **SYNOPSIS**

48391 cx       #include &lt;stdlib.h&gt;

48392           int unsetenv(const char \*name);

48393

48394 **DESCRIPTION**

48395       The *unsetenv()* function shall remove an environment variable from the environment of the  
48396       calling process. The *name* argument points to a string, which is the name of the variable to be  
48397       removed. The named argument shall not contain an '=' character. If the named variable does  
48398       not exist in the current environment, the environment shall be unchanged and the function is  
48399       considered to have completed successfully.

48400       If the application modifies *environ* or the pointers to which it points, the behavior of *unsetenv()* is  
48401       undefined. The *unsetenv()* function shall update the list of pointers to which *environ* points.

48402       The *unsetenv()* function need not be reentrant. A function that is not required to be reentrant is  
48403       not required to be thread-safe.

48404 **RETURN VALUE**

48405       Upon successful completion, zero shall be returned. Otherwise, -1 shall be returned, *errno* set to  
48406       indicate the error, and the environment shall be unchanged.

48407 **ERRORS**48408       The *unsetenv()* function shall fail if:

48409       [EINVAL]       The *name* argument is a null pointer, points to an empty string, or points to a  
48410       string containing an '=' character.

48411 **EXAMPLES**

48412       None.

48413 **APPLICATION USAGE**

48414       None.

48415 **RATIONALE**48416       Refer to the RATIONALE section in *setenv()*.48417 **FUTURE DIRECTIONS**

48418       None.

48419 **SEE ALSO**

48420       *getenv()*, *setenv()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdlib.h>,  
48421       <sys/types.h>, <unistd.h>

48422 **CHANGE HISTORY**

48423       First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

48424 **NAME**

48425            usleep — suspend execution for an interval

48426 **SYNOPSIS**

48427 OB XSI    #include &lt;unistd.h&gt;

48428            int usleep(useconds\_t useconds);

48429

48430 **DESCRIPTION**

48431            The *usleep()* function shall cause the calling thread to be suspended from execution until either  
 48432            the number of realtime microseconds specified by the argument *useconds* has elapsed or a signal  
 48433            is delivered to the calling thread and its action is to invoke a signal-catching function or to  
 48434            terminate the process. The suspension time may be longer than requested due to the scheduling  
 48435            of other activity by the system.

48436            The *useconds* argument shall be less than one million. If the value of *useconds* is 0, then the call  
 48437            has no effect.

48438            If a SIGALRM signal is generated for the calling process during execution of *usleep()* and if the  
 48439            SIGALRM signal is being ignored or blocked from delivery, it is unspecified whether *usleep()*  
 48440            returns when the SIGALRM signal is scheduled. If the signal is being blocked, it is also  
 48441            unspecified whether it remains pending after *usleep()* returns or it is discarded.

48442            If a SIGALRM signal is generated for the calling process during execution of *usleep()*, except as a  
 48443            result of a prior call to *alarm()*, and if the SIGALRM signal is not being ignored or blocked from  
 48444            delivery, it is unspecified whether that signal has any effect other than causing *usleep()* to return.

48445            If a signal-catching function interrupts *usleep()* and examines or changes either the time a  
 48446            SIGALRM is scheduled to be generated, the action associated with the SIGALRM signal, or  
 48447            whether the SIGALRM signal is blocked from delivery, the results are unspecified.

48448            If a signal-catching function interrupts *usleep()* and calls *siglongjmp()* or *longjmp()* to restore an  
 48449            environment saved prior to the *usleep()* call, the action associated with the SIGALRM signal and  
 48450            the time at which a SIGALRM signal is scheduled to be generated are unspecified. It is also  
 48451            unspecified whether the SIGALRM signal is blocked, unless the process' signal mask is restored  
 48452            as part of the environment.

48453            Implementations may place limitations on the granularity of timer values. For each interval  
 48454            timer, if the requested timer value requires a finer granularity than the implementation supports,  
 48455            the actual timer value shall be rounded up to the next supported value.

48456            Interactions between *usleep()* and any of the following are unspecified:

48457            *nanosleep()*48458            *setitimer()*48459            *timer\_create()*48460            *timer\_delete()*48461            *timer\_getoverrun()*48462            *timer\_gettime()*48463            *timer\_settime()*48464            *ualarm()*48465            *sleep()*

48466 **RETURN VALUE**

48467       Upon successful completion, *usleep()* shall return 0; otherwise, it shall return -1 and set *errno* to  
48468       indicate the error.

48469 **ERRORS**

48470       The *usleep()* function may fail if:

48471       [EINVAL]       The time interval specified one million or more microseconds.

48472 **EXAMPLES**

48473       None.

48474 **APPLICATION USAGE**

48475       Applications are recommended to use *nanosleep()* if the Timers option is supported, or  
48476       *setitimer()*, *timer\_create()*, *timer\_delete()*, *timer\_getoverrun()*, *timer\_gettime()*, or *timer\_settime()*  
48477       instead of this function.

48478 **RATIONALE**

48479       None.

48480 **FUTURE DIRECTIONS**

48481       None.

48482 **SEE ALSO**

48483       *alarm()*, *getitimer()*, *nanosleep()*, *sigaction()*, *sleep()*, *timer\_create()*, *timer\_delete()*,  
48484       *timer\_getoverrun()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>

48485 **CHANGE HISTORY**

48486       First released in Issue 4, Version 2.

48487 **Issue 5**

48488       Moved from X/OPEN UNIX extension to BASE.

48489       The DESCRIPTION is changed to indicate that timers are now thread-based rather than  
48490       process-based.

48491 **Issue 6**

48492       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

48493       This function is marked obsolescent.

## 48494 NAME

48495 utime — set file access and modification times

## 48496 SYNOPSIS

48497 #include &lt;utime.h&gt;

48498 int utime(const char \*path, const struct utimbuf \*times);

## 48499 DESCRIPTION

48500 The *utime()* function shall set the access and modification times of the file named by the *path*  
48501 argument.48502 If *times* is a null pointer, the access and modification times of the file shall be set to the current  
48503 time. The effective user ID of the process shall match the owner of the file, or the process has  
48504 write permission to the file or has appropriate privileges, to use *utime()* in this manner.48505 If *times* is not a null pointer, *times* shall be interpreted as a pointer to a **utimbuf** structure and the  
48506 access and modification times shall be set to the values contained in the designated structure.  
48507 Only a process with the effective user ID equal to the user ID of the file or a process with  
48508 appropriate privileges may use *utime()* this way.48509 The **utimbuf** structure is defined in the <**utime.h**> header. The times in the structure **utimbuf**  
48510 are measured in seconds since the Epoch.48511 Upon successful completion, *utime()* shall mark the time of the last file status change, *st\_ctime*,  
48512 to be updated; see <**sys/stat.h**>.

## 48513 RETURN VALUE

48514 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* shall  
48515 be set to indicate the error, and the file times shall not be affected.

## 48516 ERRORS

48517 The *utime()* function shall fail if:48518 [EACCES] Search permission is denied by a component of the path prefix; or the *times*  
48519 argument is a null pointer and the effective user ID of the process does not  
48520 match the owner of the file, the process does not have write permission for the  
48521 file, and the process does not have appropriate privileges.48522 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
48523 argument.48524 [ENAMETOOLONG]  
48525 The length of the *path* argument exceeds {PATH\_MAX} or a pathname  
48526 component is longer than {NAME\_MAX}.48527 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

48528 [ENOTDIR] A component of the path prefix is not a directory.

48529 [EPERM] The *times* argument is not a null pointer and the calling process' effective user  
48530 ID does not match the owner of the file and the calling process does not have  
48531 the appropriate privileges.

48532 [EROFS] The file system containing the file is read-only.

48533 The *utime()* function may fail if:48534 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
48535 resolution of the *path* argument.

48536 [ENAMETOOLONG]

48537 As a result of encountering a symbolic link in resolution of the *path* argument,  
48538 the length of the substituted pathname string exceeded {PATH\_MAX}.

48539 **EXAMPLES**

48540 None.

48541 **APPLICATION USAGE**

48542 None.

48543 **RATIONALE**

48544 The *actime* structure member must be present so that an application may set it, even though an  
48545 implementation may ignore it and not change the access time on the file. If an application  
48546 intends to leave one of the times of a file unchanged while changing the other, it should use  
48547 *stat()* to retrieve the file's *st\_atime* and *st\_mtime* parameters, set *actime* and *modtime* in the buffer,  
48548 and change one of them before making the *utime()* call.

48549 **FUTURE DIRECTIONS**

48550 None.

48551 **SEE ALSO**

48552 The Base Definitions volume of IEEE Std 1003.1-2001, <sys/stat.h>, <utime.h>

48553 **CHANGE HISTORY**

48554 First released in Issue 1. Derived from Issue 1 of the SVID.

48555 **Issue 6**

48556 The following new requirements on POSIX implementations derive from alignment with the  
48557 Single UNIX Specification:

- 48558 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
48559 required for conforming implementations of previous POSIX specifications, it was not  
48560 required for UNIX applications.
- 48561 • The [ELOOP] mandatory error condition is added.
- 48562 • A second [ENAMETOOLONG] is added as an optional error condition.

48563 The following changes were made to align with the IEEE P1003.1a draft standard:

- 48564 • The [ELOOP] optional error condition is added.

48565 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

## 48566 NAME

48567 utimes — set file access and modification times (**LEGACY**)

## 48568 SYNOPSIS

48569 XSI `#include <sys/time.h>`48570 `int utimes(const char *path, const struct timeval times[2]);`

48571

## 48572 DESCRIPTION

48573 The *utimes()* function shall set the access and modification times of the file pointed to by the *path*  
 48574 argument to the value of the *times* argument. The *utimes()* function allows time specifications  
 48575 accurate to the microsecond.

48576 For *utimes()*, the *times* argument is an array of **timeval** structures. The first array member  
 48577 represents the date and time of last access, and the second member represents the date and time  
 48578 of last modification. The times in the **timeval** structure are measured in seconds and  
 48579 microseconds since the Epoch, although rounding toward the nearest second may occur.

48580 If the *times* argument is a null pointer, the access and modification times of the file shall be set to  
 48581 the current time. The effective user ID of the process shall match the owner of the file, or has  
 48582 write access to the file or appropriate privileges to use this call in this manner. Upon completion,  
 48583 *utimes()* shall mark the time of the last file status change, *st\_ctime*, for update.

## 48584 RETURN VALUE

48585 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* shall  
 48586 be set to indicate the error, and the file times shall not be affected.

## 48587 ERRORS

48588 The *utimes()* function shall fail if:

48589 [EACCES] Search permission is denied by a component of the path prefix; or the *times*  
 48590 argument is a null pointer and the effective user ID of the process does not  
 48591 match the owner of the file and write access is denied.

48592 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 48593 argument.

48594 [ENAMETOOLONG]  
 48595 The length of the *path* argument exceeds {PATH\_MAX} or a pathname  
 48596 component is longer than {NAME\_MAX}.

48597 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

48598 [ENOTDIR] A component of the path prefix is not a directory.

48599 [EPERM] The *times* argument is not a null pointer and the calling process' effective user  
 48600 ID has write access to the file but does not match the owner of the file and the  
 48601 calling process does not have the appropriate privileges.

48602 [EROFS] The file system containing the file is read-only.

48603 The *utimes()* function may fail if:

48604 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 48605 resolution of the *path* argument.

48606 [ENAMETOOLONG]  
 48607 Pathname resolution of a symbolic link produced an intermediate result  
 48608 whose length exceeds {PATH\_MAX}.

48609 **EXAMPLES**

48610 None.

48611 **APPLICATION USAGE**

48612 For applications portability, the *utime()* function should be used to set file access and  
48613 modification times instead of *utimes()*.

48614 **RATIONALE**

48615 None.

48616 **FUTURE DIRECTIONS**

48617 This function may be withdrawn in a future version.

48618 **SEE ALSO**48619 *utime()*, the Base Definitions volume of IEEE Std 1003.1-2001, <sys/time.h>48620 **CHANGE HISTORY**

48621 First released in Issue 4, Version 2.

48622 **Issue 5**

48623 Moved from X/OPEN UNIX extension to BASE.

48624 **Issue 6**

48625 This function is marked LEGACY.

48626 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

48627 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
48628 [ELOOP] error condition is added.

48629 **NAME**48630 `va_arg`, `va_copy`, `va_end`, `va_start` — handle variable argument list48631 **SYNOPSIS**48632 `#include <stdarg.h>`48633 `type va_arg(va_list ap, type);`48634 `void va_copy(va_list dest, va_list src);`48635 `void va_end(va_list ap);`48636 `void va_start(va_list ap, argN);`48637 **DESCRIPTION**48638 Refer to the Base Definitions volume of IEEE Std 1003.1-2001, `<stdarg.h>`.

48639 **NAME**

48640 vfork — create a new process; share virtual memory

48641 **SYNOPSIS**

48642 OB XSI #include &lt;unistd.h&gt;

48643 pid\_t vfork(void);

48644

48645 **DESCRIPTION**

48646 The *vfork()* function shall be equivalent to *fork()*, except that the behavior is undefined if the  
 48647 process created by *vfork()* either modifies any data other than a variable of type **pid\_t** used to  
 48648 store the return value from *vfork()*, or returns from the function in which *vfork()* was called, or  
 48649 calls any other function before successfully calling *\_exit()* or one of the *exec* family of functions.

48650 **RETURN VALUE**

48651 Upon successful completion, *vfork()* shall return 0 to the child process and return the process ID  
 48652 of the child process to the parent process. Otherwise, -1 shall be returned to the parent, no child  
 48653 process shall be created, and *errno* shall be set to indicate the error.

48654 **ERRORS**48655 The *vfork()* function shall fail if:

48656 [EAGAIN] The system-wide limit on the total number of processes under execution  
 48657 would be exceeded, or the system-imposed limit on the total number of  
 48658 processes under execution by a single user would be exceeded.

48659 [ENOMEM] There is insufficient swap space for the new process.

48660 **EXAMPLES**

48661 None.

48662 **APPLICATION USAGE**

48663 Conforming applications are recommended not to depend on *vfork()*, but to use *fork()* instead.  
 48664 The *vfork()* function may be withdrawn in a future version.

48665 On some implementations, *vfork()* is equivalent to *fork()*.

48666 The *vfork()* function differs from *fork()* only in that the child process can share code and data  
 48667 with the calling process (parent process). This speeds cloning activity significantly at a risk to  
 48668 the integrity of the parent process if *vfork()* is misused.

48669 The use of *vfork()* for any purpose except as a prelude to an immediate call to a function from  
 48670 the *exec* family, or to *\_exit()*, is not advised.

48671 The *vfork()* function can be used to create new processes without fully copying the address  
 48672 space of the old process. If a forked process is simply going to call *exec*, the data space copied  
 48673 from the parent to the child by *fork()* is not used. This is particularly inefficient in a paged  
 48674 environment, making *vfork()* particularly useful. Depending upon the size of the parent's data  
 48675 space, *vfork()* can give a significant performance improvement over *fork()*.

48676 The *vfork()* function can normally be used just like *fork()*. It does not work, however, to return  
 48677 while running in the child's context from the caller of *vfork()* since the eventual return from  
 48678 *vfork()* would then return to a no longer existent stack frame. Care should be taken, also, to call  
 48679 *\_exit()* rather than *exit()* if *exec* cannot be used, since *exit()* flushes and closes standard I/O  
 48680 channels, thereby damaging the parent process' standard I/O data structures. (Even with *fork()*,  
 48681 it is wrong to call *exit()*, since buffered data would then be flushed twice.)

48682 If signal handlers are invoked in the child process after *vfork()*, they must follow the same rules  
 48683 as other code in the child process.

48684 **RATIONALE**

48685           None.

48686 **FUTURE DIRECTIONS**

48687           This function may be withdrawn in a future version.

48688 **SEE ALSO**48689           *exec*, *exit()*, *fork()*, *wait()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**unistd.h**>48690 **CHANGE HISTORY**

48691           First released in Issue 4, Version 2.

48692 **Issue 5**

48693           Moved from X/OPEN UNIX extension to BASE.

48694 **Issue 6**

48695           This function is marked obsolescent.

48696 **NAME**

48697 vfprintf, vprintf, vsnprintf, vsprintf — format output of a stdarg argument list

48698 **SYNOPSIS**

48699 #include &lt;stdarg.h&gt;

48700 #include &lt;stdio.h&gt;

48701 int vfprintf(FILE \*restrict *stream*, const char \*restrict *format*,  
48702 va\_list *ap*);48703 int vprintf(const char \*restrict *format*, va\_list *ap*);48704 int vsnprintf(char \*restrict *s*, size\_t *n*, const char \*restrict *format*,  
48705 va\_list *ap*);48706 int vsprintf(char \*restrict *s*, const char \*restrict *format*, va\_list *ap*);48707 **DESCRIPTION**48708 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
48709 conflict between the requirements described here and the ISO C standard is unintentional. This  
48710 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.48711 The *vprintf()*, *vfprintf()*, *vsnprintf()*, and *vsprintf()* functions shall be equivalent to *printf()*,  
48712 *fprintf()*, *snprintf()*, and *sprintf()* respectively, except that instead of being called with a variable  
48713 number of arguments, they are called with an argument list as defined by <stdarg.h>.48714 These functions shall not invoke the *va\_end* macro. As these functions invoke the *va\_arg* macro,  
48715 the value of *ap* after the return is unspecified.48716 **RETURN VALUE**48717 Refer to *fprintf()*.48718 **ERRORS**48719 Refer to *fprintf()*.48720 **EXAMPLES**

48721 None.

48722 **APPLICATION USAGE**48723 Applications using these functions should call *va\_end(ap)* afterwards to clean up.48724 **RATIONALE**

48725 None.

48726 **FUTURE DIRECTIONS**

48727 None.

48728 **SEE ALSO**48729 *fprintf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdarg.h>, <stdio.h>48730 **CHANGE HISTORY**

48731 First released in Issue 1. Derived from Issue 1 of the SVID.

48732 **Issue 5**48733 The *vsnprintf()* function is added.48734 **Issue 6**48735 The *vfprintf()*, *vprintf()*, *vsnprintf()*, and *vsprintf()* functions are updated for alignment with the  
48736 ISO/IEC 9899:1999 standard.

48737 **NAME**

48738 vfprintf, fprintf, vsprintf — format input of a stdarg argument list

48739 **SYNOPSIS**

48740 #include <stdarg.h>

48741 #include <stdio.h>

48742 int vfprintf(FILE \*restrict stream, const char \*restrict format,

48743 va\_list arg);

48744 int fprintf(const char \*restrict format, va\_list arg);

48745 int vsprintf(const char \*restrict s, const char \*restrict format,

48746 va\_list arg);

48747 **DESCRIPTION**

48748 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
48749 conflict between the requirements described here and the ISO C standard is unintentional. This  
48750 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

48751 The *vfprintf()*, *fprintf()*, and *vsprintf()* functions shall be equivalent to the *scanf()*, *fscanf()*, and  
48752 *sscanf()* functions, respectively, except that instead of being called with a variable number of  
48753 arguments, they are called with an argument list as defined in the <stdarg.h> header. These  
48754 functions shall not invoke the *va\_end* macro. As these functions invoke the *va\_arg* macro, the  
48755 value of *ap* after the return is unspecified.

48756 **RETURN VALUE**

48757 Refer to *fscanf()*.

48758 **ERRORS**

48759 Refer to *fscanf()*.

48760 **EXAMPLES**

48761 None.

48762 **APPLICATION USAGE**

48763 Applications using these functions should call *va\_end(ap)* afterwards to clean up.

48764 **RATIONALE**

48765 None.

48766 **FUTURE DIRECTIONS**

48767 None.

48768 **SEE ALSO**

48769 *fscanf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdarg.h>, <stdio.h>

48770 **CHANGE HISTORY**

48771 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

48772 **NAME**

48773 vfwprintf, vswprintf, vwprintf — wide-character formatted output of a stdarg argument list

48774 **SYNOPSIS**

48775 #include &lt;stdarg.h&gt;

48776 #include &lt;stdio.h&gt;

48777 #include &lt;wchar.h&gt;

48778 int vfwprintf(FILE \*restrict stream, const wchar\_t \*restrict format,  
48779 va\_list arg);

48780 int vswprintf(wchar\_t \*restrict ws, size\_t n,

48781 const wchar\_t \*restrict format, va\_list arg);

48782 int vwprintf(const wchar\_t \*restrict format, va\_list arg);

48783 **DESCRIPTION**48784 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
48785 conflict between the requirements described here and the ISO C standard is unintentional. This  
48786 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.48787 The *vfwprintf()*, *vswprintf()*, and *vwprintf()* functions shall be equivalent to *fwprintf()*, *swprintf()*,  
48788 and *wprintf()* respectively, except that instead of being called with a variable number of  
48789 arguments, they are called with an argument list as defined by <stdarg.h>.48790 These functions shall not invoke the *va\_end* macro. However, as these functions do invoke the  
48791 *va\_arg* macro, the value of *ap* after the return is unspecified.48792 **RETURN VALUE**48793 Refer to *fwprintf()*.48794 **ERRORS**48795 Refer to *fwprintf()*.48796 **EXAMPLES**

48797 None.

48798 **APPLICATION USAGE**48799 Applications using these functions should call *va\_end(ap)* afterwards to clean up.48800 **RATIONALE**

48801 None.

48802 **FUTURE DIRECTIONS**

48803 None.

48804 **SEE ALSO**48805 *fwprintf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdarg.h>, <stdio.h>,  
48806 <wchar.h>48807 **CHANGE HISTORY**48808 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
48809 (E).48810 **Issue 6**48811 The *vfwprintf()*, *vswprintf()*, and *vwprintf()* prototypes are updated for alignment with the  
48812 ISO/IEC 9899:1999 standard. ()

48813 **NAME**

48814 vfwscanf, vswscanf, wscanf — wide-character formatted input of a stdarg argument list

48815 **SYNOPSIS**

48816 #include <stdarg.h>

48817 #include <stdio.h>

48818 #include <wchar.h>

48819 int vfwscanf(FILE \*restrict *stream*, const wchar\_t \*restrict *format*,  
48820 va\_list *arg*);

48821 int vswscanf(const wchar\_t \*restrict *ws*, const wchar\_t \*restrict *format*,  
48822 va\_list *arg*);

48823 int wscanf(const wchar\_t \*restrict *format*, va\_list *arg*);

48824 **DESCRIPTION**

48825 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
48826 conflict between the requirements described here and the ISO C standard is unintentional. This  
48827 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

48828 The *vfwscanf()*, *vswscanf()*, and *wscanf()* functions shall be equivalent to the *fwscanf()*,  
48829 *swscanf()*, and *wscanf()* functions, respectively, except that instead of being called with a  
48830 variable number of arguments, they are called with an argument list as defined in the <stdarg.h>  
48831 header. These functions shall not invoke the *va\_end* macro. As these functions invoke the *va\_arg*  
48832 macro, the value of *ap* after the return is unspecified.

48833 **RETURN VALUE**

48834 Refer to *fwscanf()*.

48835 **ERRORS**

48836 Refer to *fwscanf()*.

48837 **EXAMPLES**

48838 None.

48839 **APPLICATION USAGE**

48840 Applications using these functions should call *va\_end(ap)* afterwards to clean up.

48841 **RATIONALE**

48842 None.

48843 **FUTURE DIRECTIONS**

48844 None.

48845 **SEE ALSO**

48846 *fwscanf()*, the Base Definitions volume of IEEE Std 1003.1-2001, <stdarg.h>, <stdio.h>,  
48847 <wchar.h>

48848 **CHANGE HISTORY**

48849 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

48850 **NAME**

48851 vprintf — format the output of a stdarg argument list

48852 **SYNOPSIS**

48853 #include &lt;stdarg.h&gt;

48854 #include &lt;stdio.h&gt;

48855 int vprintf(const char \*restrict *format*, va\_list *ap*);48856 **DESCRIPTION**48857 Refer to *fprintf()*.

48858 **NAME**

48859           vscanf — format input of a stdarg argument list

48860 **SYNOPSIS**

48861           #include &lt;stdarg.h&gt;

48862           #include &lt;stdio.h&gt;

48863           int vscanf(const char \*restrict *format*, va\_list *arg*);48864 **DESCRIPTION**48865           Refer to *vscanf()*.

48866 **NAME**

48867        vsnprintf, vsprintf — format output of a stdarg argument list

48868 **SYNOPSIS**

48869        #include &lt;stdarg.h&gt;

48870        #include &lt;stdio.h&gt;

48871        int vsnprintf(char \*restrict *s*, size\_t *n*,48872            const char \*restrict *format*, va\_list *ap*);48873        int vsprintf(char \*restrict *s*, const char \*restrict *format*,48874            va\_list *ap*);48875 **DESCRIPTION**48876        Refer to *fprintf()*.

48877 **NAME**

48878       vsscanf — format input of a stdarg argument list

48879 **SYNOPSIS**

48880       #include &lt;stdarg.h&gt;

48881       #include &lt;stdio.h&gt;

48882       int vsscanf(const char \*restrict *s*, const char \*restrict *format*,48883               va\_list *arg*);48884 **DESCRIPTION**48885       Refer to *vfscanf()*.

48886 **NAME**

48887       vswprintf — wide-character formatted output of a stdarg argument list

48888 **SYNOPSIS**

48889       #include &lt;stdarg.h&gt;

48890       #include &lt;stdio.h&gt;

48891       #include &lt;wchar.h&gt;

48892       int vswprintf(wchar\_t \*restrict *ws*, size\_t *n*,  
48893                    const wchar\_t \*restrict *format*, va\_list *arg*);48894 **DESCRIPTION**48895       Refer to *vfwprintf()*.

48896 **NAME**

48897       vswscanf — wide-character formatted input of a stdarg argument list

48898 **SYNOPSIS**

48899       #include &lt;stdarg.h&gt;

48900       #include &lt;stdio.h&gt;

48901       #include &lt;wchar.h&gt;

48902       int vswscanf(const wchar\_t \*restrict *ws*, const wchar\_t \*restrict *format*,48903               va\_list *arg*);48904 **DESCRIPTION**48905       Refer to *vfwscanf()*.

48906 **NAME**

48907           vwprintf — wide-character formatted output of a stdarg argument list

48908 **SYNOPSIS**

48909           #include &lt;stdarg.h&gt;

48910           #include &lt;stdio.h&gt;

48911           #include &lt;wchar.h&gt;

48912           int vwprintf(const wchar\_t \*restrict *format*, va\_list *arg*);48913 **DESCRIPTION**48914           Refer to *vfwprintf()*.

48915 **NAME**

48916           vwscanf — wide-character formatted input of a stdarg argument list

48917 **SYNOPSIS**

48918           #include &lt;stdarg.h&gt;

48919           #include &lt;stdio.h&gt;

48920           #include &lt;wchar.h&gt;

48921           int vwscanf(const wchar\_t \*restrict *format*, va\_list *arg*);48922 **DESCRIPTION**48923           Refer to *vfwscanf()*.

48924 **NAME**

48925 wait, waitpid — wait for a child process to stop or terminate

48926 **SYNOPSIS**

48927 #include &lt;sys/wait.h&gt;

48928 pid\_t wait(int \*stat\_loc);

48929 pid\_t waitpid(pid\_t pid, int \*stat\_loc, int options);

48930 **DESCRIPTION**

48931 The *wait()* and *waitpid()* functions shall obtain status information pertaining to one of the  
 48932 caller's child processes. Various options permit status information to be obtained for child  
 48933 processes that have terminated or stopped. If status information is available for two or more  
 48934 child processes, the order in which their status is reported is unspecified.

48935 The *wait()* function shall suspend execution of the calling thread until status information for one  
 48936 of the terminated child processes of the calling process is available, or until delivery of a signal  
 48937 whose action is either to execute a signal-catching function or to terminate the process. If more  
 48938 than one thread is suspended in *wait()* or *waitpid()* awaiting termination of the same process,  
 48939 exactly one thread shall return the process status at the time of the target process termination. If  
 48940 status information is available prior to the call to *wait()*, return shall be immediate.

48941 The *waitpid()* function shall be equivalent to *wait()* if the *pid* argument is **(pid\_t)-1** and the  
 48942 *options* argument is 0. Otherwise, its behavior shall be modified by the values of the *pid* and  
 48943 *options* arguments.

48944 The *pid* argument specifies a set of child processes for which *status* is requested. The *waitpid()*  
 48945 function shall only return the status of a child process from this set:

- 48946 • If *pid* is equal to **(pid\_t)-1**, *status* is requested for any child process. In this respect, *waitpid()*  
 48947 is then equivalent to *wait()*.
- 48948 • If *pid* is greater than 0, it specifies the process ID of a single child process for which *status* is  
 48949 requested.
- 48950 • If *pid* is 0, *status* is requested for any child process whose process group ID is equal to that of  
 48951 the calling process.
- 48952 • If *pid* is less than **(pid\_t)-1**, *status* is requested for any child process whose process group ID  
 48953 is equal to the absolute value of *pid*.

48954 The *options* argument is constructed from the bitwise-inclusive OR of zero or more of the  
 48955 following flags, defined in the <sys/wait.h> header:

48956 XSI **WCONTINUED** The *waitpid()* function shall report the status of any continued child process  
 48957 specified by *pid* whose status has not been reported since it continued from a  
 48958 job control stop.

48959 **WNOHANG** The *waitpid()* function shall not suspend execution of the calling thread if  
 48960 *status* is not immediately available for one of the child processes specified by  
 48961 *pid*.

48962 **WUNTRACED** The status of any child processes specified by *pid* that are stopped, and whose  
 48963 status has not yet been reported since they stopped, shall also be reported to  
 48964 the requesting process.

48965 XSI If the calling process has SA\_NOCLDWAIT set or has SIGCHLD set to SIG\_IGN, and the  
 48966 process has no unwaited-for children that were transformed into zombie processes, the calling  
 48967 thread shall block until all of the children of the process containing the calling thread terminate,  
 48968 and *wait()* and *waitpid()* shall fail and set *errno* to [ECHILD].

48969 If *wait()* or *waitpid()* return because the status of a child process is available, these functions  
 48970 shall return a value equal to the process ID of the child process. In this case, if the value of the  
 48971 argument *stat\_loc* is not a null pointer, information shall be stored in the location pointed to by  
 48972 *stat\_loc*. The value stored at the location pointed to by *stat\_loc* shall be 0 if and only if the status  
 48973 returned is from a terminated child process that terminated by one of the following means:

- 48974 1. The process returned 0 from *main()*.
- 48975 2. The process called *\_exit()* or *exit()* with a *status* argument of 0.
- 48976 3. The process was terminated because the last thread in the process terminated.

48977 Regardless of its value, this information may be interpreted using the following macros, which  
 48978 are defined in `<sys/wait.h>` and evaluate to integral expressions; the *stat\_val* argument is the  
 48979 integer value pointed to by *stat\_loc*.

48980 **WIFEXITED(*stat\_val*)**

48981 Evaluates to a non-zero value if *status* was returned for a child process that terminated  
 48982 normally.

48983 **WEXITSTATUS(*stat\_val*)**

48984 If the value of **WIFEXITED(*stat\_val*)** is non-zero, this macro evaluates to the low-order 8 bits  
 48985 of the *status* argument that the child process passed to *\_exit()* or *exit()*, or the value the child  
 48986 process returned from *main()*.

48987 **WIFSIGNALED(*stat\_val*)**

48988 Evaluates to a non-zero value if *status* was returned for a child process that terminated due  
 48989 to the receipt of a signal that was not caught (see `<signal.h>`).

48990 **WTERMSIG(*stat\_val*)**

48991 If the value of **WIFSIGNALED(*stat\_val*)** is non-zero, this macro evaluates to the number of  
 48992 the signal that caused the termination of the child process.

48993 **WIFSTOPPED(*stat\_val*)**

48994 Evaluates to a non-zero value if *status* was returned for a child process that is currently  
 48995 stopped.

48996 **WSTOPSIG(*stat\_val*)**

48997 If the value of **WIFSTOPPED(*stat\_val*)** is non-zero, this macro evaluates to the number of the  
 48998 signal that caused the child process to stop.

48999 XSI **WIFCONTINUED(*stat\_val*)**

49000 Evaluates to a non-zero value if *status* was returned for a child process that has continued  
 49001 from a job control stop.

49002 SPN It is unspecified whether the *status* value returned by calls to *wait()* or *waitpid()* for processes  
 49003 created by *posix\_spawn()* or *posix\_spawnnp()* can indicate a **WIFSTOPPED(*stat\_val*)** before  
 49004 subsequent calls to *wait()* or *waitpid()* indicate **WIFEXITED(*stat\_val*)** as the result of an error  
 49005 detected before the new process image starts executing.

49006 It is unspecified whether the *status* value returned by calls to *wait()* or *waitpid()* for processes  
 49007 created by *posix\_spawn()* or *posix\_spawnnp()* can indicate a **WIFSIGNALED(*stat\_val*)** if a signal is  
 49008 sent to the parent's process group after *posix\_spawn()* or *posix\_spawnnp()* is called.

49009 If the information pointed to by *stat\_loc* was stored by a call to *waitpid()* that specified the  
 49010 XSI **WUNTRACED** flag and did not specify the **WCONTINUED** flag, exactly one of the macros  
 49011 **WIFEXITED(\**stat\_loc*)**, **WIFSIGNALED(\**stat\_loc*)**, and **WIFSTOPPED(\**stat\_loc*)** shall evaluate to  
 49012 a non-zero value.

49013 If the information pointed to by *stat\_loc* was stored by a call to *waitpid()* that specified the  
 49014 XSI WUNTRACED and WCONTINUED flags, exactly one of the macros WIFEXITED(\**stat\_loc*),  
 49015 XSI WIFSIGNALED(\**stat\_loc*), WIFSTOPPED(\**stat\_loc*), and WIFCONTINUED(\**stat\_loc*) shall  
 49016 evaluate to a non-zero value.

49017 If the information pointed to by *stat\_loc* was stored by a call to *waitpid()* that did not specify the  
 49018 XSI WUNTRACED or WCONTINUED flags, or by a call to the *wait()* function, exactly one of the  
 49019 macros WIFEXITED(\**stat\_loc*) and WIFSIGNALED(\**stat\_loc*) shall evaluate to a non-zero value.

49020 If the information pointed to by *stat\_loc* was stored by a call to *waitpid()* that did not specify the  
 49021 XSI WUNTRACED flag and specified the WCONTINUED flag, or by a call to the *wait()* function,  
 49022 XSI exactly one of the macros WIFEXITED(\**stat\_loc*), WIFSIGNALED(\**stat\_loc*), and  
 49023 WIFCONTINUED(\**stat\_loc*) shall evaluate to a non-zero value.

49024 If \_POSIX\_REALTIME\_SIGNALS is defined, and the implementation queues the SIGCHLD  
 49025 signal, then if *wait()* or *waitpid()* returns because the status of a child process is available, any  
 49026 pending SIGCHLD signal associated with the process ID of the child process shall be discarded.  
 49027 Any other pending SIGCHLD signals shall remain pending.

49028 Otherwise, if SIGCHLD is blocked, if *wait()* or *waitpid()* return because the status of a child  
 49029 process is available, any pending SIGCHLD signal shall be cleared unless the status of another  
 49030 child process is available.

49031 For all other conditions, it is unspecified whether child *status* will be available when a SIGCHLD  
 49032 signal is delivered.

49033 There may be additional implementation-defined circumstances under which *wait()* or *waitpid()*  
 49034 report *status*. This shall not occur unless the calling process or one of its child processes explicitly  
 49035 makes use of a non-standard extension. In these cases the interpretation of the reported *status* is  
 49036 implementation-defined.

49037 XSI If a parent process terminates without waiting for all of its child processes to terminate, the  
 49038 remaining child processes shall be assigned a new parent process ID corresponding to an  
 49039 implementation-defined system process.

#### 49040 RETURN VALUE

49041 If *wait()* or *waitpid()* returns because the status of a child process is available, these functions  
 49042 shall return a value equal to the process ID of the child process for which *status* is reported. If  
 49043 *wait()* or *waitpid()* returns due to the delivery of a signal to the calling process, -1 shall be  
 49044 returned and *errno* set to [EINTR]. If *waitpid()* was invoked with WNOHANG set in *options*, it  
 49045 has at least one child process specified by *pid* for which *status* is not available, and *status* is not  
 49046 available for any process specified by *pid*, 0 is returned. Otherwise, (pid\_t)-1 shall be returned,  
 49047 and *errno* set to indicate the error.

#### 49048 ERRORS

49049 The *wait()* function shall fail if:

49050 [ECHILD] The calling process has no existing unwaited-for child processes.

49051 [EINTR] The function was interrupted by a signal. The value of the location pointed to  
 49052 by *stat\_loc* is undefined.

49053 The *waitpid()* function shall fail if:

49054 [ECHILD] The process specified by *pid* does not exist or is not a child of the calling  
 49055 process, or the process group specified by *pid* does not exist or does not have  
 49056 any member process that is a child of the calling process.

49057 [EINTR] The function was interrupted by a signal. The value of the location pointed to  
49058 by *stat\_loc* is undefined.

49059 [EINVAL] The *options* argument is not valid.

#### 49060 EXAMPLES

49061 None.

#### 49062 APPLICATION USAGE

49063 None.

#### 49064 RATIONALE

49065 A call to the *wait()* or *waitpid()* function only returns *status* on an immediate child process of the  
49066 calling process; that is, a child that was produced by a single *fork()* call (perhaps followed by an  
49067 *exec* or other function calls) from the parent. If a child produces grandchildren by further use of  
49068 *fork()*, none of those grandchildren nor any of their descendants affect the behavior of a *wait()*  
49069 from the original parent process. Nothing in this volume of IEEE Std 1003.1-2001 prevents an  
49070 implementation from providing extensions that permit a process to get *status* from a grandchild  
49071 or any other process, but a process that does not use such extensions must be guaranteed to see  
49072 *status* from only its direct children.

49073 The *waitpid()* function is provided for three reasons:

- 49074 1. To support job control
- 49075 2. To permit a non-blocking version of the *wait()* function
- 49076 3. To permit a library routine, such as *system()* or *pclose()*, to wait for its children without  
49077 interfering with other terminated children for which the process has not waited

49078 The first two of these facilities are based on the *wait3()* function provided by 4.3 BSD. The  
49079 function uses the *options* argument, which is equivalent to an argument to *wait3()*. The  
49080 WUNTRACED flag is used only in conjunction with job control on systems supporting job  
49081 control. Its name comes from 4.3 BSD and refers to the fact that there are two types of stopped  
49082 processes in that implementation: processes being traced via the *ptrace()* debugging facility and  
49083 (untraced) processes stopped by job control signals. Since *ptrace()* is not part of this volume of  
49084 IEEE Std 1003.1-2001, only the second type is relevant. The name WUNTRACED was retained  
49085 because its usage is the same, even though the name is not intuitively meaningful in this context.

49086 The third reason for the *waitpid()* function is to permit independent sections of a process to  
49087 spawn and wait for children without interfering with each other. For example, the following  
49088 problem occurs in developing a portable shell, or command interpreter:

```
49089 stream = popen("/bin/true");
49090 (void) system("sleep 100");
49091 (void) pclose(stream);
```

49092 On all historical implementations, the final *pclose()* fails to reap the *wait()* *status* of the *popen()*.

49093 The status values are retrieved by macros, rather than given as specific bit encodings as they are  
49094 in most historical implementations (and thus expected by existing programs). This was  
49095 necessary to eliminate a limitation on the number of signals an implementation can support that  
49096 was inherent in the traditional encodings. This volume of IEEE Std 1003.1-2001 does require that  
49097 a *status* value of zero corresponds to a process calling *\_exit(0)*, as this is the most common  
49098 encoding expected by existing programs. Some of the macro names were adopted from 4.3 BSD.

49099 These macros syntactically operate on an arbitrary integer value. The behavior is undefined  
49100 unless that value is one stored by a successful call to *wait()* or *waitpid()* in the location pointed  
49101 to by the *stat\_loc* argument. An early proposal attempted to make this clearer by specifying each

49102 argument as *\*stat\_loc* rather than *stat\_val*. However, that did not follow the conventions of other  
 49103 specifications in this volume of IEEE Std 1003.1-2001 or traditional usage. It also could have  
 49104 implied that the argument to the macro must literally be *\*stat\_loc*; in fact, that value can be  
 49105 stored or passed as an argument to other functions before being interpreted by these macros.

49106 The extension that affects *wait()* and *waitpid()* and is common in historical implementations is  
 49107 the *ptrace()* function. It is called by a child process and causes that child to stop and return a  
 49108 *status* that appears identical to the *status* indicated by WIFSTOPPED. The *status* of *ptrace()*  
 49109 children is traditionally returned regardless of the WUNTRACED flag (or by the *wait()*  
 49110 function). Most applications do not need to concern themselves with such extensions because  
 49111 they have control over what extensions they or their children use. However, applications, such  
 49112 as command interpreters, that invoke arbitrary processes may see this behavior when those  
 49113 arbitrary processes misuse such extensions.

49114 Implementations that support **core** file creation or other implementation-defined actions on  
 49115 termination of some processes traditionally provide a bit in the *status* returned by *wait()* to  
 49116 indicate that such actions have occurred.

49117 Allowing the *wait()* family of functions to discard a pending SIGCHLD signal that is associated  
 49118 with a successfully waited-for child process puts them into the *sigwait()* and *sigwaitinfo()*  
 49119 category with respect to SIGCHLD.

49120 This definition allows implementations to treat a pending SIGCHLD signal as accepted by the  
 49121 process in *wait()*, with the same meaning of “accepted” as when that word is applied to the  
 49122 *sigwait()* family of functions.

49123 Allowing the *wait()* family of functions to behave this way permits an implementation to be able  
 49124 to deal precisely with SIGCHLD signals.

49125 In particular, an implementation that does accept (discard) the SIGCHLD signal can make the  
 49126 following guarantees regardless of the queuing depth of signals in general (the list of waitable  
 49127 children can hold the SIGCHLD queue):

- 49128 1. If a SIGCHLD signal handler is established via *sigaction()* without the SA\_RESETHAND  
 49129 flag, SIGCHLD signals can be accurately counted; that is, exactly one SIGCHLD signal will  
 49130 be delivered to or accepted by the process for every child process that terminates.
- 49131 2. A single *wait()* issued from a SIGCHLD signal handler can be guaranteed to return  
 49132 immediately with status information for a child process.
- 49133 3. When SA\_SIGINFO is requested, the SIGCHLD signal handler can be guaranteed to  
 49134 receive a non-NULL pointer to a **siginfo\_t** structure that describes a child process for  
 49135 which a wait via *waitpid()* or *waitid()* will not block or fail.
- 49136 4. The *system()* function will not cause a process' SIGCHLD handler to be called as a result of  
 49137 the *fork()/exec* executed within *system()* because *system()* will accept the SIGCHLD signal  
 49138 when it performs a *waitpid()* for its child process. This is a desirable behavior of *system()*  
 49139 so that it can be used in a library without causing side effects to the application linked with  
 49140 the library.

49141 An implementation that does not permit the *wait()* family of functions to accept (discard) a  
 49142 pending SIGCHLD signal associated with a successfully waited-for child, cannot make the  
 49143 guarantees described above for the following reasons:

#### 49144 Guarantee #1

49145 Although it might be assumed that reliable queuing of all SIGCHLD signals generated by  
 49146 the system can make this guarantee, the counter-example is the case of a process that blocks  
 49147 SIGCHLD and performs an indefinite loop of *fork()/wait()* operations. If the

49148 implementation supports queued signals, then eventually the system will run out of  
49149 memory for the queue. The guarantee cannot be made because there must be some limit to  
49150 the depth of queuing.

49151 Guarantees #2 and #3

49152 These cannot be guaranteed unless the *wait()* family of functions accepts the SIGCHLD  
49153 signal. Otherwise, a *fork()/wait()* executed while SIGCHLD is blocked (as in the *system()*  
49154 function) will result in an invocation of the handler when SIGCHLD is unblocked, after the  
49155 process has disappeared.

49156 Guarantee #4

49157 Although possible to make this guarantee, *system()* would have to set the SIGCHLD  
49158 handler to SIG\_DFL so that the SIGCHLD signal generated by its *fork()* would be discarded  
49159 (the SIGCHLD default action is to be ignored), then restore it to its previous setting. This  
49160 would have the undesirable side effect of discarding all SIGCHLD signals pending to the  
49161 process.

#### 49162 FUTURE DIRECTIONS

49163 None.

#### 49164 SEE ALSO

49165 *exec*, *exit()*, *fork()*, *waitid()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**signal.h**>,  
49166 <**sys/wait.h**>

#### 49167 CHANGE HISTORY

49168 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 49169 Issue 5

49170 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

#### 49171 Issue 6

49172 The following new requirements on POSIX implementations derive from alignment with the  
49173 Single UNIX Specification:

49174 • The requirement to include <**sys/types.h**> has been removed. Although <**sys/types.h**> was  
49175 required for conforming implementations of previous POSIX specifications, it was not  
49176 required for UNIX applications.

49177 The following changes were made to align with the IEEE P1003.1a draft standard:

49178 • The processing of the SIGCHLD signal and the [ECHILD] error is clarified.

49179 The semantics of *WIFSTOPPED(stat\_val)*, *WIFEXITED(stat\_val)*, and *WIFSIGNALED(stat\_val)*  
49180 are defined with respect to *posix\_spawn()* or *posix\_spawnnp()* for alignment with  
49181 IEEE Std 1003.1d-1999.

49182 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

## 49183 NAME

49184 waitid — wait for a child process to change state

## 49185 SYNOPSIS

49186 XSI #include &lt;sys/wait.h&gt;

49187 int waitid(idtype\_t idtype, id\_t id, siginfo\_t \*infop, int options);

49188

## 49189 DESCRIPTION

49190 The *waitid()* function shall suspend the calling thread until one child of the process containing  
 49191 the calling thread changes state. It records the current state of a child in the structure pointed to  
 49192 by *infop*. If a child process changed state prior to the call to *waitid()*, *waitid()* shall return  
 49193 immediately. If more than one thread is suspended in *wait()* or *waitpid()* waiting for termination  
 49194 of the same process, exactly one thread shall return the process status at the time of the target  
 49195 process termination.

49196 The *idtype* and *id* arguments are used to specify which children *waitid()* waits for.

49197 If *idtype* is P\_PID, *waitid()* shall wait for the child with a process ID equal to (**pid\_t**)*id*.

49198 If *idtype* is P\_PGID, *waitid()* shall wait for any child with a process group ID equal to (**pid\_t**)*id*.

49199 If *idtype* is P\_ALL, *waitid()* shall wait for any children and *id* is ignored.

49200 The *options* argument is used to specify which state changes *waitid()* shall wait for. It is formed  
 49201 by OR'ing together one or more of the following flags:

49202 WEXITED Wait for processes that have exited.

49203 WSTOPPED Status shall be returned for any child that has stopped upon receipt of a signal.

49204 WCONTINUED Status shall be returned for any child that was stopped and has been  
 49205 continued.

49206 WNOHANG Return immediately if there are no children to wait for.

49207 WNOWAIT Keep the process whose status is returned in *infop* in a waitable state. This  
 49208 shall not affect the state of the process; the process may be waited for again  
 49209 after this call completes.

49210 The application shall ensure that the *infop* argument points to a **siginfo\_t** structure. If *waitid()*  
 49211 returns because a child process was found that satisfied the conditions indicated by the  
 49212 arguments *idtype* and *options*, then the structure pointed to by *infop* shall be filled in by the  
 49213 system with the status of the process. The *si\_signo* member shall always be equal to SIGCHLD.

## 49214 RETURN VALUE

49215 If WNOHANG was specified and there are no children to wait for, 0 shall be returned. If *waitid()*  
 49216 returns due to the change of state of one of its children, 0 shall be returned. Otherwise, -1 shall  
 49217 be returned and *errno* set to indicate the error.

## 49218 ERRORS

49219 The *waitid()* function shall fail if:

49220 [ECHILD] The calling process has no existing unwaited-for child processes.

49221 [EINTR] The *waitid()* function was interrupted by a signal.

49222 [EINVAL] An invalid value was specified for *options*, or *idtype* and *id* specify an invalid  
 49223 set of processes.

49224 **EXAMPLES**

49225 None.

49226 **APPLICATION USAGE**

49227 None.

49228 **RATIONALE**

49229 None.

49230 **FUTURE DIRECTIONS**

49231 None.

49232 **SEE ALSO**49233 *exec*, *exit()*, *wait()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**sys/wait.h**>49234 **CHANGE HISTORY**

49235 First released in Issue 4, Version 2.

49236 **Issue 5**

49237 Moved from X/OPEN UNIX extension to BASE.

49238 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

49239 **Issue 6**

49240 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

49241 **NAME**

49242       waitpid — wait for a child process to stop or terminate

49243 **SYNOPSIS**

49244       #include &lt;sys/wait.h&gt;

49245       pid\_t waitpid(pid\_t *pid*, int \**stat\_loc*, int *options*);49246 **DESCRIPTION**49247       Refer to *wait()*.

49248 **NAME**

49249 wrtomb — convert a wide-character code to a character (restartable)

49250 **SYNOPSIS**

49251 #include &lt;stdio.h&gt;

49252 size\_t wrtomb(char \*restrict *s*, wchar\_t *wc*, mbstate\_t \*restrict *ps*);49253 **DESCRIPTION**

49254 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 49255 conflict between the requirements described here and the ISO C standard is unintentional. This  
 49256 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49257 If *s* is a null pointer, the *wrtomb()* function shall be equivalent to the call:49258 wrtomb(*buf*, L'\0', *ps*)49259 where *buf* is an internal buffer.

49260 If *s* is not a null pointer, the *wrtomb()* function shall determine the number of bytes needed to  
 49261 represent the character that corresponds to the wide character given by *wc* (including any shift  
 49262 sequences), and store the resulting bytes in the array whose first element is pointed to by *s*. At  
 49263 most {MB\_CUR\_MAX} bytes are stored. If *wc* is a null wide character, a null byte shall be stored,  
 49264 preceded by any shift sequence needed to restore the initial shift state. The resulting state  
 49265 described shall be the initial conversion state.

49266 If *ps* is a null pointer, the *wrtomb()* function shall use its own internal **mbstate\_t** object, which is  
 49267 initialized at program start-up to the initial conversion state. Otherwise, the **mbstate\_t** object  
 49268 pointed to by *ps* shall be used to completely describe the current conversion state of the  
 49269 associated character sequence. The implementation shall behave as if no function defined in this  
 49270 volume of IEEE Std 1003.1-2001 calls *wrtomb()*.

49271 cx If the application uses any of the **\_POSIX\_THREAD\_SAFE\_FUNCTIONS** or **\_POSIX\_THREADS**  
 49272 functions, the application shall ensure that the *wrtomb()* function is called with a non-NULL *ps*  
 49273 argument.

49274 The behavior of this function shall be affected by the **LC\_CTYPE** category of the current locale.49275 **RETURN VALUE**

49276 The *wrtomb()* function shall return the number of bytes stored in the array object (including any  
 49277 shift sequences). When *wc* is not a valid wide character, an encoding error shall occur. In this  
 49278 case, the function shall store the value of the macro [EILSEQ] in *errno* and shall return (**size\_t**)-1;  
 49279 the conversion state shall be undefined.

49280 **ERRORS**49281 The *wrtomb()* function may fail if:49282 cx [EINVAL] *ps* points to an object that contains an invalid conversion state.

49283 [EILSEQ] Invalid wide-character code is detected.

49284 **EXAMPLES**

49285 None.

49286 **APPLICATION USAGE**

49287 None.

49288 **RATIONALE**

49289 None.

49290 **FUTURE DIRECTIONS**

49291 None.

49292 **SEE ALSO**49293 *mbstinit()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>49294 **CHANGE HISTORY**

49295 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

49296 (E).

49297 **Issue 6**

49298 In the DESCRIPTION, a note on using this function in a threaded application is added.

49299 Extensions beyond the ISO C standard are marked.

49300 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

49301 The *wcr tomb()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49302 **NAME**

49303 wscat — concatenate two wide-character strings

49304 **SYNOPSIS**

49305 #include &lt;wchar.h&gt;

49306 wchar\_t \*wscat(wchar\_t \*restrict *ws1*, const wchar\_t \*restrict *ws2*);49307 **DESCRIPTION**

49308 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
49309 conflict between the requirements described here and the ISO C standard is unintentional. This  
49310 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49311 The *wscat()* function shall append a copy of the wide-character string pointed to by *ws2*  
49312 (including the terminating null wide-character code) to the end of the wide-character string  
49313 pointed to by *ws1*. The initial wide-character code of *ws2* shall overwrite the null wide-character  
49314 code at the end of *ws1*. If copying takes place between objects that overlap, the behavior is  
49315 undefined.

49316 **RETURN VALUE**49317 The *wscat()* function shall return *ws1*; no return value is reserved to indicate an error.49318 **ERRORS**

49319 No errors are defined.

49320 **EXAMPLES**

49321 None.

49322 **APPLICATION USAGE**

49323 None.

49324 **RATIONALE**

49325 None.

49326 **FUTURE DIRECTIONS**

49327 None.

49328 **SEE ALSO**49329 *wscncat()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>49330 **CHANGE HISTORY**

49331 First released in Issue 4. Derived from the MSE working draft.

49332 **Issue 6**49333 The Open Group Corrigendum U040/2 is applied. In the RETURN VALUE section, *s1* is changed  
49334 to *ws1*.49335 The *wscat()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49336 **NAME**49337 `wcschr` — wide-character string scanning operation49338 **SYNOPSIS**49339 `#include <wchar.h>`49340 `wchar_t *wcschr(const wchar_t *ws, wchar_t wc);`49341 **DESCRIPTION**

49342 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
49343 conflict between the requirements described here and the ISO C standard is unintentional. This  
49344 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49345 The `wcschr()` function shall locate the first occurrence of `wc` in the wide-character string pointed  
49346 to by `ws`. The application shall ensure that the value of `wc` is a character representable as a type  
49347 **wchar\_t** and a wide-character code corresponding to a valid character in the current locale. The  
49348 terminating null wide-character code is considered to be part of the wide-character string.

49349 **RETURN VALUE**

49350 Upon completion, `wcschr()` shall return a pointer to the wide-character code, or a null pointer if  
49351 the wide-character code is not found.

49352 **ERRORS**

49353 No errors are defined.

49354 **EXAMPLES**

49355 None.

49356 **APPLICATION USAGE**

49357 None.

49358 **RATIONALE**

49359 None.

49360 **FUTURE DIRECTIONS**

49361 None.

49362 **SEE ALSO**49363 `wcsrchr()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<wchar.h>`49364 **CHANGE HISTORY**

49365 First released in Issue 4. Derived from the MSE working draft.

49366 **Issue 6**

49367 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

49368 **NAME**

49369           wscmp — compare two wide-character strings

49370 **SYNOPSIS**

49371           #include &lt;wchar.h&gt;

49372           int wscmp(const wchar\_t \*ws1, const wchar\_t \*ws2);

49373 **DESCRIPTION**

49374 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
49375 conflict between the requirements described here and the ISO C standard is unintentional. This  
49376 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49377       The *wscmp()* function shall compare the wide-character string pointed to by *ws1* to the wide-  
49378 character string pointed to by *ws2*.

49379       The sign of a non-zero return value shall be determined by the sign of the difference between the  
49380 values of the first pair of wide-character codes that differ in the objects being compared.

49381 **RETURN VALUE**

49382       Upon completion, *wscmp()* shall return an integer greater than, equal to, or less than 0, if the  
49383 wide-character string pointed to by *ws1* is greater than, equal to, or less than the wide-character  
49384 string pointed to by *ws2*, respectively.

49385 **ERRORS**

49386       No errors are defined.

49387 **EXAMPLES**

49388       None.

49389 **APPLICATION USAGE**

49390       None.

49391 **RATIONALE**

49392       None.

49393 **FUTURE DIRECTIONS**

49394       None.

49395 **SEE ALSO**49396       *wcsncmp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>49397 **CHANGE HISTORY**

49398       First released in Issue 4. Derived from the MSE working draft.

49399 **NAME**

49400 `wscoll` — wide-character string comparison using collating information

49401 **SYNOPSIS**

49402 `#include <wchar.h>`

49403 `int wscoll(const wchar_t *ws1, const wchar_t *ws2);`

49404 **DESCRIPTION**

49405 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
49406 conflict between the requirements described here and the ISO C standard is unintentional. This  
49407 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49408 The `wscoll()` function shall compare the wide-character string pointed to by `ws1` to the wide-  
49409 character string pointed to by `ws2`, both interpreted as appropriate to the `LC_COLLATE` category  
49410 of the current locale.

49411 CX The `wscoll()` function shall not change the setting of `errno` if successful.

49412 An application wishing to check for error situations should set `errno` to 0 before calling `wscoll()`.  
49413 If `errno` is non-zero on return, an error has occurred.

49414 **RETURN VALUE**

49415 Upon successful completion, `wscoll()` shall return an integer greater than, equal to, or less than  
49416 0, according to whether the wide-character string pointed to by `ws1` is greater than, equal to, or  
49417 less than the wide-character string pointed to by `ws2`, when both are interpreted as appropriate  
49418 CX to the current locale. On error, `wscoll()` shall set `errno`, but no return value is reserved to  
49419 indicate an error.

49420 **ERRORS**

49421 The `wscoll()` function may fail if:

49422 CX [EINVAL] The `ws1` or `ws2` arguments contain wide-character codes outside the domain of  
49423 the collating sequence.

49424 **EXAMPLES**

49425 None.

49426 **APPLICATION USAGE**

49427 The `wcsxfrm()` and `wscmp()` functions should be used for sorting large lists.

49428 **RATIONALE**

49429 None.

49430 **FUTURE DIRECTIONS**

49431 None.

49432 **SEE ALSO**

49433 `wscmp()`, `wcsxfrm()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<wchar.h>`

49434 **CHANGE HISTORY**

49435 First released in Issue 4. Derived from the MSE working draft.

49436 **Issue 5**

49437 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

49438 The DESCRIPTION is updated to indicate that `errno` is not changed if the function is successful.

49439 **NAME**

49440 wscspy — copy a wide-character string

49441 **SYNOPSIS**

49442 #include &lt;wchar.h&gt;

49443 wchar\_t \*wscspy(wchar\_t \*restrict ws1, const wchar\_t \*restrict ws2);

49444 **DESCRIPTION**

49445 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
49446 conflict between the requirements described here and the ISO C standard is unintentional. This  
49447 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49448 The *wscspy()* function shall copy the wide-character string pointed to by *ws2* (including the  
49449 terminating null wide-character code) into the array pointed to by *ws1*. If copying takes place  
49450 between objects that overlap, the behavior is undefined.

49451 **RETURN VALUE**49452 The *wscspy()* function shall return *ws1*; no return value is reserved to indicate an error.49453 **ERRORS**

49454 No errors are defined.

49455 **EXAMPLES**

49456 None.

49457 **APPLICATION USAGE**

49458 None.

49459 **RATIONALE**

49460 None.

49461 **FUTURE DIRECTIONS**

49462 None.

49463 **SEE ALSO**49464 *wscncpy()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>49465 **CHANGE HISTORY**

49466 First released in Issue 4. Derived from the MSE working draft.

49467 **Issue 6**49468 The *wscspy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49469 **NAME**49470 `wcscspn` — get the length of a complementary wide substring49471 **SYNOPSIS**49472 `#include <wchar.h>`49473 `size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2);`49474 **DESCRIPTION**

49475 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
49476 conflict between the requirements described here and the ISO C standard is unintentional. This  
49477 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49478 The `wcscspn()` function shall compute the length (in wide characters) of the maximum initial  
49479 segment of the wide-character string pointed to by `ws1` which consists entirely of wide-character  
49480 codes *not* from the wide-character string pointed to by `ws2`.

49481 **RETURN VALUE**

49482 The `wcscspn()` function shall return the length of the initial substring of `ws1`; no return value is  
49483 reserved to indicate an error.

49484 **ERRORS**

49485 No errors are defined.

49486 **EXAMPLES**

49487 None.

49488 **APPLICATION USAGE**

49489 None.

49490 **RATIONALE**

49491 None.

49492 **FUTURE DIRECTIONS**

49493 None.

49494 **SEE ALSO**49495 `wcssp()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<wchar.h>`49496 **CHANGE HISTORY**

49497 First released in Issue 4. Derived from the MSE working draft.

49498 **Issue 5**

49499 The RETURN VALUE section is updated to indicate that `wcscspn()` returns the length of `ws1`,  
49500 rather than `ws1` itself.

49501 **NAME**49502 `wcsftime` — convert date and time to a wide-character string49503 **SYNOPSIS**49504 `#include <wchar.h>`49505 `size_t wcsftime(wchar_t *restrict wcs, size_t maxsize,`  
49506 `const wchar_t *restrict format, const struct tm *restrict timeptr);`49507 **DESCRIPTION**49508 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
49509 conflict between the requirements described here and the ISO C standard is unintentional. This  
49510 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.49511 The `wcsftime()` function shall be equivalent to the `strftime()` function, except that:

- 49512
- The argument `wcs` points to the initial element of an array of wide characters into which the  
49513 generated output is to be placed.
  - The argument `maxsize` indicates the maximum number of wide characters to be placed in the  
49514 output array.
  - The argument `format` is a wide-character string and the conversion specifications are replaced  
49515 by corresponding sequences of wide characters.
  - The return value indicates the number of wide characters placed in the output array.

49516  
49517  
49518  
49519 If copying takes place between objects that overlap, the behavior is undefined.49520 **RETURN VALUE**49521 If the total number of resulting wide-character codes including the terminating null wide-  
49522 character code is no more than `maxsize`, `wcsftime()` shall return the number of wide-character  
49523 codes placed into the array pointed to by `wcs`, not including the terminating null wide-character  
49524 code. Otherwise, zero is returned and the contents of the array are unspecified.49525 **ERRORS**

49526 No errors are defined.

49527 **EXAMPLES**

49528 None.

49529 **APPLICATION USAGE**

49530 None.

49531 **RATIONALE**

49532 None.

49533 **FUTURE DIRECTIONS**

49534 None.

49535 **SEE ALSO**49536 `strftime()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<wchar.h>`49537 **CHANGE HISTORY**

49538 First released in Issue 4.

49539 **Issue 5**

49540 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

49541 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of the `format`  
49542 argument is changed from `const char *` to `const wchar_t *`.

49543 **Issue 6**

49544

The *wcsftime()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49545 **NAME**

49546           wcslen — get wide-character string length

49547 **SYNOPSIS**

49548           #include &lt;wchar.h&gt;

49549           size\_t wcslen(const wchar\_t \*ws);

49550 **DESCRIPTION**

49551 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
49552       conflict between the requirements described here and the ISO C standard is unintentional. This  
49553       volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49554       The *wcslen()* function shall compute the number of wide-character codes in the wide-character  
49555       string to which *ws* points, not including the terminating null wide-character code.

49556 **RETURN VALUE**

49557       The *wcslen()* function shall return the length of *ws*; no return value is reserved to indicate an  
49558       error.

49559 **ERRORS**

49560       No errors are defined.

49561 **EXAMPLES**

49562       None.

49563 **APPLICATION USAGE**

49564       None.

49565 **RATIONALE**

49566       None.

49567 **FUTURE DIRECTIONS**

49568       None.

49569 **SEE ALSO**49570       The Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>49571 **CHANGE HISTORY**

49572       First released in Issue 4. Derived from the MSE working draft.

49573 **NAME**49574 `wcsncat` — concatenate a wide-character string with part of another49575 **SYNOPSIS**49576 `#include <wchar.h>`49577 `wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict ws2,`  
49578 `size_t n);`49579 **DESCRIPTION**49580 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
49581 conflict between the requirements described here and the ISO C standard is unintentional. This  
49582 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.49583 The `wcsncat()` function shall append not more than *n* wide-character codes (a null wide-  
49584 character code and wide-character codes that follow it are not appended) from the array pointed  
49585 to by *ws2* to the end of the wide-character string pointed to by *ws1*. The initial wide-character  
49586 code of *ws2* shall overwrite the null wide-character code at the end of *ws1*. A terminating null  
49587 wide-character code shall always be appended to the result. If copying takes place between  
49588 objects that overlap, the behavior is undefined.49589 **RETURN VALUE**49590 The `wcsncat()` function shall return *ws1*; no return value is reserved to indicate an error.49591 **ERRORS**

49592 No errors are defined.

49593 **EXAMPLES**

49594 None.

49595 **APPLICATION USAGE**

49596 None.

49597 **RATIONALE**

49598 None.

49599 **FUTURE DIRECTIONS**

49600 None.

49601 **SEE ALSO**49602 `wscat()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<wchar.h>`49603 **CHANGE HISTORY**

49604 First released in Issue 4. Derived from the MSE working draft.

49605 **Issue 6**49606 The `wcsncat()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49607 **NAME**

49608       wcsncmp — compare part of two wide-character strings

49609 **SYNOPSIS**

49610       #include &lt;wchar.h&gt;

49611       int wcsncmp(const wchar\_t \*ws1, const wchar\_t \*ws2, size\_t n);

49612 **DESCRIPTION**

49613 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
49614 conflict between the requirements described here and the ISO C standard is unintentional. This  
49615 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49616       The *wcsncmp()* function shall compare not more than *n* wide-character codes (wide-character  
49617 codes that follow a null wide-character code are not compared) from the array pointed to by *ws1*  
49618 to the array pointed to by *ws2*.

49619       The sign of a non-zero return value shall be determined by the sign of the difference between the  
49620 values of the first pair of wide-character codes that differ in the objects being compared.

49621 **RETURN VALUE**

49622       Upon successful completion, *wcsncmp()* shall return an integer greater than, equal to, or less  
49623 than 0, if the possibly null-terminated array pointed to by *ws1* is greater than, equal to, or less  
49624 than the possibly null-terminated array pointed to by *ws2*, respectively.

49625 **ERRORS**

49626       No errors are defined.

49627 **EXAMPLES**

49628       None.

49629 **APPLICATION USAGE**

49630       None.

49631 **RATIONALE**

49632       None.

49633 **FUTURE DIRECTIONS**

49634       None.

49635 **SEE ALSO**49636       *wscmp()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>49637 **CHANGE HISTORY**

49638       First released in Issue 4. Derived from the MSE working draft.

49639 **NAME**49640 `wcsncpy` — copy part of a wide-character string49641 **SYNOPSIS**49642 `#include <wchar.h>`49643 `wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,`  
49644 `size_t n);`49645 **DESCRIPTION**49646 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
49647 conflict between the requirements described here and the ISO C standard is unintentional. This  
49648 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.49649 The `wcsncpy()` function shall copy not more than *n* wide-character codes (wide-character codes  
49650 that follow a null wide-character code are not copied) from the array pointed to by *ws2* to the  
49651 array pointed to by *ws1*. If copying takes place between objects that overlap, the behavior is  
49652 undefined.49653 If the array pointed to by *ws2* is a wide-character string that is shorter than *n* wide-character  
49654 codes, null wide-character codes shall be appended to the copy in the array pointed to by *ws1*,  
49655 until *n* wide-character codes in all are written.49656 **RETURN VALUE**49657 The `wcsncpy()` function shall return *ws1*; no return value is reserved to indicate an error.49658 **ERRORS**

49659 No errors are defined.

49660 **EXAMPLES**

49661 None.

49662 **APPLICATION USAGE**49663 If there is no null wide-character code in the first *n* wide-character codes of the array pointed to  
49664 by *ws2*, the result is not null-terminated.49665 **RATIONALE**

49666 None.

49667 **FUTURE DIRECTIONS**

49668 None.

49669 **SEE ALSO**49670 `wscpy()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<wchar.h>`49671 **CHANGE HISTORY**

49672 First released in Issue 4. Derived from the MSE working draft.

49673 **Issue 6**49674 The `wcsncpy()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49675 **NAME**

49676 `wenspbrk` — scan a wide-character string for a wide-character code

49677 **SYNOPSIS**

49678 `#include <wchar.h>`

49679 `wchar_t *wenspbrk(const wchar_t *ws1, const wchar_t *ws2);`

49680 **DESCRIPTION**

49681 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
49682 conflict between the requirements described here and the ISO C standard is unintentional. This  
49683 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49684 The `wenspbrk()` function shall locate the first occurrence in the wide-character string pointed to by  
49685 `ws1` of any wide-character code from the wide-character string pointed to by `ws2`.

49686 **RETURN VALUE**

49687 Upon successful completion, `wenspbrk()` shall return a pointer to the wide-character code or a null  
49688 pointer if no wide-character code from `ws2` occurs in `ws1`.

49689 **ERRORS**

49690 No errors are defined.

49691 **EXAMPLES**

49692 None.

49693 **APPLICATION USAGE**

49694 None.

49695 **RATIONALE**

49696 None.

49697 **FUTURE DIRECTIONS**

49698 None.

49699 **SEE ALSO**

49700 `wenschr()`, `wensrchr()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<wchar.h>`

49701 **CHANGE HISTORY**

49702 First released in Issue 4. Derived from the MSE working draft.

49703 **NAME**

49704       wcsrchr — wide-character string scanning operation

49705 **SYNOPSIS**

49706       #include &lt;wchar.h&gt;

49707       wchar\_t \*wcsrchr(const wchar\_t \*ws, wchar\_t wc);

49708 **DESCRIPTION**

49709 cx     The functionality described on this reference page is aligned with the ISO C standard. Any  
49710     conflict between the requirements described here and the ISO C standard is unintentional. This  
49711     volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49712     The *wcsrchr()* function shall locate the last occurrence of *wc* in the wide-character string pointed  
49713     to by *ws*. The application shall ensure that the value of *wc* is a character representable as a type  
49714     **wchar\_t** and a wide-character code corresponding to a valid character in the current locale. The  
49715     terminating null wide-character code shall be considered to be part of the wide-character string.

49716 **RETURN VALUE**

49717     Upon successful completion, *wcsrchr()* shall return a pointer to the wide-character code or a null  
49718     pointer if *wc* does not occur in the wide-character string.

49719 **ERRORS**

49720     No errors are defined.

49721 **EXAMPLES**

49722     None.

49723 **APPLICATION USAGE**

49724     None.

49725 **RATIONALE**

49726     None.

49727 **FUTURE DIRECTIONS**

49728     None.

49729 **SEE ALSO**49730     *wchr()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>49731 **CHANGE HISTORY**

49732     First released in Issue 4. Derived from the MSE working draft.

49733 **Issue 6**

49734     The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

## 49735 NAME

49736       wcsrtoombs — convert a wide-character string to a character string (restartable)

## 49737 SYNOPSIS

49738       #include <wchar.h>

49739       size\_t wcsrtoombs(char \*restrict dst, const wchar\_t \*\*restrict src,  
49740                       size\_t len, mbstate\_t \*restrict ps);

## 49741 DESCRIPTION

49742 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
49743 conflict between the requirements described here and the ISO C standard is unintentional. This  
49744 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49745       The *wcsrtoombs()* function shall convert a sequence of wide characters from the array indirectly  
49746 pointed to by *src* into a sequence of corresponding characters, beginning in the conversion state  
49747 described by the object pointed to by *ps*. If *dst* is not a null pointer, the converted characters  
49748 shall then be stored into the array pointed to by *dst*. Conversion continues up to and including a  
49749 terminating null wide character, which shall also be stored. Conversion shall stop earlier in the  
49750 following cases:

- 49751       • When a code is reached that does not correspond to a valid character
- 49752       • When the next character would exceed the limit of *len* total bytes to be stored in the array  
49753 pointed to by *dst* (and *dst* is not a null pointer)

49754       Each conversion shall take place as if by a call to the *wcrtomb()* function.

49755       If *dst* is not a null pointer, the pointer object pointed to by *src* shall be assigned either a null  
49756 pointer (if conversion stopped due to reaching a terminating null wide character) or the address  
49757 just past the last wide character converted (if any). If conversion stopped due to reaching a  
49758 terminating null wide character, the resulting state described shall be the initial conversion state.

49759       If *ps* is a null pointer, the *wcsrtoombs()* function shall use its own internal **mbstate\_t** object, which  
49760 is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate\_t** object  
49761 pointed to by *ps* shall be used to completely describe the current conversion state of the  
49762 associated character sequence. The implementation shall behave as if no function defined in this  
49763 volume of IEEE Std 1003.1-2001 calls *wcsrtoombs()*.

49764 CX       If the application uses any of the `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS`  
49765 functions, the application shall ensure that the *wcsrtoombs()* function is called with a non-NULL  
49766 *ps* argument.

49767       The behavior of this function shall be affected by the *LC\_CTYPE* category of the current locale.

## 49768 RETURN VALUE

49769       If conversion stops because a code is reached that does not correspond to a valid character, an  
49770 encoding error occurs. In this case, the *wcsrtoombs()* function shall store the value of the macro  
49771 `[EILSEQ]` in *errno* and return **(size\_t)–1**; the conversion state is undefined. Otherwise, it shall  
49772 return the number of bytes in the resulting character sequence, not including the terminating  
49773 null (if any).

## 49774 ERRORS

49775       The *wcsrtoombs()* function may fail if:

- 49776 CX       **[EINVAL]**       *ps* points to an object that contains an invalid conversion state.
- 49777       **[EILSEQ]**       A wide-character code does not correspond to a valid character.

49778 **EXAMPLES**

49779 None.

49780 **APPLICATION USAGE**

49781 None.

49782 **RATIONALE**

49783 None.

49784 **FUTURE DIRECTIONS**

49785 None.

49786 **SEE ALSO**49787 *mbsinit()*, *wcrtomb()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>49788 **CHANGE HISTORY**49789 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
49790 (E).49791 **Issue 6**

49792 In the DESCRIPTION, a note on using this function in a threaded application is added.

49793 Extensions beyond the ISO C standard are marked.

49794 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

49795 The *wcsrtombs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49796 **NAME**

49797 wcsspn — get the length of a wide substring

49798 **SYNOPSIS**

49799 #include &lt;wchar.h&gt;

49800 size\_t wcsspn(const wchar\_t \*ws1, const wchar\_t \*ws2);

49801 **DESCRIPTION**

49802 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
49803 conflict between the requirements described here and the ISO C standard is unintentional. This  
49804 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49805 The *wcsspn()* function shall compute the length (in wide characters) of the maximum initial  
49806 segment of the wide-character string pointed to by *ws1* which consists entirely of wide-character  
49807 codes from the wide-character string pointed to by *ws2*.

49808 **RETURN VALUE**

49809 The *wcsspn()* function shall return the length of the initial substring of *ws1*; no return value is  
49810 reserved to indicate an error.

49811 **ERRORS**

49812 No errors are defined.

49813 **EXAMPLES**

49814 None.

49815 **APPLICATION USAGE**

49816 None.

49817 **RATIONALE**

49818 None.

49819 **FUTURE DIRECTIONS**

49820 None.

49821 **SEE ALSO**49822 *wcscspn()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>49823 **CHANGE HISTORY**

49824 First released in Issue 4. Derived from the MSE working draft.

49825 **Issue 5**

49826 The RETURN VALUE section is updated to indicate that *wcsspn()* returns the length of *ws1*  
49827 rather than *ws1* itself.

49828 **NAME**49829 `wcsstr` — find a wide-character substring49830 **SYNOPSIS**49831 `#include <wchar.h>`49832 `wchar_t *wcsstr(const wchar_t *restrict ws1,`  
49833 `const wchar_t *restrict ws2);`49834 **DESCRIPTION**49835 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
49836 conflict between the requirements described here and the ISO C standard is unintentional. This  
49837 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.49838 The `wcsstr()` function shall locate the first occurrence in the wide-character string pointed to by  
49839 `ws1` of the sequence of wide characters (excluding the terminating null wide character) in the  
49840 wide-character string pointed to by `ws2`.49841 **RETURN VALUE**49842 Upon successful completion, `wcsstr()` shall return a pointer to the located wide-character string,  
49843 or a null pointer if the wide-character string is not found.49844 If `ws2` points to a wide-character string with zero length, the function shall return `ws1`.49845 **ERRORS**

49846 No errors are defined.

49847 **EXAMPLES**

49848 None.

49849 **APPLICATION USAGE**

49850 None.

49851 **RATIONALE**

49852 None.

49853 **FUTURE DIRECTIONS**

49854 None.

49855 **SEE ALSO**49856 `wcschr()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<wchar.h>`49857 **CHANGE HISTORY**49858 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
49859 (E).49860 **Issue 6**49861 The `wcsstr()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

## 49862 NAME

49863 wcstod, wcstof, wcstold — convert a wide-character string to a double-precision number

## 49864 SYNOPSIS

49865 #include <wchar.h>

49866 double wcstod(const wchar\_t \*restrict nptr, wchar\_t \*\*restrict endptr);

49867 float wcstof(const wchar\_t \*restrict nptr, wchar\_t \*\*restrict endptr);

49868 long double wcstold(const wchar\_t \*restrict nptr,

49869 wchar\_t \*\*restrict endptr);

## 49870 DESCRIPTION

49871 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
49872 conflict between the requirements described here and the ISO C standard is unintentional. This  
49873 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

49874 These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to  
49875 **double**, **float**, and **long double** representation, respectively. First, they shall decompose the  
49876 input wide-character string into three parts:

- 49877 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by  
49878 *iswspace()*)
- 49879 2. A subject sequence interpreted as a floating-point constant or representing infinity or NaN
- 49880 3. A final wide-character string of one or more unrecognized wide-character codes, including  
49881 the terminating null wide-character code of the input wide-character string

49882 Then they shall attempt to convert the subject sequence to a floating-point number, and return  
49883 the result.

49884 The expected form of the subject sequence is an optional plus or minus sign, then one of the  
49885 following:

- 49886 • A non-empty sequence of decimal digits optionally containing a radix character, then an  
49887 optional exponent part
- 49888 • A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix  
49889 character, then an optional binary exponent part
- 49890 • One of INF or INFINITY, or any other wide string equivalent except for case
- 49891 • One of NAN or NAN(*n-wchar-sequence<sub>opt</sub>*), or any other wide string ignoring case in the NAN  
49892 part, where:

49893 n-wchar-sequence:

49894 digit

49895 nondigit

49896 n-wchar-sequence digit

49897 n-wchar-sequence nondigit

49898 The subject sequence is defined as the longest initial subsequence of the input wide string,  
49899 starting with the first non-white-space wide character, that is of the expected form. The subject  
49900 sequence contains no wide characters if the input wide string is not of the expected form.

49901 If the subject sequence has the expected form for a floating-point number, the sequence of wide  
49902 characters starting with the first digit or the radix character (whichever occurs first) shall be  
49903 interpreted as a floating constant according to the rules of the C language, except that the radix  
49904 character shall be used in place of a period, and that if neither an exponent part nor a radix  
49905 character appears in a decimal floating-point number, or if a binary exponent part does not

49906 appear in a hexadecimal floating-point number, an exponent part of the appropriate type with  
 49907 value zero shall be assumed to follow the last digit in the string. If the subject sequence begins  
 49908 with a minus sign, the sequence shall be interpreted as negated. A wide-character sequence INF  
 49909 or INFINITY shall be interpreted as an infinity, if representable in the return type, else as if it  
 49910 were a floating constant that is too large for the range of the return type. A wide-character  
 49911 sequence NAN or NAN(*n-wchar-sequence<sub>opt</sub>*) shall be interpreted as a quiet NaN, if supported in  
 49912 the return type, else as if it were a subject sequence part that does not have the expected form;  
 49913 the meaning of the *n-wchar* sequences is implementation-defined. A pointer to the final wide  
 49914 string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

49915 If the subject sequence has the hexadecimal form and FLT\_RADIX is a power of 2, the  
 49916 conversion shall be rounded in an implementation-defined manner.

49917 CX The radix character shall be as defined in the program's locale (category *LC\_NUMERIC*). In the  
 49918 POSIX locale, or in a locale where the radix character is not defined, the radix character shall  
 49919 default to a period ('.').

49920 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be  
 49921 accepted.

49922 If the subject sequence is empty or does not have the expected form, no conversion shall be  
 49923 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that  
 49924 *endptr* is not a null pointer.

49925 CX The *wcstod()* function shall not change the setting of *errno* if successful.

49926 Since 0 is returned on error and is also a valid return on success, an application wishing to check  
 49927 for error situations should set *errno* to 0, then call *wcstod()*, *wcstof()*, or *wcstold()*, then check  
 49928 *errno*.

#### 49929 RETURN VALUE

49930 Upon successful completion, these functions shall return the converted value. If no conversion  
 49931 CX could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

49932 If the correct value is outside the range of representable values, ±HUGE\_VAL, ±HUGE\_VALF, or |  
 49933 ±HUGE\_VALL shall be returned (according to the sign of the value), and *errno* shall be set to |  
 49934 [ERANGE].

49935 If the correct value would cause underflow, a value whose magnitude is no greater than the  
 49936 smallest normalized positive number in the return type shall be returned and *errno* set to  
 49937 [ERANGE].

#### 49938 ERRORS

49939 The *wcstod()* function shall fail if:

49940 [ERANGE] The value to be returned would cause overflow or underflow.

49941 The *wcstod()* function may fail if:

49942 CX [EINVAL] No conversion could be performed.

49943 **EXAMPLES**

49944 None.

49945 **APPLICATION USAGE**

49946 If the subject sequence has the hexadecimal form and FLT\_RADIX is not a power of 2, and the  
 49947 result is not exactly representable, the result should be one of the two numbers in the  
 49948 appropriate internal format that are adjacent to the hexadecimal floating source value, with the  
 49949 extra stipulation that the error should have a correct sign for the current rounding direction.

49950 If the subject sequence has the decimal form and at most DECIMAL\_DIG (defined in `<float.h>`)  
 49951 significant digits, the result should be correctly rounded. If the subject sequence *D* has the  
 49952 decimal form and more than DECIMAL\_DIG significant digits, consider the two bounding,  
 49953 adjacent decimal strings *L* and *U*, both having DECIMAL\_DIG significant digits, such that the  
 49954 values of *L*, *D*, and *U* satisfy " $L \leq D \leq U$ ". The result should be one of the (equal or  
 49955 adjacent) values that would be obtained by correctly rounding *L* and *U* according to the current  
 49956 rounding direction, with the extra stipulation that the error with respect to *D* should have a  
 49957 correct sign for the current rounding direction.

49958 **RATIONALE**

49959 None.

49960 **FUTURE DIRECTIONS**

49961 None.

49962 **SEE ALSO**

49963 *iswspace()*, *localeconv()*, *scanf()*, *setlocale()*, *wcstol()*, the Base Definitions volume of  
 49964 IEEE Std 1003.1-2001, Chapter 7, Locale, `<float.h>`, `<wchar.h>`

49965 **CHANGE HISTORY**

49966 First released in Issue 4. Derived from the MSE working draft.

49967 **Issue 5**49968 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.49969 **Issue 6**

49970 Extensions beyond the ISO C standard are marked.

49971 The following new requirements on POSIX implementations derive from alignment with the  
 49972 Single UNIX Specification:

- 49973 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
 49974 added if no conversion could be performed.

49975 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 49976 • The *wcstod()* prototype is updated.
- 49977 • The *wcstof()* and *wcstold()* functions are added.
- 49978 • If the correct value for *wcstod()* would cause underflow, the return value changed from 0 (as  
 49979 specified in Issue 5) to the smallest normalized positive number.
- 49980 • The DESCRIPTION, RETURN VALUE, and APPLICATION USAGE sections are extensively  
 49981 updated.

49982 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

49983 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/66 is applied, correcting the second  
 49984 paragraph in the RETURN VALUE section.

49985 **NAME**

49986           wcstoimax, wcstoumax — convert a wide-character string to an integer type

49987 **SYNOPSIS**

49988           #include &lt;stddef.h&gt;

49989           #include &lt;inttypes.h&gt;

49990           intmax\_t wcstoimax(const wchar\_t \*restrict nptr,

49991                            wchar\_t \*\*restrict endptr, int base);

49992           uintmax\_t wcstoumax(const wchar\_t \*restrict nptr,

49993                            wchar\_t \*\*restrict endptr, int base);

49994 **DESCRIPTION**49995 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
49996 conflict between the requirements described here and the ISO C standard is unintentional. This  
49997 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.49998       These functions shall be equivalent to the *wcstol()*, *wcstoll()*, *wcstoul()*, and *wcstoull()* functions,  
49999 respectively, except that the initial portion of the wide string shall be converted to **intmax\_t** and  
50000 **uintmax\_t** representation, respectively.50001 **RETURN VALUE**

50002       These functions shall return the converted value, if any.

50003       If no conversion could be performed, zero shall be returned. If the correct value is outside the  
50004 range of representable values, {INTMAX\_MAX}, {INTMAX\_MIN}, or {UINTMAX\_MAX} shall  
50005 be returned (according to the return type and sign of the value, if any), and *errno* shall be set to  
50006 [ERANGE].50007 **ERRORS**

50008       These functions shall fail if:

50009       [EINVAL]       The value of *base* is not supported.

50010       [ERANGE]      The value to be returned is not representable.

50011       These functions may fail if:

50012       [EINVAL]      No conversion could be performed.

50013 **EXAMPLES**

50014       None.

50015 **APPLICATION USAGE**

50016       None.

50017 **RATIONALE**

50018       None.

50019 **FUTURE DIRECTIONS**

50020       None.

50021 **SEE ALSO**50022       *wcstol()*, *wcstoul()*, the Base Definitions volume of IEEE Std 1003.1-2001, <inttypes.h>,  
50023 <stddef.h>50024 **CHANGE HISTORY**

50025       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

50026 **NAME**

50027           wcstok — split a wide-character string into tokens

50028 **SYNOPSIS**

50029           #include &lt;wchar.h&gt;

50030           wchar\_t \*wcstok(wchar\_t \*restrict ws1, const wchar\_t \*restrict ws2,  
50031                            wchar\_t \*\*restrict ptr);50032 **DESCRIPTION**50033 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
50034       conflict between the requirements described here and the ISO C standard is unintentional. This  
50035       volume of IEEE Std 1003.1-2001 defers to the ISO C standard.50036       A sequence of calls to *wcstok()* shall break the wide-character string pointed to by *ws1* into a  
50037       sequence of tokens, each of which shall be delimited by a wide-character code from the wide-  
50038       character string pointed to by *ws2*. The *ptr* argument points to a caller-provided **wchar\_t** pointer  
50039       into which the *wcstok()* function shall store information necessary for it to continue scanning the  
50040       same wide-character string.50041       The first call in the sequence has *ws1* as its first argument, and is followed by calls with a null  
50042       pointer as their first argument. The separator string pointed to by *ws2* may be different from call  
50043       to call.50044       The first call in the sequence shall search the wide-character string pointed to by *ws1* for the first  
50045       wide-character code that is *not* contained in the current separator string pointed to by *ws2*. If no  
50046       such wide-character code is found, then there are no tokens in the wide-character string pointed  
50047       to by *ws1* and *wcstok()* shall return a null pointer. If such a wide-character code is found, it shall  
50048       be the start of the first token.50049       The *wcstok()* function shall then search from there for a wide-character code that *is* contained in  
50050       the current separator string. If no such wide-character code is found, the current token extends  
50051       to the end of the wide-character string pointed to by *ws1*, and subsequent searches for a token  
50052       shall return a null pointer. If such a wide-character code is found, it shall be overwritten by a  
50053       null wide character, which terminates the current token. The *wcstok()* function shall save a  
50054       pointer to the following wide-character code, from which the next search for a token shall start.50055       Each subsequent call, with a null pointer as the value of the first argument, shall start searching  
50056       from the saved pointer and behave as described above.50057       The implementation shall behave as if no function calls *wcstok()*.50058 **RETURN VALUE**50059       Upon successful completion, the *wcstok()* function shall return a pointer to the first wide-  
50060       character code of a token. Otherwise, if there is no token, *wcstok()* shall return a null pointer.50061 **ERRORS**

50062       No errors are defined.

50063 **EXAMPLES**

50064 None.

50065 **APPLICATION USAGE**

50066 None.

50067 **RATIONALE**

50068 None.

50069 **FUTURE DIRECTIONS**

50070 None.

50071 **SEE ALSO**

50072 The Base Definitions volume of IEEE Std 1003.1-2001, &lt;wchar.h&gt;

50073 **CHANGE HISTORY**

50074 First released in Issue 4.

50075 **Issue 5**50076 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, a third argument is  
50077 added to the definition of *wcstok()* in the SYNOPSIS.50078 **Issue 6**50079 The *wcstok()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

## 50080 NAME

50081 `wcstol`, `wcstoll` — convert a wide-character string to a long integer

## 50082 SYNOPSIS

50083 `#include <wchar.h>`

50084 `long wcstol(const wchar_t *restrict nptr, wchar_t **restrict endpnr,`  
 50085 `int base);`

50086 `long long wcstoll(const wchar_t *restrict nptr,`  
 50087 `wchar_t **restrict endpnr, int base);`

## 50088 DESCRIPTION

50089 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 50090 conflict between the requirements described here and the ISO C standard is unintentional. This  
 50091 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50092 These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to  
 50093 **long**, **long long**, **unsigned long**, and **unsigned long long** representation, respectively. First, they  
 50094 shall decompose the input string into three parts:

- 50095 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by  
 50096 `iswspace()`)
- 50097 2. A subject sequence interpreted as an integer represented in some radix determined by the  
 50098 value of *base*
- 50099 3. A final wide-character string of one or more unrecognized wide-character codes, including  
 50100 the terminating null wide-character code of the input wide-character string

50101 Then they shall attempt to convert the subject sequence to an integer, and return the result.

50102 If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant,  
 50103 or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal  
 50104 constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal  
 50105 constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7'  
 50106 only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the  
 50107 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

50108 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence  
 50109 of letters and digits representing an integer with the radix specified by *base*, optionally preceded  
 50110 by a '+' or '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z'  
 50111 (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less  
 50112 than that of *base* shall be permitted. If the value of *base* is 16, the wide-character code  
 50113 representations of 0x or 0X may optionally precede the sequence of letters and digits, following  
 50114 the sign if present.

50115 The subject sequence is defined as the longest initial subsequence of the input wide-character  
 50116 string, starting with the first non-white-space wide-character code that is of the expected form.  
 50117 The subject sequence contains no wide-character codes if the input wide-character string is  
 50118 empty or consists entirely of white-space wide-character code, or if the first non-white-space  
 50119 wide-character code is other than a sign or a permissible letter or digit.

50120 If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes  
 50121 starting with the first digit shall be interpreted as an integer constant. If the subject sequence has  
 50122 the expected form and the value of *base* is between 2 and 36, it shall be used as the base for  
 50123 conversion, ascribing to each letter its value as given above. If the subject sequence begins with a  
 50124 minus sign, the value resulting from the conversion shall be negated. A pointer to the final  
 50125 wide-character string shall be stored in the object pointed to by *endpnr*, provided that *endpnr* is

- 50126 not a null pointer.
- 50127 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be  
50128 accepted.
- 50129 If the subject sequence is empty or does not have the expected form, no conversion shall be  
50130 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that  
50131 *endptr* is not a null pointer.
- 50132 CX These functions shall not change the setting of *errno* if successful.
- 50133 Since 0, {LONG\_MIN} or {LLONG\_MIN} and {LONG\_MAX} or {LLONG\_MAX} are returned on  
50134 error and are also valid returns on success, an application wishing to check for error situations  
50135 should set *errno* to 0, then call *wcstol()* or *wcstoll()*, then check *errno*.
- 50136 **RETURN VALUE**
- 50137 Upon successful completion, these functions shall return the converted value, if any. If no  
50138 CX conversion could be performed, 0 shall be returned and *errno* may be set to indicate the error. If  
50139 the correct value is outside the range of representable values, {LONG\_MIN}, {LONG\_MAX},  
50140 {LLONG\_MIN}, or {LLONG\_MAX} shall be returned (according to the sign of the value), and  
50141 *errno* set to [ERANGE].
- 50142 **ERRORS**
- 50143 These functions shall fail if:
- 50144 CX [EINVAL] The value of *base* is not supported.
- 50145 [ERANGE] The value to be returned is not representable.
- 50146 These functions may fail if:
- 50147 CX [EINVAL] No conversion could be performed.
- 50148 **EXAMPLES**
- 50149 None.
- 50150 **APPLICATION USAGE**
- 50151 None.
- 50152 **RATIONALE**
- 50153 None.
- 50154 **FUTURE DIRECTIONS**
- 50155 None.
- 50156 **SEE ALSO**
- 50157 *iswalpha()*, *scanf()*, *wcstod()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>
- 50158 **CHANGE HISTORY**
- 50159 First released in Issue 4. Derived from the MSE working draft.
- 50160 **Issue 5**
- 50161 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.
- 50162 **Issue 6**
- 50163 Extensions beyond the ISO C standard are marked.
- 50164 The following new requirements on POSIX implementations derive from alignment with the  
50165 Single UNIX Specification:
- 50166 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
50167 added if no conversion could be performed.

- 50168        The following changes are made for alignment with the ISO/IEC 9899:1999 standard:
- 50169        • The *wcstol()* prototype is updated.
- 50170        • The *wcstoll()* function is added.

50171 **NAME**

50172           wcstold — convert a wide-character string to a double-precision number

50173 **SYNOPSIS**

50174           #include <wchar.h>

50175           long double wcstold(const wchar\_t \*restrict *nptr*,

50176                    wchar\_t \*\*restrict *endptr*);

50177 **DESCRIPTION**

50178           Refer to *wcstod*().

50179 **NAME**

50180           wcstoll — convert a wide-character string to a long integer

50181 **SYNOPSIS**

50182           #include <wchar.h>

50183           long long wcstoll(const wchar\_t \*restrict *nptr*,

50184                        wchar\_t \*\*restrict *endptr*, int *base*);

50185 **DESCRIPTION**

50186           Refer to *wcstol*().

50187 **NAME**

50188           wcstombs — convert a wide-character string to a character string

50189 **SYNOPSIS**

50190           #include &lt;stdlib.h&gt;

50191           size\_t wcstombs(char \*restrict *s*, const wchar\_t \*restrict *pwcs*,  
50192                           size\_t *n*);50193 **DESCRIPTION**50194 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
50195 conflict between the requirements described here and the ISO C standard is unintentional. This  
50196 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.50197       The *wcstombs()* function shall convert the sequence of wide-character codes that are in the array  
50198 pointed to by *pwcs* into a sequence of characters that begins in the initial shift state and store  
50199 these characters into the array pointed to by *s*, stopping if a character would exceed the limit of *n*  
50200 total bytes or if a null byte is stored. Each wide-character code shall be converted as if by a call to  
50201 *wctomb()*, except that the shift state of *wctomb()* shall not be affected.50202       The behavior of this function shall be affected by the *LC\_CTYPE* category of the current locale.50203       No more than *n* bytes shall be modified in the array pointed to by *s*. If copying takes place  
50204 **CX**       between objects that overlap, the behavior is undefined. If *s* is a null pointer, *wcstombs()* shall  
50205 return the length required to convert the entire array regardless of the value of *n*, but no values  
50206 are stored.50207       The *wcstombs()* function need not be reentrant. A function that is not required to be reentrant is  
50208 not required to be thread-safe.50209 **RETURN VALUE**50210       If a wide-character code is encountered that does not correspond to a valid character (of one or  
50211 more bytes each), *wcstombs()* shall return (**size\_t**)−1. Otherwise, *wcstombs()* shall return the  
50212 number of bytes stored in the character array, not including any terminating null byte. The array  
50213 shall not be null-terminated if the value returned is *n*.50214 **ERRORS**50215       The *wcstombs()* function may fail if:50216 **CX**       [EILSEQ]       A wide-character code does not correspond to a valid character.50217 **EXAMPLES**

50218       None.

50219 **APPLICATION USAGE**

50220       None.

50221 **RATIONALE**

50222       None.

50223 **FUTURE DIRECTIONS**

50224       None.

50225 **SEE ALSO**50226       *mblen()*, *mbtowc()*, *mbstowcs()*, *wctomb()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
50227 <stdlib.h>

50228 **CHANGE HISTORY**

50229 First released in Issue 4. Derived from the ISO C standard.

50230 **Issue 6**

50231 The following new requirements on POSIX implementations derive from alignment with the  
50232 Single UNIX Specification:

- 50233 • The DESCRIPTION states the effect of when *s* is a null pointer.
- 50234 • The [EILSEQ] error condition is added.

50235 The *wcstombs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

## 50236 NAME

50237 wcstoul, wcstoull — convert a wide-character string to an unsigned long

## 50238 SYNOPSIS

50239 #include &lt;wchar.h&gt;

50240 unsigned long wcstoul(const wchar\_t \*restrict *nptr*,  
50241 wchar\_t \*\*restrict *endptr*, int *base*);50242 unsigned long long wcstoull(const wchar\_t \*restrict *nptr*,  
50243 wchar\_t \*\*restrict *endptr*, int *base*);

## 50244 DESCRIPTION

50245 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50246 conflict between the requirements described here and the ISO C standard is unintentional. This  
50247 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.50248 The *wcstoul()* and *wcstoull()* functions shall convert the initial portion of the wide-character  
50249 string pointed to by *nptr* to **unsigned long** and **unsigned long long** representation, respectively.  
50250 First, they shall decompose the input wide-character string into three parts:

- 50251 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by  
50252 *iswspace()*)
- 50253 2. A subject sequence interpreted as an integer represented in some radix determined by the  
50254 value of *base*
- 50255 3. A final wide-character string of one or more unrecognized wide-character codes, including  
50256 the terminating null wide-character code of the input wide-character string

50257 Then they shall attempt to convert the subject sequence to an unsigned integer, and return the  
50258 result.50259 If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant,  
50260 or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal  
50261 constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal  
50262 constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7'  
50263 only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the  
50264 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.50265 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence  
50266 of letters and digits representing an integer with the radix specified by *base*, optionally preceded  
50267 by a '+' or '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z'  
50268 (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less  
50269 than that of *base* shall be permitted. If the value of *base* is 16, the wide-character codes 0x or 0X  
50270 may optionally precede the sequence of letters and digits, following the sign if present.50271 The subject sequence is defined as the longest initial subsequence of the input wide-character  
50272 string, starting with the first wide-character code that is not white space and is of the expected  
50273 form. The subject sequence contains no wide-character codes if the input wide-character string is  
50274 empty or consists entirely of white-space wide-character codes, or if the first wide-character  
50275 code that is not white space is other than a sign or a permissible letter or digit.50276 If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes  
50277 starting with the first digit shall be interpreted as an integer constant. If the subject sequence has  
50278 the expected form and the value of *base* is between 2 and 36, it shall be used as the base for  
50279 conversion, ascribing to each letter its value as given above. If the subject sequence begins with a  
50280 minus sign, the value resulting from the conversion shall be negated. A pointer to the final  
50281 wide-character string shall be stored in the object pointed to by *endptr*, provided that *endptr* is

- 50282 not a null pointer.
- 50283 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be  
50284 accepted.
- 50285 If the subject sequence is empty or does not have the expected form, no conversion shall be  
50286 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that  
50287 *endptr* is not a null pointer.
- 50288 CX The *wcstoul()* function shall not change the setting of *errno* if successful.
- 50289 Since 0, {ULONG\_MAX}, and {ULLONG\_MAX} are returned on error and 0 is also a valid return  
50290 on success, an application wishing to check for error situations should set *errno* to 0, then call  
50291 *wcstoul()* or *wcstoull()*, then check *errno*.
- 50292 **RETURN VALUE**
- 50293 Upon successful completion, the *wcstoul()* and *wcstoull()* functions shall return the converted  
50294 CX value, if any. If no conversion could be performed, 0 shall be returned and *errno* may be set to  
50295 indicate the error. If the correct value is outside the range of representable values,  
50296 {ULONG\_MAX} or {ULLONG\_MAX} respectively shall be returned and *errno* set to [ERANGE].
- 50297 **ERRORS**
- 50298 These functions shall fail if:
- 50299 CX [EINVAL] The value of *base* is not supported.
- 50300 [ERANGE] The value to be returned is not representable.
- 50301 These functions may fail if:
- 50302 CX [EINVAL] No conversion could be performed.
- 50303 **EXAMPLES**
- 50304 None.
- 50305 **APPLICATION USAGE**
- 50306 None.
- 50307 **RATIONALE**
- 50308 None.
- 50309 **FUTURE DIRECTIONS**
- 50310 None.
- 50311 **SEE ALSO**
- 50312 *iswalph()*, *scanf()*, *wcstod()*, *wcstol()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
50313 <wchar.h>
- 50314 **CHANGE HISTORY**
- 50315 First released in Issue 4. Derived from the MSE working draft.
- 50316 **Issue 5**
- 50317 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.
- 50318 **Issue 6**
- 50319 Extensions beyond the ISO C standard are marked.
- 50320 The following new requirements on POSIX implementations derive from alignment with the  
50321 Single UNIX Specification:
- 50322
- The [EINVAL] error condition is added for when the value of *base* is not supported.

50323            In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
50324            added if no conversion could be performed.

50325            The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 50326            • The *wcstoul()* prototype is updated.
- 50327            • The *wcstoull()* function is added.

50328 **NAME**

50329           wcstoumax — convert a wide-character string to an integer type

50330 **SYNOPSIS**

50331           #include &lt;stddef.h&gt;

50332           #include &lt;inttypes.h&gt;

50333           uintmax\_t wcstoumax(const wchar\_t \*restrict *nptr*,50334                            wchar\_t \*\*restrict *endptr*, int *base*);50335 **DESCRIPTION**50336           Refer to *wcstoimax()*.

50337 **NAME**50338           wcswcs — find a wide substring (**LEGACY**)50339 **SYNOPSIS**

50340 XSI       #include &lt;wchar.h&gt;

50341           wchar\_t \*wcswcs(const wchar\_t \*ws1, const wchar\_t \*ws2);

50342

50343 **DESCRIPTION**

50344           The `wcswcs()` function shall locate the first occurrence in the wide-character string pointed to by  
50345           `ws1` of the sequence of wide-character codes (excluding the terminating null wide-character  
50346           code) in the wide-character string pointed to by `ws2`.

50347 **RETURN VALUE**

50348           Upon successful completion, `wcswcs()` shall return a pointer to the located wide-character string  
50349           or a null pointer if the wide-character string is not found.

50350           If `ws2` points to a wide-character string with zero length, the function shall return `ws1`.

50351 **ERRORS**

50352           No errors are defined.

50353 **EXAMPLES**

50354           None.

50355 **APPLICATION USAGE**

50356           This function was not included in the final ISO/IEC 9899:1990/Amendment 1:1995 (E).

50357           Application developers are strongly encouraged to use the `wcsstr()` function instead.50358 **RATIONALE**

50359           None.

50360 **FUTURE DIRECTIONS**

50361           This function may be withdrawn in a future version.

50362 **SEE ALSO**50363           `wcschr()`, `wcsstr()`, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>50364 **CHANGE HISTORY**

50365           First released in Issue 4. Derived from the MSE working draft.

50366 **Issue 5**

50367           Marked EX.

50368 **Issue 6**

50369           This function is marked LEGACY.

50370 **NAME**

50371       wcswidth — number of column positions of a wide-character string

50372 **SYNOPSIS**

```
50373 xSI #include <wchar.h>
```

```
50374 int wcswidth(const wchar_t *pwcs, size_t n);
```

50375

50376 **DESCRIPTION**

50377       The *wcswidth()* function shall determine the number of column positions required for *n* wide-character codes (or fewer than *n* wide-character codes if a null wide-character code is encountered before *n* wide-character codes are exhausted) in the string pointed to by *pwcs*.

50380 **RETURN VALUE**

50381       The *wcswidth()* function either shall return 0 (if *pwcs* points to a null wide-character code), or return the number of column positions to be occupied by the wide-character string pointed to by *pwcs*, or return -1 (if any of the first *n* wide-character codes in the wide-character string pointed to by *pwcs* is not a printable wide-character code).

50385 **ERRORS**

50386       No errors are defined.

50387 **EXAMPLES**

50388       None.

50389 **APPLICATION USAGE**

50390       This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the return value for a non-printable wide character is not specified.

50392 **RATIONALE**

50393       None.

50394 **FUTURE DIRECTIONS**

50395       None.

50396 **SEE ALSO**

50397       *wcwidth()*, the Base Definitions volume of IEEE Std 1003.1-2001, Section 3.103, Column Position,  
50398       <**wchar.h**>

50399 **CHANGE HISTORY**

50400       First released in Issue 4. Derived from the MSE working draft.

50401 **Issue 6**

50402       The Open Group Corrigendum U021/11 is applied. The function is marked as an extension.

50403 **NAME**50404        `wcsxfrm` — wide-character string transformation50405 **SYNOPSIS**50406        `#include <wchar.h>`50407        `size_t wcsxfrm(wchar_t *restrict ws1, const wchar_t *restrict ws2,`  
50408            `size_t n);`50409 **DESCRIPTION**50410 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
50411        conflict between the requirements described here and the ISO C standard is unintentional. This  
50412        volume of IEEE Std 1003.1-2001 defers to the ISO C standard.50413        The `wcsxfrm()` function shall transform the wide-character string pointed to by `ws2` and place the  
50414        resulting wide-character string into the array pointed to by `ws1`. The transformation shall be  
50415        such that if `wscmp()` is applied to two transformed wide strings, it shall return a value greater  
50416        than, equal to, or less than 0, corresponding to the result of `wscoll()` applied to the same two  
50417        original wide-character strings. No more than `n` wide-character codes shall be placed into the  
50418        resulting array pointed to by `ws1`, including the terminating null wide-character code. If `n` is 0,  
50419        `ws1` is permitted to be a null pointer. If copying takes place between objects that overlap, the  
50420        behavior is undefined.50421 **CX**        The `wcsxfrm()` function shall not change the setting of `errno` if successful.50422        Since no return value is reserved to indicate an error, an application wishing to check for error  
50423        situations should set `errno` to 0, then call `wcsxfrm()`, then check `errno`.50424 **RETURN VALUE**50425        The `wcsxfrm()` function shall return the length of the transformed wide-character string (not  
50426        including the terminating null wide-character code). If the value returned is `n` or more, the  
50427        contents of the array pointed to by `ws1` are unspecified.50428 **CX**        On error, the `wcsxfrm()` function may set `errno`, but no return value is reserved to indicate an  
50429        error.50430 **ERRORS**50431        The `wcsxfrm()` function may fail if:50432 **CX**        [EINVAL]        The wide-character string pointed to by `ws2` contains wide-character codes  
50433        outside the domain of the collating sequence.50434 **EXAMPLES**

50435        None.

50436 **APPLICATION USAGE**50437        The transformation function is such that two transformed wide-character strings can be ordered  
50438        by `wscmp()` as appropriate to collating sequence information in the program's locale (category  
50439        `LC_COLLATE`).50440        The fact that when `n` is 0 `ws1` is permitted to be a null pointer is useful to determine the size of  
50441        the `ws1` array prior to making the transformation.50442 **RATIONALE**

50443        None.

50444 **FUTURE DIRECTIONS**

50445 None.

50446 **SEE ALSO**50447 *wscmp()*, *wscoll()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>50448 **CHANGE HISTORY**

50449 First released in Issue 4. Derived from the MSE working draft.

50450 **Issue 5**

50451 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

50452 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.50453 **Issue 6**

50454 In previous versions, this function was required to return -1 on error.

50455 Extensions beyond the ISO C standard are marked.

50456 The following new requirements on POSIX implementations derive from alignment with the  
50457 Single UNIX Specification:

- 50458
- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
- 
- 50459 added if no conversion could be performed.

50460 The *wcsxfrm()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

50461 **NAME**

50462 wctob — wide-character to single-byte conversion

50463 **SYNOPSIS**

50464 #include &lt;stdio.h&gt;

50465 #include &lt;wchar.h&gt;

50466 int wctob(wint\_t c);

50467 **DESCRIPTION**

50468 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50469 conflict between the requirements described here and the ISO C standard is unintentional. This  
50470 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50471 The *wctob()* function shall determine whether *c* corresponds to a member of the extended  
50472 character set whose character representation is a single byte when in the initial shift state.

50473 The behavior of this function shall be affected by the *LC\_CTYPE* category of the current locale.

50474 **RETURN VALUE**

50475 The *wctob()* function shall return EOF if *c* does not correspond to a character with length one in  
50476 the initial shift state. Otherwise, it shall return the single-byte representation of that character as  
50477 an **unsigned char** converted to **int**.

50478 **ERRORS**

50479 No errors are defined.

50480 **EXAMPLES**

50481 None.

50482 **APPLICATION USAGE**

50483 None.

50484 **RATIONALE**

50485 None.

50486 **FUTURE DIRECTIONS**

50487 None.

50488 **SEE ALSO**50489 *btowc()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>50490 **CHANGE HISTORY**

50491 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
50492 (E).

50493 **NAME**

50494 wctomb — convert a wide-character code to a character

50495 **SYNOPSIS**

50496 #include &lt;stdlib.h&gt;

50497 int wctomb(char \*s, wchar\_t wchar);

50498 **DESCRIPTION**

50499 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50500 conflict between the requirements described here and the ISO C standard is unintentional. This  
50501 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50502 The *wctomb()* function shall determine the number of bytes needed to represent the character  
50503 corresponding to the wide-character code whose value is *wchar* (including any change in the  
50504 shift state). It shall store the character representation (possibly multiple bytes and any special  
50505 bytes to change shift state) in the array object pointed to by *s* (if *s* is not a null pointer). At most  
50506 {MB\_CUR\_MAX} bytes shall be stored. If *wchar* is 0, a null byte shall be stored, preceded by any  
50507 shift sequence needed to restore the initial shift state, and *wctomb()* shall be left in the initial shift  
50508 state.

50509 cx The behavior of this function is affected by the *LC\_CTYPE* category of the current locale. For a  
50510 state-dependent encoding, this function shall be placed into its initial state by a call for which its  
50511 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null  
50512 pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a  
50513 null pointer shall cause this function to return a non-zero value if encodings have state  
50514 dependency, and 0 otherwise. Changing the *LC\_CTYPE* category causes the shift state of this  
50515 function to be unspecified.

50516 The *wctomb()* function need not be reentrant. A function that is not required to be reentrant is  
50517 not required to be thread-safe.

50518 The implementation shall behave as if no function defined in this volume of  
50519 IEEE Std 1003.1-2001 calls *wctomb()*.

50520 **RETURN VALUE**

50521 If *s* is a null pointer, *wctomb()* shall return a non-zero or 0 value, if character encodings,  
50522 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *wctomb()*  
50523 shall return -1 if the value of *wchar* does not correspond to a valid character, or return the  
50524 number of bytes that constitute the character corresponding to the value of *wchar*.

50525 In no case shall the value returned be greater than the value of the {MB\_CUR\_MAX} macro.

50526 **ERRORS**

50527 No errors are defined.

50528 **EXAMPLES**

50529 None.

50530 **APPLICATION USAGE**

50531 None.

50532 **RATIONALE**

50533 None.

50534 **FUTURE DIRECTIONS**

50535 None.

50536 **SEE ALSO**

50537            *mblen()*, *mbtowc()*, *mbstowcs()*, *wcstombs()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
50538            <**stdlib.h**>

50539 **CHANGE HISTORY**

50540            First released in Issue 4. Derived from the ANSI C standard.

50541 **Issue 6**

50542            Extensions beyond the ISO C standard are marked.

50543            In the DESCRIPTION, a note about reentrancy and thread-safety is added.

50544 **NAME**

50545 wctrans — define character mapping

50546 **SYNOPSIS**

50547 #include &lt;wctype.h&gt;

50548 wctrans\_t wctrans(const char \*charclass);

50549 **DESCRIPTION**

50550 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 50551 conflict between the requirements described here and the ISO C standard is unintentional. This  
 50552 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50553 The *wctrans()* function is defined for valid character mapping names identified in the current  
 50554 locale. The *charclass* is a string identifying a generic character mapping name for which codeset-  
 50555 specific information is required. The following character mapping names are defined in all  
 50556 locales: **tolower** and **toupper**.

50557 The function shall return a value of type **wctrans\_t**, which can be used as the second argument  
 50558 to subsequent calls of *towctrans()*. The *wctrans()* function shall determine values of **wctrans\_t**  
 50559 according to the rules of the coded character set defined by character mapping information in  
 50560 the program's locale (category *LC\_CTYPE*). The values returned by *wctrans()* shall be valid until  
 50561 a call to *setlocale()* that modifies the category *LC\_CTYPE*.

50562 **RETURN VALUE**

50563 cx The *wctrans()* function shall return 0 and may set *errno* to indicate the error if the given  
 50564 character mapping name is not valid for the current locale (category *LC\_CTYPE*); otherwise, it  
 50565 shall return a non-zero object of type **wctrans\_t** that can be used in calls to *towctrans()*.

50566 **ERRORS**50567 The *wctrans()* function may fail if:

50568 cx [EINVAL] The character mapping name pointed to by *charclass* is not valid in the current  
 50569 locale.

50570 **EXAMPLES**

50571 None.

50572 **APPLICATION USAGE**

50573 None.

50574 **RATIONALE**

50575 None.

50576 **FUTURE DIRECTIONS**

50577 None.

50578 **SEE ALSO**50579 *towctrans()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wctype.h>50580 **CHANGE HISTORY**

50581 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

50582 **NAME**

50583 wctype — define character class

50584 **SYNOPSIS**

50585 #include &lt;wctype.h&gt;

50586 wctype\_t wctype(const char \*property);

50587 **DESCRIPTION**

50588 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 50589 conflict between the requirements described here and the ISO C standard is unintentional. This  
 50590 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50591 The *wctype()* function is defined for valid character class names as defined in the current locale.  
 50592 The *property* argument is a string identifying a generic character class for which codeset-specific  
 50593 type information is required. The following character class names shall be defined in all locales:

|       |              |              |               |
|-------|--------------|--------------|---------------|
| 50594 | <b>alnum</b> | <b>digit</b> | <b>punct</b>  |
| 50595 | <b>alpha</b> | <b>graph</b> | <b>space</b>  |
| 50596 | <b>blank</b> | <b>lower</b> | <b>upper</b>  |
| 50597 | <b>cntrl</b> | <b>print</b> | <b>xdigit</b> |

50598 Additional character class names defined in the locale definition file (category *LC\_CTYPE*) can  
 50599 also be specified.

50600 The function shall return a value of type **wctype\_t**, which can be used as the second argument to  
 50601 subsequent calls of *iswctype()*. The *wctype()* function shall determine values of **wctype\_t**  
 50602 according to the rules of the coded character set defined by character type information in the  
 50603 program's locale (category *LC\_CTYPE*). The values returned by *wctype()* shall be valid until a  
 50604 call to *setlocale()* that modifies the category *LC\_CTYPE*.

50605 **RETURN VALUE**

50606 The *wctype()* function shall return 0 if the given character class name is not valid for the current  
 50607 locale (category *LC\_CTYPE*); otherwise, it shall return an object of type **wctype\_t** that can be  
 50608 used in calls to *iswctype()*.

50609 **ERRORS**

50610 No errors are defined.

50611 **EXAMPLES**

50612 None.

50613 **APPLICATION USAGE**

50614 None.

50615 **RATIONALE**

50616 None.

50617 **FUTURE DIRECTIONS**

50618 None.

50619 **SEE ALSO**50620 *iswctype()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wctype.h**>50621 **CHANGE HISTORY**

50622 First released in Issue 4.

50623 **Issue 5**

50624 The following change has been made in this issue for alignment with  
50625 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 50626 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
50627 now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

50628 **NAME**

50629 `wcwidth` — number of column positions of a wide-character code

50630 **SYNOPSIS**

50631 XSI `#include <wchar.h>`

50632 `int wcwidth(wchar_t wc);`

50633

50634 **DESCRIPTION**

50635 The `wcwidth()` function shall determine the number of column positions required for the wide  
50636 character `wc`. The application shall ensure that the value of `wc` is a character representable as a  
50637 **wchar\_t**, and is a wide-character code corresponding to a valid character in the current locale.

50638 **RETURN VALUE**

50639 The `wcwidth()` function shall either return 0 (if `wc` is a null wide-character code), or return the  
50640 number of column positions to be occupied by the wide-character code `wc`, or return -1 (if `wc`  
50641 does not correspond to a printable wide-character code).

50642 **ERRORS**

50643 No errors are defined.

50644 **EXAMPLES**

50645 None.

50646 **APPLICATION USAGE**

50647 This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the  
50648 return value for a non-printable wide character is not specified.

50649 **RATIONALE**

50650 None.

50651 **FUTURE DIRECTIONS**

50652 None.

50653 **SEE ALSO**

50654 `wcswidth()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<wchar.h>`

50655 **CHANGE HISTORY**

50656 First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working  
50657 draft.

50658 **Issue 6**

50659 The Open Group Corrigendum U021/12 is applied. This function is marked as an extension.

50660 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50661 **NAME**

50662 wmemchr — find a wide character in memory

50663 **SYNOPSIS**

50664 #include &lt;wchar.h&gt;

50665 wchar\_t \*wmemchr(const wchar\_t \*ws, wchar\_t wc, size\_t n);

50666 **DESCRIPTION**

50667 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50668 conflict between the requirements described here and the ISO C standard is unintentional. This  
50669 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50670 The *wmemchr()* function shall locate the first occurrence of *wc* in the initial *n* wide characters of  
50671 the object pointed to by *ws*. This function shall not be affected by locale and all **wchar\_t** values  
50672 shall be treated identically. The null wide character and **wchar\_t** values not corresponding to  
50673 valid characters shall not be treated specially.

50674 If *n* is zero, the application shall ensure that *ws* is a valid pointer and the function behaves as if  
50675 no valid occurrence of *wc* is found.

50676 **RETURN VALUE**

50677 The *wmemchr()* function shall return a pointer to the located wide character, or a null pointer if  
50678 the wide character does not occur in the object.

50679 **ERRORS**

50680 No errors are defined.

50681 **EXAMPLES**

50682 None.

50683 **APPLICATION USAGE**

50684 None.

50685 **RATIONALE**

50686 None.

50687 **FUTURE DIRECTIONS**

50688 None.

50689 **SEE ALSO**

50690 *wmemcmp()*, *wmemcpy()*, *wmemmove()*, *wmemset()*, the Base Definitions volume of  
50691 IEEE Std 1003.1-2001, <wchar.h>

50692 **CHANGE HISTORY**

50693 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
50694 (E).

50695 **Issue 6**

50696 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50697 **NAME**

50698 wmemcmp — compare wide characters in memory

50699 **SYNOPSIS**

50700 #include &lt;wchar.h&gt;

50701 int wmemcmp(const wchar\_t \*ws1, const wchar\_t \*ws2, size\_t n);

50702 **DESCRIPTION**

50703 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50704 conflict between the requirements described here and the ISO C standard is unintentional. This  
50705 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50706 The *wmemcmp()* function shall compare the first *n* wide characters of the object pointed to by  
50707 *ws1* to the first *n* wide characters of the object pointed to by *ws2*. This function shall not be  
50708 affected by locale and all **wchar\_t** values shall be treated identically. The null wide character and  
50709 **wchar\_t** values not corresponding to valid characters shall not be treated specially.

50710 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function  
50711 shall behave as if the two objects compare equal.

50712 **RETURN VALUE**

50713 The *wmemcmp()* function shall return an integer greater than, equal to, or less than zero,  
50714 respectively, as the object pointed to by *ws1* is greater than, equal to, or less than the object  
50715 pointed to by *ws2*.

50716 **ERRORS**

50717 No errors are defined.

50718 **EXAMPLES**

50719 None.

50720 **APPLICATION USAGE**

50721 None.

50722 **RATIONALE**

50723 None.

50724 **FUTURE DIRECTIONS**

50725 None.

50726 **SEE ALSO**

50727 *wmemchr()*, *wmemcpy()*, *wmemmove()*, *wmemset()*, the Base Definitions volume of  
50728 IEEE Std 1003.1-2001, <**wchar.h**>

50729 **CHANGE HISTORY**

50730 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
50731 (E).

50732 **Issue 6**

50733 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50734 **NAME**

50735 wmemcpy — copy wide characters in memory

50736 **SYNOPSIS**

50737 #include &lt;wchar.h&gt;

50738 wchar\_t \*wmemcpy(wchar\_t \*restrict ws1, const wchar\_t \*restrict ws2,  
50739 size\_t n);50740 **DESCRIPTION**50741 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50742 conflict between the requirements described here and the ISO C standard is unintentional. This  
50743 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.50744 The *wmemcpy()* function shall copy *n* wide characters from the object pointed to by *ws2* to the  
50745 object pointed to by *ws1*. This function shall not be affected by locale and all **wchar\_t** values  
50746 shall be treated identically. The null wide character and **wchar\_t** values not corresponding to  
50747 valid characters shall not be treated specially.50748 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function  
50749 shall copy zero wide characters.50750 **RETURN VALUE**50751 The *wmemcpy()* function shall return the value of *ws1*.50752 **ERRORS**

50753 No errors are defined.

50754 **EXAMPLES**

50755 None.

50756 **APPLICATION USAGE**

50757 None.

50758 **RATIONALE**

50759 None.

50760 **FUTURE DIRECTIONS**

50761 None.

50762 **SEE ALSO**50763 *wmemchr()*, *wmemcmp()*, *wmemmove()*, *wmemset()*, the Base Definitions volume of  
50764 IEEE Std 1003.1-2001, <**wchar.h**>50765 **CHANGE HISTORY**50766 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
50767 (E).50768 **Issue 6**

50769 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50770 The *wmemcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

50771 **NAME**

50772 wmemmove — copy wide characters in memory with overlapping areas

50773 **SYNOPSIS**

50774 #include <wchar.h>

50775 wchar\_t \*wmemmove(wchar\_t \*ws1, const wchar\_t \*ws2, size\_t n);

50776 **DESCRIPTION**

50777 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50778 conflict between the requirements described here and the ISO C standard is unintentional. This  
50779 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50780 The *wmemmove()* function shall copy *n* wide characters from the object pointed to by *ws2* to the  
50781 object pointed to by *ws1*. Copying shall take place as if the *n* wide characters from the object  
50782 pointed to by *ws2* are first copied into a temporary array of *n* wide characters that does not  
50783 overlap the objects pointed to by *ws1* or *ws2*, and then the *n* wide characters from the temporary  
50784 array are copied into the object pointed to by *ws1*.

50785 This function shall not be affected by locale and all **wchar\_t** values shall be treated identically.  
50786 The null wide character and **wchar\_t** values not corresponding to valid characters shall not be  
50787 treated specially.

50788 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function  
50789 shall copy zero wide characters.

50790 **RETURN VALUE**

50791 The *wmemmove()* function shall return the value of *ws1*.

50792 **ERRORS**

50793 No errors are defined

50794 **EXAMPLES**

50795 None.

50796 **APPLICATION USAGE**

50797 None.

50798 **RATIONALE**

50799 None.

50800 **FUTURE DIRECTIONS**

50801 None.

50802 **SEE ALSO**

50803 *wmemchr()*, *wmemcmp()*, *wmemcpy()*, *wmemset()*, the Base Definitions volume of  
50804 IEEE Std 1003.1-2001, <**wchar.h**>

50805 **CHANGE HISTORY**

50806 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
50807 (E).

50808 **Issue 6**

50809 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50810 **NAME**

50811 wmemset — set wide characters in memory

50812 **SYNOPSIS**

50813 #include &lt;wchar.h&gt;

50814 wchar\_t \*wmemset(wchar\_t \*ws, wchar\_t wc, size\_t n);

50815 **DESCRIPTION**

50816 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50817 conflict between the requirements described here and the ISO C standard is unintentional. This  
50818 volume of IEEE Std 1003.1-2001 defers to the ISO C standard.

50819 The *wmemset()* function shall copy the value of *wc* into each of the first *n* wide characters of the  
50820 object pointed to by *ws*. This function shall not be affected by locale and all **wchar\_t** values shall  
50821 be treated identically. The null wide character and **wchar\_t** values not corresponding to valid  
50822 characters shall not be treated specially.

50823 If *n* is zero, the application shall ensure that *ws* is a valid pointer, and the function shall copy  
50824 zero wide characters.

50825 **RETURN VALUE**50826 The *wmemset()* functions shall return the value of *ws*.50827 **ERRORS**

50828 No errors are defined.

50829 **EXAMPLES**

50830 None.

50831 **APPLICATION USAGE**

50832 None.

50833 **RATIONALE**

50834 None.

50835 **FUTURE DIRECTIONS**

50836 None.

50837 **SEE ALSO**

50838 *wmemchr()*, *wmemcmp()*, *wmemcpy()*, *wmemmove()*, the Base Definitions volume of  
50839 IEEE Std 1003.1-2001, <**wchar.h**>

50840 **CHANGE HISTORY**

50841 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
50842 (E).

50843 **Issue 6**

50844 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50845 **NAME**

50846 wordexp, wordfree — perform word expansions

50847 **SYNOPSIS**

50848 #include &lt;wordexp.h&gt;

50849 int wordexp(const char \*restrict words, wordexp\_t \*restrict pwordexp,  
50850 int flags);

50851 void wordfree(wordexp\_t \*pwordexp);

50852 **DESCRIPTION**

50853 The *wordexp()* function shall perform word expansions as described in the Shell and Utilities  
50854 volume of IEEE Std 1003.1-2001, Section 2.6, Word Expansions, subject to quoting as in the Shell  
50855 and Utilities volume of IEEE Std 1003.1-2001, Section 2.2, Quoting, and place the list of expanded  
50856 words into the structure pointed to by *pwordexp*.

50857 The *words* argument is a pointer to a string containing one or more words to be expanded. The  
50858 expansions shall be the same as would be performed by the command line interpreter if *words*  
50859 were the part of a command line representing the arguments to a utility. Therefore, the  
50860 application shall ensure that *words* does not contain an unquoted <newline> or any of the  
50861 unquoted shell special characters ' | ', '&', ';', '<', '>' except in the context of command  
50862 substitution as specified in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.6.3,  
50863 Command Substitution. It also shall not contain unquoted parentheses or braces, except in the  
50864 context of command or variable substitution. The application shall ensure that every member of  
50865 *words* which it expects to have expanded by *wordexp()* does not contain an unquoted initial  
50866 comment character. The application shall also ensure that any words which it intends to be  
50867 ignored (because they begin or continue a comment) are deleted from *words*. If the argument  
50868 *words* contains an unquoted comment character (number sign) that is the beginning of a token,  
50869 *wordexp()* shall either treat the comment character as a regular character, or interpret it as a  
50870 comment indicator and ignore the remainder of *words*.

50871 The structure type **wordexp\_t** is defined in the <**wordexp.h**> header and includes at least the  
50872 following members:

50873

50874

| Member Type | Member Name | Description                                                         |
|-------------|-------------|---------------------------------------------------------------------|
| size_t      | we_wordc    | Count of words matched by <i>words</i> .                            |
| char **     | we_wordv    | Pointer to list of expanded words.                                  |
| size_t      | we_offs     | Slots to reserve at the beginning of <i>pwordexp-&gt;we_wordv</i> . |

50877

50878 The *wordexp()* function shall store the number of generated words into *pwordexp->we\_wordc* and  
50879 a pointer to a list of pointers to words in *pwordexp->we\_wordv*. Each individual field created  
50880 during field splitting (see the Shell and Utilities volume of IEEE Std 1003.1-2001, Section 2.6.5,  
50881 Field Splitting) or pathname expansion (see the Shell and Utilities volume of  
50882 IEEE Std 1003.1-2001, Section 2.6.6, Pathname Expansion) shall be a separate word in the  
50883 *pwordexp->we\_wordv* list. The words shall be in order as described in the Shell and Utilities  
50884 volume of IEEE Std 1003.1-2001, Section 2.6, Word Expansions. The first pointer after the last  
50885 word pointer shall be a null pointer. The expansion of special parameters described in the Shell  
50886 and Utilities volume of IEEE Std 1003.1-2001, Section 2.5.2, Special Parameters is unspecified.

50887 It is the caller's responsibility to allocate the storage pointed to by *pwordexp*. The *wordexp()*  
50888 function shall allocate other space as needed, including memory pointed to by  
50889 *pwordexp->we\_wordv*. The *wordfree()* function frees any memory associated with *pwordexp* from a  
50890 previous call to *wordexp()*.

50891 The *flags* argument is used to control the behavior of *wordexp()*. The value of *flags* is the  
 50892 bitwise-inclusive OR of zero or more of the following constants, which are defined in  
 50893 **<wordexp.h>**:

- |       |              |                                                                                                    |
|-------|--------------|----------------------------------------------------------------------------------------------------|
| 50894 | WRDE_APPEND  | Append words generated to the ones from a previous call to <i>wordexp()</i> .                      |
| 50895 | WRDE_DOOFFS  | Make use of <i>pwordexp-&gt;we_offs</i> . If this flag is set, <i>pwordexp-&gt;we_offs</i> is used |
| 50896 |              | to specify how many null pointers to add to the beginning of                                       |
| 50897 |              | <i>pwordexp-&gt;we_wordv</i> . In other words, <i>pwordexp-&gt;we_wordv</i> shall point to         |
| 50898 |              | <i>pwordexp-&gt;we_offs</i> null pointers, followed by <i>pwordexp-&gt;we_wordc</i> word           |
| 50899 |              | pointers, followed by a null pointer.                                                              |
| 50900 | WRDE_NOCMD   | If the implementation supports the utilities defined in the Shell and                              |
| 50901 |              | Utilities volume of IEEE Std 1003.1-2001, fail if command substitution, as                         |
| 50902 |              | specified in the Shell and Utilities volume of IEEE Std 1003.1-2001, Section                       |
| 50903 |              | 2.6.3, Command Substitution, is requested.                                                         |
| 50904 | WRDE_REUSE   | The <i>pwordexp</i> argument was passed to a previous successful call to                           |
| 50905 |              | <i>wordexp()</i> , and has not been passed to <i>wordfree()</i> . The result shall be the          |
| 50906 |              | same as if the application had called <i>wordfree()</i> and then called <i>wordexp()</i>           |
| 50907 |              | without WRDE_REUSE.                                                                                |
| 50908 | WRDE_SHOWERR | Do not redirect <i>stderr</i> to <b>/dev/null</b> .                                                |
| 50909 | WRDE_UNDEF   | Report error on an attempt to expand an undefined shell variable.                                  |

50910 The WRDE\_APPEND flag can be used to append a new set of words to those generated by a  
 50911 previous call to *wordexp()*. The following rules apply to applications when two or more calls to  
 50912 *wordexp()* are made with the same value of *pwordexp* and without intervening calls to *wordfree()*:

- 50913 1. The first such call shall not set WRDE\_APPEND. All subsequent calls shall set it.
- 50914 2. All of the calls shall set WRDE\_DOOFFS, or all shall not set it.
- 50915 3. After the second and each subsequent call, *pwordexp->we\_wordv* shall point to a list  
 50916 containing the following:
  - 50917 a. Zero or more null pointers, as specified by WRDE\_DOOFFS and *pwordexp->we\_offs*
  - 50918 b. Pointers to the words that were in the *pwordexp->we\_wordv* list before the call, in the  
 50919 same order as before
  - 50920 c. Pointers to the new words generated by the latest call, in the specified order
- 50921 4. The count returned in *pwordexp->we\_wordc* shall be the total number of words from all of  
 50922 the calls.
- 50923 5. The application can change any of the fields after a call to *wordexp()*, but if it does it shall  
 50924 reset them to the original value before a subsequent call, using the same *pwordexp* value, to  
 50925 *wordfree()* or *wordexp()* with the WRDE\_APPEND or WRDE\_REUSE flag.

50926 If the implementation supports the utilities defined in the Shell and Utilities volume of  
 50927 IEEE Std 1003.1-2001, and *words* contains an unquoted character—<newline>, ' | ', ' & ', ' ; ',  
 50928 ' < ', ' > ', ' ( ', ' ) ', ' { ', ' } '—in an inappropriate context, *wordexp()* shall fail, and the number  
 50929 of expanded words shall be 0.

50930 Unless WRDE\_SHOWERR is set in *flags*, *wordexp()* shall redirect *stderr* to **/dev/null** for any  
 50931 utilities executed as a result of command substitution while expanding *words*. If  
 50932 WRDE\_SHOWERR is set, *wordexp()* may write messages to *stderr* if syntax errors are detected  
 50933 while expanding *words*.

50934 The application shall ensure that if WRDE\_DOOFFS is set, then *pwordexp->we\_offs* has the same  
50935 value for each *wordexp()* call and *wordfree()* call using a given *pwordexp*.

50936 The following constants are defined as error return values:

50937 WRDE\_BADCHAR One of the unquoted characters—<newline>, ' | ', ' & ', ' ; ', ' < ', ' > ',  
50938 ' ( ', ' ) ', ' { ', ' } '—appears in *words* in an inappropriate context.

50939 WRDE\_BADVAL Reference to undefined shell variable when WRDE\_UNDEF is set in *flags*.

50940 WRDE\_CMDSUB Command substitution requested when WRDE\_NOCMD was set in *flags*.

50941 WRDE\_NOSPACE Attempt to allocate memory failed.

50942 WRDE\_SYNTAX Shell syntax error, such as unbalanced parentheses or unterminated  
50943 string.

#### 50944 RETURN VALUE

50945 Upon successful completion, *wordexp()* shall return 0. Otherwise, a non-zero value, as described  
50946 in <**wordexp.h**>, shall be returned to indicate an error. If *wordexp()* returns the value  
50947 WRDE\_NOSPACE, then *pwordexp->we\_wordc* and *pwordexp->we\_wordv* shall be updated to  
50948 reflect any words that were successfully expanded. In other cases, they shall not be modified.

50949 The *wordfree()* function shall not return a value.

#### 50950 ERRORS

50951 No errors are defined.

#### 50952 EXAMPLES

50953 None.

#### 50954 APPLICATION USAGE

50955 The *wordexp()* function is intended to be used by an application that wants to do all of the shell's  
50956 expansions on a word or words obtained from a user. For example, if the application prompts  
50957 for a filename (or list of filenames) and then uses *wordexp()* to process the input, the user could  
50958 respond with anything that would be valid as input to the shell.

50959 The WRDE\_NOCMD flag is provided for applications that, for security or other reasons, want to  
50960 prevent a user from executing shell commands. Disallowing unquoted shell special characters  
50961 also prevents unwanted side effects, such as executing a command or writing a file.

#### 50962 RATIONALE

50963 This function was included as an alternative to *glob()*. There had been continuing controversy  
50964 over exactly what features should be included in *glob()*. It is hoped that by providing *wordexp()*  
50965 (which provides all of the shell word expansions, but which may be slow to execute) and *glob()*  
50966 (which is faster, but which only performs pathname expansion, without tilde or parameter  
50967 expansion) this will satisfy the majority of applications.

50968 While *wordexp()* could be implemented entirely as a library routine, it is expected that most  
50969 implementations run a shell in a subprocess to do the expansion.

50970 Two different approaches have been proposed for how the required information might be  
50971 presented to the shell and the results returned. They are presented here as examples.

50972 One proposal is to extend the *echo* utility by adding a **-q** option. This option would cause *echo* to  
50973 add a backslash before each backslash and <blank> that occurs within an argument. The  
50974 *wordexp()* function could then invoke the shell as follows:

```
50975 (void) strcpy(buffer, "echo -q");
50976 (void) strcat(buffer, words);
50977 if ((flags & WRDE_SHOWERR) == 0)
```

```
50978 (void) strcat(buffer, "2>/dev/null");
50979 f = popen(buffer, "r");
```

50980 The *wordexp()* function would read the resulting output, remove unquoted backslashes, and  
 50981 break into words at unquoted <blank>s. If the WRDE\_NOCMD flag was set, *wordexp()* would  
 50982 have to scan *words* before starting the subshell to make sure that there would be no command  
 50983 substitution. In any case, it would have to scan *words* for unquoted special characters.

50984 Another proposal is to add the following options to *sh*:

50985 **-w** *wordlist*

50986 This option provides a wordlist expansion service to applications. The words in *wordlist*  
 50987 shall be expanded and the following written to standard output:

- 50988 1. The count of the number of words after expansion, in decimal, followed by a null byte
- 50989 2. The number of bytes needed to represent the expanded words (not including null  
 50990 separators), in decimal, followed by a null byte
- 50991 3. The expanded words, each terminated by a null byte

50992 If an error is encountered during word expansion, *sh* exits with a non-zero status after  
 50993 writing the former to report any words successfully expanded

50994 **-P** Run in “protected” mode. If specified with the **-w** option, no command substitution shall  
 50995 be performed.

50996 With these options, *wordexp()* could be implemented fairly simply by creating a subprocess  
 50997 using *fork()* and executing *sh* using the line:

```
50998 execl(<shell path>, "sh", "-P", "-w", words, (char *)0);
```

50999 after directing standard error to **/dev/null**.

51000 It seemed objectionable for a library routine to write messages to standard error, unless  
 51001 explicitly requested, so *wordexp()* is required to redirect standard error to **/dev/null** to ensure  
 51002 that no messages are generated, even for commands executed for command substitution. The  
 51003 WRDE\_SHOWERR flag can be specified to request that error messages be written.

51004 The WRDE\_REUSE flag allows the implementation to avoid the expense of freeing and  
 51005 reallocating memory, if that is possible. A minimal implementation can call *wordfree()* when  
 51006 WRDE\_REUSE is set.

#### 51007 FUTURE DIRECTIONS

51008 None.

#### 51009 SEE ALSO

51010 *fnmatch()*, *glob()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<wordexp.h>**, the Shell  
 51011 and Utilities volume of IEEE Std 1003.1-2001, Chapter 2, Shell Command Language

#### 51012 CHANGE HISTORY

51013 First released in Issue 4. Derived from the ISO POSIX-2 standard.

#### 51014 Issue 5

51015 Moved from POSIX2 C-language Binding to BASE.

#### 51016 Issue 6

51017 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

51018 The **restrict** keyword is added to the *wordexp()* prototype for alignment with the  
 51019 ISO/IEC 9899:1999 standard.

51020 **NAME**

51021           wprintf — print formatted wide-character output

51022 **SYNOPSIS**

51023           #include &lt;stdio.h&gt;

51024           #include &lt;wchar.h&gt;

51025           int wprintf(const wchar\_t \*restrict *format*, ...);51026 **DESCRIPTION**51027           Refer to *fwprintf()*.

## 51028 NAME

51029 pwrite, write — write on a file

## 51030 SYNOPSIS

51031 #include &lt;unistd.h&gt;

51032 XSI ssize\_t pwrite(int *fildev*, const void \**buf*, size\_t *nbyte*,  
51033 off\_t *offset*);51034 ssize\_t write(int *fildev*, const void \**buf*, size\_t *nbyte*);

## 51035 DESCRIPTION

51036 The *write()* function shall attempt to write *nbyte* bytes from the buffer pointed to by *buf* to the  
51037 file associated with the open file descriptor, *fildev*.51038 Before any action described below is taken, and if *nbyte* is zero and the file is a regular file, the  
51039 *write()* function may detect and return errors as described below. In the absence of errors, or if  
51040 error detection is not performed, the *write()* function shall return zero and have no other results.  
51041 If *nbyte* is zero and the file is not a regular file, the results are unspecified.51042 On a regular file or other file capable of seeking, the actual writing of data shall proceed from the  
51043 position in the file indicated by the file offset associated with *fildev*. Before successful return  
51044 from *write()*, the file offset shall be incremented by the number of bytes actually written. On a  
51045 regular file, if this incremented file offset is greater than the length of the file, the length of the  
51046 file shall be set to this file offset.51047 On a file not capable of seeking, writing shall always take place starting at the current position.  
51048 The value of a file offset associated with such a device is undefined.51049 If the O\_APPEND flag of the file status flags is set, the file offset shall be set to the end of the file  
51050 prior to each write and no intervening file modification operation shall occur between changing  
51051 the file offset and the write operation.51052 XSI If a *write()* requests that more bytes be written than there is room for (for example, the process'  
51053 file size limit or the physical end of a medium), only as many bytes as there is room for shall be  
51054 written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A  
51055 write of 512 bytes will return 20. The next write of a non-zero number of bytes would give a  
51056 failure return (except as noted below).51057 XSI If the request would cause the file size to exceed the soft file size limit for the process and there  
51058 is no room for any bytes to be written, the request shall fail and the implementation shall  
51059 generate the SIGXFSZ signal for the thread.51060 If *write()* is interrupted by a signal before it writes any data, it shall return  $-1$  with *errno* set to  
51061 [EINTR].51062 If *write()* is interrupted by a signal after it successfully writes some data, it shall return the  
51063 number of bytes written.51064 If the value of *nbyte* is greater than {SSIZE\_MAX}, the result is implementation-defined.51065 After a *write()* to a regular file has successfully returned:

- 51066
- Any successful *read()* from each byte position in the file that was modified by that write shall  
51067 return the data specified by the *write()* for that position until such byte positions are again  
51068 modified.
  - Any subsequent successful *write()* to the same byte position in the file shall overwrite that  
51069 file data.  
51070

- 51071 Write requests to a pipe or FIFO shall be handled in the same way as a regular file with the  
51072 following exceptions:
- 51073 • There is no file offset associated with a pipe, hence each write request shall append to the  
51074 end of the pipe.
  - 51075 • Write requests of {PIPE\_BUF} bytes or less shall not be interleaved with data from other  
51076 processes doing writes on the same pipe. Writes of greater than {PIPE\_BUF} bytes may have  
51077 data interleaved, on arbitrary boundaries, with writes by other processes, whether or not the  
51078 O\_NONBLOCK flag of the file status flags is set.
  - 51079 • If the O\_NONBLOCK flag is clear, a write request may cause the thread to block, but on  
51080 normal completion it shall return *nbyte*.
  - 51081 • If the O\_NONBLOCK flag is set, *write()* requests shall be handled differently, in the  
51082 following ways:
    - 51083 — The *write()* function shall not block the thread.
    - 51084 — A write request for {PIPE\_BUF} or fewer bytes shall have the following effect: if there is  
51085 sufficient space available in the pipe, *write()* shall transfer all the data and return the  
51086 number of bytes requested. Otherwise, *write()* shall transfer no data and return  $-1$  with  
51087 *errno* set to [EAGAIN].
    - 51088 — A write request for more than {PIPE\_BUF} bytes shall cause one of the following:
      - 51089 — When at least one byte can be written, transfer what it can and return the number of  
51090 bytes written. When all data previously written to the pipe is read, it shall transfer at  
51091 least {PIPE\_BUF} bytes.
      - 51092 — When no data can be written, transfer no data, and return  $-1$  with *errno* set to  
51093 [EAGAIN].
- 51094 When attempting to write to a file descriptor (other than a pipe or FIFO) that supports non-  
51095 blocking writes and cannot accept the data immediately:
- 51096 • If the O\_NONBLOCK flag is clear, *write()* shall block the calling thread until the data can be  
51097 accepted.
  - 51098 • If the O\_NONBLOCK flag is set, *write()* shall not block the thread. If some data can be  
51099 written without blocking the thread, *write()* shall write what it can and return the number of  
51100 bytes written. Otherwise, it shall return  $-1$  and set *errno* to [EAGAIN].
- 51101 Upon successful completion, where *nbyte* is greater than 0, *write()* shall mark for update the  
51102 *st\_ctime* and *st\_mtime* fields of the file, and if the file is a regular file, the S\_ISUID and S\_ISGID  
51103 bits of the file mode may be cleared.
- 51104 For regular files, no data transfer shall occur past the offset maximum established in the open  
51105 file description associated with *fildes*.
- 51106 If *fildes* refers to a socket, *write()* shall be equivalent to *send()* with no flags set.
- 51107 SIO If the O\_DSYNC bit has been set, write I/O operations on the file descriptor shall complete as  
51108 defined by synchronized I/O data integrity completion.
- 51109 If the O\_SYNC bit has been set, write I/O operations on the file descriptor shall complete as  
51110 defined by synchronized I/O file integrity completion.
- 51111 SHM If *fildes* refers to a shared memory object, the result of the *write()* function is unspecified.
- 51112 TYM If *fildes* refers to a typed memory object, the result of the *write()* function is unspecified.

51113 XSR If *fildev* refers to a STREAM, the operation of *write()* shall be determined by the values of the  
 51114 minimum and maximum *nbyte* range (packet size) accepted by the STREAM. These values are  
 51115 determined by the topmost STREAM module. If *nbyte* falls within the packet size range, *nbyte*  
 51116 bytes shall be written. If *nbyte* does not fall within the range and the minimum packet size value  
 51117 is 0, *write()* shall break the buffer into maximum packet size segments prior to sending the data  
 51118 downstream (the last segment may contain less than the maximum packet size). If *nbyte* does not  
 51119 fall within the range and the minimum value is non-zero, *write()* shall fail with *errno* set to  
 51120 [ERANGE]. Writing a zero-length buffer (*nbyte* is 0) to a STREAMS device sends 0 bytes with 0  
 51121 returned. However, writing a zero-length buffer to a STREAMS-based pipe or FIFO sends no  
 51122 message and 0 is returned. The process may issue *I\_SWROPT ioctl()* to enable zero-length  
 51123 messages to be sent across the pipe or FIFO.

51124 When writing to a STREAM, data messages are created with a priority band of 0. When writing  
 51125 to a STREAM that is not a pipe or FIFO:

- If *O\_NONBLOCK* is clear, and the STREAM cannot accept data (the STREAM write queue is full due to internal flow control conditions), *write()* shall block until data can be accepted.

- If *O\_NONBLOCK* is set and the STREAM cannot accept data, *write()* shall return *-1* and set *errno* to [EAGAIN].

- If *O\_NONBLOCK* is set and part of the buffer has been written while a condition in which the STREAM cannot accept additional data occurs, *write()* shall terminate and return the number of bytes written.

51133 In addition, *write()* shall fail if the STREAM head has processed an asynchronous error before  
 51134 the call. In this case, the value of *errno* does not reflect the result of *write()*, but reflects the prior  
 51135 error.

51136 XSI The *pwrite()* function shall be equivalent to *write()*, except that it writes into a given position  
 51137 without changing the file pointer. The first three arguments to *pwrite()* are the same as *write()*  
 51138 with the addition of a fourth argument offset for the desired position inside the file.

#### 51139 RETURN VALUE

51140 XSI Upon successful completion, *write()* and *pwrite()* shall return the number of bytes actually  
 51141 written to the file associated with *fildev*. This number shall never be greater than *nbyte*.  
 51142 Otherwise, *-1* shall be returned and *errno* set to indicate the error.

#### 51143 ERRORS

51144 XSI The *write()* and *pwrite()* functions shall fail if:

51145 [EAGAIN] The *O\_NONBLOCK* flag is set for the file descriptor and the thread would be  
 51146 delayed in the *write()* operation.

51147 [EBADF] The *fildev* argument is not a valid file descriptor open for writing.

51148 [EFBIG] An attempt was made to write a file that exceeds the implementation-defined  
 51149 XSI maximum file size or the process' file size limit, and there was no room for  
 51150 any bytes to be written.

51151 [EFBIG] The file is a regular file, *nbyte* is greater than 0, and the starting position is  
 51152 greater than or equal to the offset maximum established in the open file  
 51153 description associated with *fildev*.

51154 [EINTR] The write operation was terminated due to the receipt of a signal, and no data  
 51155 was transferred.

51156 [EIO] The process is a member of a background process group attempting to write  
 51157 to its controlling terminal, *TOSTOP* is set, the process is neither ignoring nor

|           |                           |                                                                                                                                                                                                               |
|-----------|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 51158     |                           | blocking SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.                                                                  |
| 51159     |                           |                                                                                                                                                                                                               |
| 51160     | [ENOSPC]                  | There was no free space remaining on the device containing the file.                                                                                                                                          |
| 51161     | [EPIPE]                   | An attempt is made to write to a pipe or FIFO that is not open for reading by any process, or that only has one end open. A SIGPIPE signal shall also be sent to the thread.                                  |
| 51162     |                           |                                                                                                                                                                                                               |
| 51163     |                           |                                                                                                                                                                                                               |
| 51164 XSR | [ERANGE]                  | The transfer request size was outside the range supported by the STREAMS file associated with <i>fildev</i> .                                                                                                 |
| 51165     |                           |                                                                                                                                                                                                               |
| 51166     |                           | The <i>write()</i> function shall fail if:                                                                                                                                                                    |
| 51167     | [EAGAIN] or [EWOULDBLOCK] |                                                                                                                                                                                                               |
| 51168     |                           | The file descriptor is for a socket, is marked O_NONBLOCK, and write would block.                                                                                                                             |
| 51169     |                           |                                                                                                                                                                                                               |
| 51170     | [ECONNRESET]              | A write was attempted on a socket that is not connected.                                                                                                                                                      |
| 51171     | [EPIPE]                   | A write was attempted on a socket that is shut down for writing, or is no longer connected. In the latter case, if the socket is of type SOCK_STREAM, the SIGPIPE signal is generated to the calling process. |
| 51172     |                           |                                                                                                                                                                                                               |
| 51173     |                           |                                                                                                                                                                                                               |
| 51174 XSI |                           | The <i>write()</i> and <i>pwrite()</i> functions may fail if:                                                                                                                                                 |
| 51175 XSR | [EINVAL]                  | The STREAM or multiplexer referenced by <i>fildev</i> is linked (directly or indirectly) downstream from a multiplexer.                                                                                       |
| 51176     |                           |                                                                                                                                                                                                               |
| 51177     | [EIO]                     | A physical I/O error has occurred.                                                                                                                                                                            |
| 51178     | [ENOBUFS]                 | Insufficient resources were available in the system to perform the operation.                                                                                                                                 |
| 51179     | [ENXIO]                   | A request was made of a nonexistent device, or the request was outside the capabilities of the device.                                                                                                        |
| 51180     |                           |                                                                                                                                                                                                               |
| 51181 XSR | [ENXIO]                   | A hangup occurred on the STREAM being written to.                                                                                                                                                             |
| 51182 XSR |                           | A write to a STREAMS file may fail if an error message has been received at the STREAM head. In this case, <i>errno</i> is set to the value included in the error message.                                    |
| 51183     |                           |                                                                                                                                                                                                               |
| 51184     |                           | The <i>write()</i> function may fail if:                                                                                                                                                                      |
| 51185     | [EACCES]                  | A write was attempted on a socket and the calling process does not have appropriate privileges.                                                                                                               |
| 51186     |                           |                                                                                                                                                                                                               |
| 51187     | [ENETDOWN]                | A write was attempted on a socket and the local network interface used to reach the destination is down.                                                                                                      |
| 51188     |                           |                                                                                                                                                                                                               |
| 51189     | [ENETUNREACH]             |                                                                                                                                                                                                               |
| 51190     |                           | A write was attempted on a socket and no route to the network is present.                                                                                                                                     |
| 51191 XSI |                           | The <i>pwrite()</i> function shall fail and the file pointer remain unchanged if:                                                                                                                             |
| 51192 XSI | [EINVAL]                  | The <i>offset</i> argument is invalid. The value is negative.                                                                                                                                                 |
| 51193 XSI | [ESPIPE]                  | <i>fildev</i> is associated with a pipe or FIFO.                                                                                                                                                              |

51194 **EXAMPLES**51195 **Writing from a Buffer**

51196 The following example writes data from the buffer pointed to by *buf* to the file associated with  
51197 the file descriptor *fd*.

```
51198 #include <sys/types.h>
51199 #include <string.h>
51200 ...
51201 char buf[20];
51202 size_t nbytes;
51203 ssize_t bytes_written;
51204 int fd;
51205 ...
51206 strcpy(buf, "This is a test\n");
51207 nbytes = strlen(buf);

51208 bytes_written = write(fd, buf, nbytes);
51209 ...
```

51210 **APPLICATION USAGE**

51211 None.

51212 **RATIONALE**

51213 See also the RATIONALE section in *read()*.

51214 An attempt to write to a pipe or FIFO has several major characteristics:

- 51215 • *Atomic/non-atomic*: A write is atomic if the whole amount written in one operation is not  
51216 interleaved with data from any other process. This is useful when there are multiple writers  
51217 sending data to a single reader. Applications need to know how large a write request can be  
51218 expected to be performed atomically. This maximum is called {PIPE\_BUF}. This volume of  
51219 IEEE Std 1003.1-2001 does not say whether write requests for more than {PIPE\_BUF} bytes  
51220 are atomic, but requires that writes of {PIPE\_BUF} or fewer bytes shall be atomic.
- 51221 • *Blocking/immediate*: Blocking is only possible with O\_NONBLOCK clear. If there is enough  
51222 space for all the data requested to be written immediately, the implementation should do so.  
51223 Otherwise, the process may block; that is, pause until enough space is available for writing.  
51224 The effective size of a pipe or FIFO (the maximum amount that can be written in one  
51225 operation without blocking) may vary dynamically, depending on the implementation, so it  
51226 is not possible to specify a fixed value for it.

- 51227 • *Complete/partial/deferred*: A write request:

```
51228 int fildes;
51229 size_t nbyte;
51230 ssize_t ret;
51231 char *buf;

51232 ret = write(fildes, buf, nbyte);
```

51233 may return:

51234 Complete    *ret*=*nbyte*

51235 Partial     *ret*<*nbyte*

51236 This shall never happen if *nbyte*≤{PIPE\_BUF}. If it does happen (with  
51237 *nbyte*>{PIPE\_BUF}), this volume of IEEE Std 1003.1-2001 does not guarantee

51238 atomicity, even if  $ret \leq \{PIPE\_BUF\}$ , because atomicity is guaranteed according  
51239 to the amount *requested*, not the amount *written*.

51240 Deferred:  $ret = -1$ ,  $errno = [EAGAIN]$

51241 This error indicates that a later request may succeed. It does not indicate that it  
51242 *shall* succeed, even if  $nbyte \leq \{PIPE\_BUF\}$ , because if no process reads from the  
51243 pipe or FIFO, the write never succeeds. An application could usefully count the  
51244 number of times `[EAGAIN]` is caused by a particular value of  
51245  $nbyte > \{PIPE\_BUF\}$  and perhaps do later writes with a smaller value, on the  
51246 assumption that the effective size of the pipe may have decreased.

51247 Partial and deferred writes are only possible with `O_NONBLOCK` set.

51248 The relations of these properties are shown in the following tables:

51249

51250

51251

51252

51253

51254

| Write to a Pipe or FIFO with <code>O_NONBLOCK</code> clear |                                 |                                 |                                  |
|------------------------------------------------------------|---------------------------------|---------------------------------|----------------------------------|
| Immediately Writable:                                      | None                            | Some                            | <i>nbyte</i>                     |
| $nbyte \leq \{PIPE\_BUF\}$                                 | Atomic blocking<br><i>nbyte</i> | Atomic blocking<br><i>nbyte</i> | Atomic immediate<br><i>nbyte</i> |
| $nbyte > \{PIPE\_BUF\}$                                    | Blocking <i>nbyte</i>           | Blocking <i>nbyte</i>           | Blocking <i>nbyte</i>            |

51255 If the `O_NONBLOCK` flag is clear, a write request shall block if the amount writable  
51256 immediately is less than that requested. If the flag is set (by `fcntl()`), a write request shall never  
51257 block.

51258

51259

51260

51261

51262

51263

| Write to a Pipe or FIFO with <code>O_NONBLOCK</code> set |                           |                                                |                                                |
|----------------------------------------------------------|---------------------------|------------------------------------------------|------------------------------------------------|
| Immediately Writable:                                    | None                      | Some                                           | <i>nbyte</i>                                   |
| $nbyte \leq \{PIPE\_BUF\}$                               | -1, <code>[EAGAIN]</code> | -1, <code>[EAGAIN]</code>                      | Atomic <i>nbyte</i>                            |
| $nbyte > \{PIPE\_BUF\}$                                  | -1, <code>[EAGAIN]</code> | < <i>nbyte</i> or -1,<br><code>[EAGAIN]</code> | ≤ <i>nbyte</i> or -1,<br><code>[EAGAIN]</code> |

51264 There is no exception regarding partial writes when `O_NONBLOCK` is set. With the exception  
51265 of writing to an empty pipe, this volume of IEEE Std 1003.1-2001 does not specify exactly when a  
51266 partial write is performed since that would require specifying internal details of the  
51267 implementation. Every application should be prepared to handle partial writes when  
51268 `O_NONBLOCK` is set and the requested amount is greater than `{PIPE_BUF}`, just as every  
51269 application should be prepared to handle partial writes on other kinds of file descriptors.

51270 The intent of forcing writing at least one byte if any can be written is to assure that each write  
51271 makes progress if there is any room in the pipe. If the pipe is empty, `{PIPE_BUF}` bytes must be  
51272 written; if not, at least some progress must have been made.

51273 Where this volume of IEEE Std 1003.1-2001 requires -1 to be returned and *errno* set to  
51274 `[EAGAIN]`, most historical implementations return zero (with the `O_NDELAY` flag set, which is  
51275 the historical predecessor of `O_NONBLOCK`, but is not itself in this volume of  
51276 IEEE Std 1003.1-2001). The error indications in this volume of IEEE Std 1003.1-2001 were chosen  
51277 so that an application can distinguish these cases from end-of-file. While `write()` cannot receive  
51278 an indication of end-of-file, `read()` can, and the two functions have similar return values. Also,  
51279 some existing systems (for example, Eighth Edition) permit a write of zero bytes to mean that  
51280 the reader should get an end-of-file indication; for those systems, a return value of zero from  
51281 `write()` indicates a successful write of an end-of-file indication.

51282 Implementations are allowed, but not required, to perform error checking for *write()* requests of  
51283 zero bytes.

51284 The concept of a {PIPE\_MAX} limit (indicating the maximum number of bytes that can be  
51285 written to a pipe in a single operation) was considered, but rejected, because this concept would  
51286 unnecessarily limit application writing.

51287 See also the discussion of O\_NONBLOCK in *read()*.

51288 Writes can be serialized with respect to other reads and writes. If a *read()* of file data can be  
51289 proven (by any means) to occur after a *write()* of the data, it must reflect that *write()*, even if the  
51290 calls are made by different processes. A similar requirement applies to multiple write operations  
51291 to the same file position. This is needed to guarantee the propagation of data from *write()* calls  
51292 to subsequent *read()* calls. This requirement is particularly significant for networked file  
51293 systems, where some caching schemes violate these semantics.

51294 Note that this is specified in terms of *read()* and *write()*. The XSI extensions *readv()* and *writv()*  
51295 also obey these semantics. A new “high-performance” write analog that did not follow these  
51296 serialization requirements would also be permitted by this wording. This volume of  
51297 IEEE Std 1003.1-2001 is also silent about any effects of application-level caching (such as that  
51298 done by *stdio*).

51299 This volume of IEEE Std 1003.1-2001 does not specify the value of the file offset after an error is  
51300 returned; there are too many cases. For programming errors, such as [EBADF], the concept is  
51301 meaningless since no file is involved. For errors that are detected immediately, such as  
51302 [EAGAIN], clearly the pointer should not change. After an interrupt or hardware error, however,  
51303 an updated value would be very useful and is the behavior of many implementations.

51304 This volume of IEEE Std 1003.1-2001 does not specify behavior of concurrent writes to a file from  
51305 multiple processes. Applications should use some form of concurrency control.

#### 51306 FUTURE DIRECTIONS

51307 None.

#### 51308 SEE ALSO

51309 *chmod()*, *creat()*, *dup()*, *fcntl()*, *getrlimit()*, *lseek()*, *open()*, *pipe()*, *ulimit()*, *writv()*, the Base  
51310 Definitions volume of IEEE Std 1003.1-2001, <limits.h>, <stropts.h>, <sys/uio.h>, <unistd.h>

#### 51311 CHANGE HISTORY

51312 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 51313 Issue 5

51314 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
51315 Threads Extension.

51316 Large File Summit extensions are added.

51317 The *pwrite()* function is added.

#### 51318 Issue 6

51319 The DESCRIPTION states that the *write()* function does not block the thread. Previously this  
51320 said “process” rather than “thread”.

51321 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are  
51322 marked as part of the XSI STREAMS Option Group.

51323 The following new requirements on POSIX implementations derive from alignment with the  
51324 Single UNIX Specification:

- 51325 • The DESCRIPTION now states that if `write()` is interrupted by a signal after it has  
51326 successfully written some data, it returns the number of bytes written. In the POSIX.1-1988  
51327 standard, it was optional whether `write()` returned the number of bytes written, or whether it  
51328 returned `-1` with `errno` set to `[EINTR]`. This is a FIPS requirement.
- 51329 • The following changes are made to support large files:
- 51330 — For regular files, no data transfer occurs past the offset maximum established in the open  
51331 file description associated with the *files*.
  - 51332 — A second `[EFBIG]` error condition is added.
- 51333 • The `[EIO]` error condition is added.
- 51334 • The `[EPIPE]` error condition is added for when a pipe has only one end open.
- 51335 • The `[ENXIO]` optional error condition is added.
- 51336 Text referring to sockets is added to the DESCRIPTION.
- 51337 The following changes were made to align with the IEEE P1003.1a draft standard:
- 51338 • The effect of reading zero bytes is clarified.
- 51339 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
51340 `write()` results are unspecified for typed memory objects.
- 51341 The following error conditions are added for operations on sockets: `[EAGAIN]`,  
51342 `[EWOULDBLOCK]`, `[ECONNRESET]`, `[ENOTCONN]`, and `[EPIPE]`.
- 51343 The `[EIO]` error is changed to “may fail”.
- 51344 The `[ENOBUFFS]` error is added for sockets.
- 51345 The following error conditions are added for operations on sockets: `[EACCES]`, `[ENETDOWN]`,  
51346 and `[ENETUNREACH]`.
- 51347 The `writenv()` function is split out into a separate reference page.

51348 **NAME**

51349 writev — write a vector

51350 **SYNOPSIS**51351 XSI 

```
#include <sys/uio.h>
```

51352 

```
ssize_t writev(int fildev, const struct iovec *iov, int iovcnt);
```

51353

51354 **DESCRIPTION**

51355 The *writev()* function shall be equivalent to *write()*, except as described below. The *writev()*  
 51356 function shall gather output data from the *iovcnt* buffers specified by the members of the *iov*  
 51357 array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*-1]. The *iovcnt* argument is valid if greater than 0 and less than  
 51358 or equal to {IOV\_MAX}, as defined in <limits.h>.

51359 Each *iovec* entry specifies the base address and length of an area in memory from which data  
 51360 should be written. The *writev()* function shall always write a complete area before proceeding to  
 51361 the next.

51362 If *fildev* refers to a regular file and all of the *iov\_len* members in the array pointed to by *iov* are 0,  
 51363 *writev()* shall return 0 and have no other effect. For other file types, the behavior is unspecified.

51364 If the sum of the *iov\_len* values is greater than {SSIZE\_MAX}, the operation shall fail and no data  
 51365 shall be transferred.

51366 **RETURN VALUE**

51367 Upon successful completion, *writev()* shall return the number of bytes actually written.  
 51368 Otherwise, it shall return a value of -1, the file-pointer shall remain unchanged, and *errno* shall  
 51369 be set to indicate an error.

51370 **ERRORS**51371 Refer to *write()*.51372 In addition, the *writev()* function shall fail if:51373 [EINVAL] The sum of the *iov\_len* values in the *iov* array would overflow an *ssize\_t*.51374 The *writev()* function may fail and set *errno* to:51375 [EINVAL] The *iovcnt* argument was less than or equal to 0, or greater than {IOV\_MAX}.51376 **EXAMPLES**51377 **Writing Data from an Array**

51378 The following example writes data from the buffers specified by members of the *iov* array to the  
 51379 file associated with the file descriptor *fd*.

51380 

```
#include <sys/types.h>
```

51381 

```
#include <sys/uio.h>
```

51382 

```
#include <unistd.h>
```

51383 

```
...
```

51384 

```
ssize_t bytes_written;
```

51385 

```
int fd;
```

51386 

```
char *buf0 = "short string\n";
```

51387 

```
char *buf1 = "This is a longer string\n";
```

51388 

```
char *buf2 = "This is the longest string in this example\n";
```

51389 

```
int iovcnt;
```

51390 

```
struct iovec iov[3];
```

```
51391 iov[0].iov_base = buf0;
51392 iov[0].iov_len = strlen(buf0);
51393 iov[1].iov_base = buf1;
51394 iov[1].iov_len = strlen(buf1);
51395 iov[2].iov_base = buf2;
51396 iov[2].iov_len = strlen(buf2);
51397 ...
51398 iovcnt = sizeof(iov) / sizeof(struct iovec);

51399 bytes_written = writev(fd, iov, iovcnt);
51400 ...
```

**51401 APPLICATION USAGE**

51402 None.

**51403 RATIONALE**

51404 Refer to *write()*.

**51405 FUTURE DIRECTIONS**

51406 None.

**51407 SEE ALSO**

51408 *readv()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-2001, <limits.h>, <sys/uio.h>

**51409 CHANGE HISTORY**

51410 First released in Issue 4, Version 2.

**51411 Issue 6**

51412 Split out from the *write()* reference page.

51413 **NAME**

51414           wscanf — convert formatted wide-character input

51415 **SYNOPSIS**

51416           #include &lt;stdio.h&gt;

51417           #include &lt;wchar.h&gt;

51418           int wscanf(const wchar\_t \*restrict *format*, ... );51419 **DESCRIPTION**51420           Refer to *fwscanf()*.

51421 **NAME**51422 `y0, y1, yn` — Bessel functions of the second kind51423 **SYNOPSIS**

```
51424 xSI #include <math.h>
51425 double y0(double x);
51426 double y1(double x);
51427 double yn(int n, double x);
51428
```

51429 **DESCRIPTION**

51430 The `y0()`, `y1()`, and `yn()` functions shall compute Bessel functions of  $x$  of the second kind of  
 51431 orders 0, 1, and  $n$ , respectively.

51432 An application wishing to check for error situations should set `errno` to zero and call  
 51433 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or  
 51434 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-  
 51435 zero, an error has occurred.

51436 **RETURN VALUE**

51437 Upon successful completion, these functions shall return the relevant Bessel value of  $x$  of the  
 51438 second kind.

51439 If  $x$  is NaN, NaN shall be returned.

51440 If the  $x$  argument to these functions is negative, `-HUGE_VAL` or NaN shall be returned, and a  
 51441 domain error may occur.

51442 If  $x$  is 0.0, `-HUGE_VAL` shall be returned and a range error may occur.

51443 If the correct result would cause underflow, 0.0 shall be returned and a range error may occur.

51444 If the correct result would cause overflow, `-HUGE_VAL` or 0.0 shall be returned and a range  
 51445 error may occur.

51446 **ERRORS**

51447 These functions may fail if:

51448 **Domain Error** The value of  $x$  is negative.

51449 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,  
 51450 then `errno` shall be set to [EDOM]. If the integer expression `(math_errhandling`  
 51451 `& MATH_ERREXCEPT)` is non-zero, then the invalid floating-point exception  
 51452 shall be raised.

51453 **Range Error** The value of  $x$  is 0.0, or the correct result would cause overflow.

51454 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,  
 51455 then `errno` shall be set to [ERANGE]. If the integer expression  
 51456 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the overflow  
 51457 floating-point exception shall be raised.

51458 **Range Error** The value of  $x$  is too large in magnitude, or the correct result would cause  
 51459 underflow.

51460 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,  
 51461 then `errno` shall be set to [ERANGE]. If the integer expression  
 51462 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the underflow  
 51463 floating-point exception shall be raised.

51464 **EXAMPLES**

51465 None.

51466 **APPLICATION USAGE**

51467 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
51468 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

51469 **RATIONALE**

51470 None.

51471 **FUTURE DIRECTIONS**

51472 None.

51473 **SEE ALSO**

51474 *feclearexcept()*, *fetetestexcept()*, *isnan()*, *j0()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
51475 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h>

51476 **CHANGE HISTORY**

51477 First released in Issue 1. Derived from Issue 1 of the SVID.

51478 **Issue 5**

51479 The DESCRIPTION is updated to indicate how an application should check for an error. This  
51480 text was previously published in the APPLICATION USAGE section.

51481 **Issue 6**

51482 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

51483 The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling  
51484 with the ISO/IEC 9899:1999 standard.

# Index

|                                                  |                |                                               |               |
|--------------------------------------------------|----------------|-----------------------------------------------|---------------|
| <code>_CS_PATH</code> .....                      | 214            | <code>_POSIX2 constants</code>                |               |
| <code>_CS_XBS5_ILP32_OFF32_CFLAGS</code> .....   | 214            | in <code>sysconf</code> .....                 | 1473          |
| <code>_CS_XBS5_ILP32_OFF32_LDFLAGS</code> .....  | 214            | <code>_POSIX2_CHAR_TERM</code> .....          | 1475          |
| <code>_CS_XBS5_ILP32_OFF32_LIBS</code> .....     | 214            | <code>_POSIX2_C_BIND</code> .....             | 1475          |
| <code>_CS_XBS5_ILP32_OFFBIG_CFLAGS</code> .....  | 214            | <code>_POSIX2_C_DEV</code> .....              | 1475          |
| <code>_CS_XBS5_ILP32_OFFBIG_LDFLAGS</code> ..... | 214            | <code>_POSIX2_C_VERSION</code> .....          | 1475          |
| <code>_CS_XBS5_ILP32_OFFBIG_LIBS</code> .....    | 214            | <code>_POSIX2_FORT_DEV</code> .....           | 1475          |
| <code>_CS_XBS5_LP64_OFF64_CFLAGS</code> .....    | 214            | <code>_POSIX2_FORT_RUN</code> .....           | 1475          |
| <code>_CS_XBS5_LP64_OFF64_LDFLAGS</code> .....   | 214            | <code>_POSIX2_LOCALEDEF</code> .....          | 1475          |
| <code>_CS_XBS5_LP64_OFF64_LIBS</code> .....      | 214            | <code>_POSIX2_PBS</code> .....                | 1475          |
| <code>_CS_XBS5_LPBIG_OFFBIG_CFLAGS</code> .....  | 214            | <code>_POSIX2_PBS_ACCOUNTING</code> .....     | 1475          |
| <code>_CS_XBS5_LPBIG_OFFBIG_LDFLAGS</code> ..... | 214            | <code>_POSIX2_PBS_LOCATE</code> .....         | 1475          |
| <code>_CS_XBS5_LPBIG_OFFBIG_LIBS</code> .....    | 214            | <code>_POSIX2_PBS_MESSAGE</code> .....        | 1475          |
| <code>_exit</code> .....                         | 85, 307, 1592  | <code>_POSIX2_PBS_TRACK</code> .....          | 1475          |
| <code>_Exit()</code> .....                       | 307, <b>85</b> | <code>_POSIX2_SW_DEV</code> .....             | 1475          |
| <code>_FILE</code> .....                         | 128            | <code>_POSIX2_UPE</code> .....                | 1475          |
| <code>_IOFBF</code> .....                        | 1285, 1327     | <code>_POSIX2_VERSION</code> .....            | 1475          |
| <code>_IOLBF</code> .....                        | 376, 1327      | <code>_POSIX</code> .....                     | 15, 17        |
| <code>_IONBF</code> .....                        | 1285, 1327     | <code>_POSIX_ADVISORY_INFO</code> .....       | 1473          |
| <code>_LINE</code> .....                         | 128            | <code>_POSIX_ASYNCHRONOUS_IO</code> .....     | 1473          |
| <code>_longjmp()</code> .....                    | <b>86</b>      | <code>_POSIX_ASYNC_IO</code> .....            | 399           |
| <code>_LVL</code> .....                          | 17             | <code>_POSIX_BARRIERS</code> .....            | 1473          |
| <code>_MAX</code> .....                          | 16             | <code>_POSIX_CHOWN_RESTRICTED</code> .....    | 190, 399, 401 |
| <code>_MIN</code> .....                          | 16             | <code>_POSIX_CLOCK_SELECTION</code> .....     | 1473          |
| <code>_PC constants</code>                       |                | <code>_POSIX_CPUTIME</code> .....             | 1473          |
| used in <code>pathconf</code> .....              | 399            | <code>_POSIX_C_SOURCE</code> .....            | 14            |
| <code>_PC_ALLOC_SIZE_MIN</code> .....            | 399            | <code>_POSIX_FILE_LOCKING</code> .....        | 1474          |
| <code>_PC_ASYNC_IO</code> .....                  | 399            | <code>_POSIX_FSYNC</code> .....               | 1474          |
| <code>_PC_CHOWN_RESTRICTED</code> .....          | 399            | <code>_POSIX_JOB_CONTROL</code> .....         | 1474          |
| <code>_PC_FILESIZEBITS</code> .....              | 399            | <code>_POSIX_MAPPED_FILES</code> .....        | 1474          |
| <code>_PC_LINK_MAX</code> .....                  | 399            | <code>_POSIX_MEMLOCK</code> .....             | 1474          |
| <code>_PC_MAX_CANON</code> .....                 | 399            | <code>_POSIX_MEMLOCK_RANGE</code> .....       | 1474          |
| <code>_PC_MAX_INPUT</code> .....                 | 399            | <code>_POSIX_MEMORY_PROTECTION</code> .....   | 1474          |
| <code>_PC_NAME_MAX</code> .....                  | 399            | <code>_POSIX_MESSAGE_PASSING</code> .....     | 1474          |
| <code>_PC_NO_TRUNC</code> .....                  | 399            | <code>_POSIX_MONOTONIC_CLOCK</code> .....     | 1474          |
| <code>_PC_PATH_MAX</code> .....                  | 399            | <code>_POSIX_MULTI_PROCESS</code> .....       | 1474          |
| <code>_PC_PIPE_BUF</code> .....                  | 399            | <code>_POSIX_NO_TRUNC</code> .....            | 399           |
| <code>_PC_PRIO_IO</code> .....                   | 399            | <code>_POSIX_OPEN_MAX</code> .....            | 556           |
| <code>_PC_REC_INCR_XFER_SIZE</code> .....        | 399            | <code>_POSIX_PRIORITIZED_IO</code> .....      | 42, 1474      |
| <code>_PC_REC_MAX_XFER_SIZE</code> .....         | 399            | <code>_POSIX_PRIORITY_SCHEDULING</code> ..... | 42, 1474      |
| <code>_PC_REC_MIN_XFER_SIZE</code> .....         | 399            | <code>_POSIX_PRIO_IO</code> .....             | 399           |
| <code>_PC_REC_XFER_ALIGN</code> .....            | 399            | <code>_POSIX_READER_WRITER_LOCKS</code> ..... | 1474          |
| <code>_PC_SYMLINK_MAX</code> .....               | 399            | <code>_POSIX_REALTIME_SIGNALS</code> .....    | 1474          |
| <code>_PC_SYNC_IO</code> .....                   | 399            | <code>_POSIX_REGEX</code> .....               | 1474          |
| <code>_PC_VDISABLE</code> .....                  | 399            | <code>_POSIX_SAVED_IDS</code> .....           | 1474          |
|                                                  |                | <code>_POSIX_SEMAPHORES</code> .....          | 1474          |

|                                                      |                 |
|------------------------------------------------------|-----------------|
| <code>_POSIX_SHARED_MEMORY_OBJECTS</code> .....      | 1474            |
| <code>_POSIX_SHELL</code> .....                      | 1474            |
| <code>_POSIX_SOURCE</code> .....                     | 14              |
| <code>_POSIX_SPAWN</code> .....                      | 1474            |
| <code>_POSIX_SPIN_LOCKS</code> .....                 | 1474            |
| <code>_POSIX_SPARADIC_SERVER</code> .....            | 1474            |
| <code>_POSIX_SYNCHRONIZED_IO</code> .....            | 1474            |
| <code>_POSIX_SYNC_IO</code> .....                    | 399             |
| <code>_POSIX_THREADS</code> .....                    | 241, 1474, 1533 |
| <code>_POSIX_THREAD_ATTR_STACKADDR</code> .....      | 1474            |
| <code>_POSIX_THREAD_ATTR_STACKSIZE</code> .....      | 1474            |
| <code>_POSIX_THREAD_CPUTIME</code> .....             | 1474            |
| <code>_POSIX_THREAD_PRIORITY_SCHEDULING</code> ..... | 1474            |
| <code>_POSIX_THREAD_PRIO_INHERIT</code> .....        | 1474            |
| <code>_POSIX_THREAD_PRIO_PROTECT</code> .....        | 1474            |
| <code>_POSIX_THREAD_PROCESS_SHARED</code> .....      | 1092            |
| .....                                                | 1474            |
| <code>_POSIX_THREAD_SAFE_FUNCTIONS</code> .....      | 241             |
| .....                                                | 1474, 1533      |
| <code>_POSIX_THREAD_SPORADIC_SERVER</code> .....     | 1474            |
| <code>_POSIX_TIMEOUTS</code> .....                   | 1474            |
| <code>_POSIX_TIMERS</code> .....                     | 1474            |
| <code>_POSIX_TRACE</code> .....                      | 1474            |
| <code>_POSIX_TRACE_EVENT_FILTER</code> .....         | 1474            |
| <code>_POSIX_TRACE_EVENT_NAME_MAX</code> .....       | 939, 941        |
| <code>_POSIX_TRACE_INHERIT</code> .....              | 1474            |
| <code>_POSIX_TRACE_LOG</code> .....                  | 1474            |
| <code>_POSIX_TRACE_SYS_MAX</code> .....              | 936             |
| <code>_POSIX_TRACE_USER_EVENT_MAX</code> .....       | 941             |
| <code>_POSIX_TYPED_MEMORY_OBJECTS</code> .....       | 1474            |
| <code>_POSIX_V6_ILP32_OFF32</code> .....             | 1474            |
| <code>_POSIX_V6_ILP32_OFFBIG</code> .....            | 1474            |
| <code>_POSIX_V6_LP64_OFF64</code> .....              | 1474            |
| <code>_POSIX_V6_LPBIG_OFFBIG</code> .....            | 1475            |
| <code>_POSIX_VDISABLE</code> .....                   | 399             |
| <code>_POSIX_VERSION</code> .....                    | 1474, 1557      |
| <code>_PROCESS</code> .....                          | 17              |
| <code>_PTHREAD_THREADS_MAX</code> .....              | 1066            |
| <code>_REGEX_VERSION</code> .....                    | 1475            |
| <code>_SC constants</code>                           |                 |
| in <code>sysconf</code> .....                        | 1473            |
| <code>_SC_2_CHAR_TERM</code> .....                   | 1475            |
| <code>_SC_2_C_BIND</code> .....                      | 1475            |
| <code>_SC_2_C_DEV</code> .....                       | 1475            |
| <code>_SC_2_C_VERSION</code> .....                   | 1475            |
| <code>_SC_2_FORT_DEV</code> .....                    | 1475            |
| <code>_SC_2_FORT_RUN</code> .....                    | 1475            |
| <code>_SC_2_LOCALEDEF</code> .....                   | 1475            |
| <code>_SC_2_PBS_ACCOUNTING</code> .....              | 1475            |
| <code>_SC_2_PBS_LOCATE</code> .....                  | 1475            |
| <code>_SC_2_PBS_MESSAGE</code> .....                 | 1475            |
| <code>_SC_2_PBS_TRACK</code> .....                   | 1475            |
| <code>_SC_2_SW_DEV</code> .....                      | 1475            |
| <code>_SC_2_UPE</code> .....                         | 1475            |
| <code>_SC_2_VERSION</code> .....                     | 863, 1475       |
| <code>_SC_ADVISORY_INFO</code> .....                 | 1473            |
| <code>_SC_AIO_LISTIO_MAX</code> .....                | 1473            |
| <code>_SC_AIO_MAX</code> .....                       | 1473            |
| <code>_SC_AIO_PRIO_DELTA_MAX</code> .....            | 1473            |
| <code>_SC_ARG_MAX</code> .....                       | 1473            |
| <code>_SC_ASYNCHRONOUS_IO</code> .....               | 1473            |
| <code>_SC_ATEXIT_MAX</code> .....                    | 1473            |
| <code>_SC_BARRIERS</code> .....                      | 1473            |
| <code>_SC_BC_BASE_MAX</code> .....                   | 1473            |
| <code>_SC_BC_DIM_MAX</code> .....                    | 1473            |
| <code>_SC_BC_SCALE_MAX</code> .....                  | 1473            |
| <code>_SC_BC_STRING_MAX</code> .....                 | 1473            |
| <code>_SC_CHILD_MAX</code> .....                     | 1473            |
| <code>_SC_CLK_TCK</code> .....                       | 1473, 1529      |
| <code>_SC_CLOCK_SELECTION</code> .....               | 1473            |
| <code>_SC_COLL_WEIGHTS_MAX</code> .....              | 1473            |
| <code>_SC_CPUTIME</code> .....                       | 1473            |
| <code>_SC_DELAYTIMER_MAX</code> .....                | 1473            |
| <code>_SC_EXPR_NEST_MAX</code> .....                 | 1473            |
| <code>_SC_FILE_LOCKING</code> .....                  | 1474            |
| <code>_SC_FSYNC</code> .....                         | 1474            |
| <code>_SC_GETGR_R_SIZE_MAX</code> .....              | 507, 510, 1473  |
| <code>_SC_GETPW_R_SIZE_MAX</code> .....              | 548, 551, 1473  |
| <code>_SC_IOV_MAX</code> .....                       | 1473            |
| <code>_SC_JOB_CONTROL</code> .....                   | 1474            |
| <code>_SC_LINE_MAX</code> .....                      | 1473            |
| <code>_SC_LOGIN_NAME_MAX</code> .....                | 1473            |
| <code>_SC_MEMLOCK</code> .....                       | 1474            |
| <code>_SC_MEMLOCK_RANGE</code> .....                 | 1474            |
| <code>_SC_MEMORY_PROTECTION</code> .....             | 1474            |
| <code>_SC_MESSAGE_PASSING</code> .....               | 1474            |
| <code>_SC_MONOTONIC_CLOCK</code> .....               | 1474            |
| <code>_SC_MQ_OPEN_MAX</code> .....                   | 1473            |
| <code>_SC_MQ_PRIO_MAX</code> .....                   | 1473            |
| <code>_SC_MULTI_PROCESS</code> .....                 | 1474            |
| <code>_SC_NGROUPS_MAX</code> .....                   | 1473            |
| <code>_SC_OPEN_MAX</code> .....                      | 1473            |
| <code>_SC_PAGESIZE</code> .....                      | 776, 868, 1475  |
| <code>_SC_PAGE_SIZE</code> .....                     | 776, 1475       |
| <code>_SC_PRIORITIZED_IO</code> .....                | 1474            |
| <code>_SC_PRIORITY_SCHEDULING</code> .....           | 1474            |
| <code>_SC_READER_WRITER_LOCKS</code> .....           | 1474            |
| <code>_SC_REALTIME_SIGNALS</code> .....              | 1474            |
| <code>_SC_REGEX</code> .....                         | 1474            |
| <code>_SC_REGEX_VERSION</code> .....                 | 1475            |
| <code>_SC_RE_DUP_MAX</code> .....                    | 1475            |

## Index

|                                                     |      |                                               |                                          |
|-----------------------------------------------------|------|-----------------------------------------------|------------------------------------------|
| <code>_SC_RTSIG_MAX</code> .....                    | 1475 | <code>_SC_XOPEN_REALTIME</code> .....         | 1475                                     |
| <code>_SC_SAVED_IDS</code> .....                    | 1474 | <code>_SC_XOPEN_REALTIME_THREADS</code> ..... | 1475                                     |
| <code>_SC_SEMAPHORES</code> .....                   | 1474 | <code>_SC_XOPEN_SHM</code> .....              | 1475                                     |
| <code>_SC_SEM_NSEMS_MAX</code> .....                | 1475 | <code>_SC_XOPEN_UNIX</code> .....             | 1475                                     |
| <code>_SC_SEM_VALUE_MAX</code> .....                | 1475 | <code>_SC_XOPEN_VERSION</code> .....          | 1475                                     |
| <code>_SC_SHARED_MEMORY_OBJECTS</code> .....        | 1474 | <code>_SC_XOPEN_XCU_VERSION</code> .....      | 1476                                     |
| <code>_SC_SHELL</code> .....                        | 1474 | <code>_setjmp</code> .....                    | 86                                       |
| <code>_SC_SIGQUEUE_MAX</code> .....                 | 1475 | <code>_t</code> .....                         | 17                                       |
| <code>_SC_SPAWN</code> .....                        | 1474 | <code>_TIME</code> .....                      | 17                                       |
| <code>_SC_SPIN_LOCKS</code> .....                   | 1474 | <code>_tolower()</code> .....                 | <b>88</b>                                |
| <code>_SC_SPORADIC_SERVER</code> .....              | 1474 | <code>_toupper()</code> .....                 | <b>89</b>                                |
| <code>_SC_STREAM_MAX</code> .....                   | 1475 | <code>_XBS5_ILP32_OFF32</code> .....          | 1475                                     |
| <code>_SC_SYMLOOP_MAX</code> .....                  | 1475 | <code>_XBS5_ILP32_OFFBIG</code> .....         | 1475                                     |
| <code>_SC_SYNCHRONIZED_IO</code> .....              | 1474 | <code>_XBS5_LP64_OFF64</code> .....           | 1475                                     |
| <code>_SC_THREADS</code> .....                      | 1474 | <code>_XBS5_LPBIG_OFFBIG</code> .....         | 1475                                     |
| <code>_SC_THREAD_ATTR_STACKADDR</code> .....        | 1474 | <code>_XOPEN_CRYPT</code> .....               | 1475                                     |
| <code>_SC_THREAD_ATTR_STACKSIZE</code> .....        | 1474 | <code>_XOPEN_ENH_I18N</code> .....            | 1475                                     |
| <code>_SC_THREAD_CPU_TIME</code> .....              | 1474 | <code>_XOPEN_LEGACY</code> .....              | 1475                                     |
| <code>_SC_THREAD_DESTRUCTOR_ITERATIONS</code> ..... | 1475 | <code>_XOPEN_REALTIME</code> .....            | 337, 1475                                |
| <code>_SC_THREAD_KEYS_MAX</code> .....              | 1475 | <code>_XOPEN_REALTIME_THREADS</code> .....    | 1475                                     |
| <code>_SC_THREAD_PRIORITY_SCHEDULING</code> .....   | 1474 | <code>_XOPEN_SHM</code> .....                 | 1475                                     |
| <code>_SC_THREAD_PRIO_INHERIT</code> .....          | 1474 | <code>_XOPEN_SOURCE</code> .....              | 14                                       |
| <code>_SC_THREAD_PRIO_PROTECT</code> .....          | 1474 | <code>_XOPEN_UNIX</code> .....                | 1475                                     |
| <code>_SC_THREAD_PROCESS_SHARED</code> .....        | 1474 | <code>_XOPEN_VERSION</code> .....             | 1475                                     |
| <code>_SC_THREAD_SAFE_FUNCTIONS</code> .....        | 1474 | <code>_XOPEN_XCU_VERSION</code> .....         | 1476                                     |
| <code>_SC_THREAD_SPORADIC_SERVER</code> .....       | 1474 | <code>a64l()</code> .....                     | <b>90</b>                                |
| <code>_SC_THREAD_STACK_MIN</code> .....             | 1475 | <code>ABDAY_1</code> .....                    | 834                                      |
| <code>_SC_THREAD_THREADS_MAX</code> .....           | 1475 | <code>abort()</code> .....                    | <b>92</b>                                |
| <code>_SC_TIMEOUTS</code> .....                     | 1474 | <code>abs()</code> .....                      | <b>94</b>                                |
| <code>_SC_TIMERS</code> .....                       | 1474 | <code>accept()</code> .....                   | <b>95</b>                                |
| <code>_SC_TIMER_MAX</code> .....                    | 1475 | <code>access()</code> .....                   | <b>97</b>                                |
| <code>_SC_TRACE</code> .....                        | 1474 | <code>acos()</code> .....                     | <b>100</b>                               |
| <code>_SC_TRACE_EVENT_FILTER</code> .....           | 1474 | <code>acosf</code> .....                      | 100                                      |
| <code>_SC_TRACE_INHERIT</code> .....                | 1474 | <code>acosh()</code> .....                    | <b>102</b>                               |
| <code>_SC_TRACE_LOG</code> .....                    | 1474 | <code>acoshf</code> .....                     | 102                                      |
| <code>_SC_TTY_NAME_MAX</code> .....                 | 1475 | <code>acoshl</code> .....                     | 102                                      |
| <code>_SC_TYPED_MEMORY_OBJECTS</code> .....         | 1474 | <code>acosl()</code> .....                    | 100, <b>104</b>                          |
| <code>_SC_TZNAME_MAX</code> .....                   | 1475 | <code>ACTION</code> .....                     | 584                                      |
| <code>_SC_V6_ILP32_OFF32</code> .....               | 1474 | <code>address information</code> .....        | 426                                      |
| <code>_SC_V6_ILP32_OFFBIG</code> .....              | 1474 | <code>address string</code> .....             | 426                                      |
| <code>_SC_V6_LP64_OFF64</code> .....                | 1474 | <code>addrinfo structure</code> .....         | 426                                      |
| <code>_SC_V6_LPBIG_OFFBIG</code> .....              | 1475 | <code>ADV</code> .....                        | <b>3</b>                                 |
| <code>_SC_VERSION</code> .....                      | 1474 | <code>ADVANCED REALTIME</code> .....          | 196, 200, 800-801, 864                   |
| <code>_SC_XBS5_ILP32_OFF32</code> .....             | 1475 | .....                                         | 866, 868, 870, 872, 875, 883, 886        |
| <code>_SC_XBS5_ILP32_OFFBIG</code> .....            | 1475 | .....                                         | 888-890, 892, 894, 896, 898, 900         |
| <code>_SC_XBS5_LP64_OFF64</code> .....              | 1475 | .....                                         | 902, 904-911, 964, 966, 1043, 1088, 1259 |
| <code>_SC_XBS5_LPBIG_OFFBIG</code> .....            | 1475 | <code>ADVANCED REALTIME THREADS</code> .....  | 1012, 1014                               |
| <code>_SC_XOPEN_CRYPT</code> .....                  | 1475 | .....                                         | 1016, 1018, 1020-1021, 1060, 1141        |
| <code>_SC_XOPEN_ENH_I18N</code> .....               | 1475 | .....                                         | 1143, 1145                               |
| <code>_SC_XOPEN_LEGACY</code> .....                 | 1475 | <code>AF</code> .....                         | 18                                       |
|                                                     |      | <code>AIO</code> .....                        | <b>3</b>                                 |

|                        |                         |                                |                                    |
|------------------------|-------------------------|--------------------------------|------------------------------------|
| AIO_                   | 16                      | atol()                         | 141                                |
| aio_                   | 16                      | atoll                          | 141                                |
| AIO_ALLDONE            | 105                     | attributes, clock-resolution   | 76, 914                            |
| aio_cancel()           | 105                     | attributes, creation-time      | 76, 914                            |
| AIO_CANCELED           | 105                     | attributes, generation-version | 75, 914                            |
| aio_error()            | 107                     | attributes, inheritance        | 76, 916                            |
| aio_fsync()            | 108                     | attributes, log-full-policy    | 74, 76, 916, 919                   |
| AIO_LISTIO_MAX         | 687, 1473               | attributes, log-max-size       | 76, 917, 919                       |
| AIO_MAX                | 687, 1473               | attributes, max-data-size      | 76, 919-920                        |
| AIO_NOTCANCELED        | 105                     | attributes, stream-full-policy | 72-73, 76, 917                     |
| AIO_PRIO_DELTA_MAX     | 42, 1473                | attributes, stream-min-size    | 76, 920                            |
| aio_read()             | 110                     | attributes, trace-name         | 76, 914                            |
| aio_return()           | 113                     | attributes, truncation-status  | 939                                |
| aio_suspend()          | 114                     | background                     | 1305                               |
| aio_write()            | 116                     | background process             | 1504                               |
| AI_ALL                 | 427                     | BAR                            | 3                                  |
| AI_CANONNAME           | 427                     | basename()                     | 142                                |
| AI_INET6               | 427                     | baud rate functions            | 180                                |
| AI_NUMERICHOST         | 427                     | bcmp()                         | 144                                |
| AI_NUMERICSERV         | 427                     | bcopy()                        | 145                                |
| AI_PASSIVE             | 427                     | BC_constants                   |                                    |
| AI_V4MAPPED            | 427                     | in sysconf                     | 1473                               |
| alarm()                | 119                     | BC_BASE_MAX                    | 1473                               |
| anycast                | 67                      | BC_DIM_MAX                     | 1473                               |
| ANYMARK                | 620                     | BC_SCALE_MAX                   | 1473                               |
| appropriate privileges | 98, 400                 | BC_STRING_MAX                  | 1473                               |
| argc                   | 302                     | BE                             | 4                                  |
| ARG_MAX                | 22, 297, 300, 304, 1473 | bind()                         | 146                                |
| asctime()              | 121                     | bi_                            | 16                                 |
| asctime_r              | 121                     | BOOT_TIME                      | 284-285                            |
| asin()                 | 123                     | broadcasting a condition       | 1030                               |
| asinf                  | 123                     | BSD                            | 119, 191, 310, 334, 343, 401       |
| asinh()                | 125                     |                                | 539, 669, 756, 823, 1176, 1216     |
| asinhf                 | 125                     |                                | 1224, 1305, 1348, 1372, 1496, 1519 |
| asinhf                 | 125                     |                                | 1557, 1592                         |
| asinhl                 | 125                     | bsd_signal()                   | 148                                |
| asinl()                | 123, 127                | bsearch()                      | 150                                |
| assert()               | 128                     | btowc()                        | 153                                |
| async-signal-safe      | 979                     | buffer cache                   | 452                                |
| atan()                 | 129                     | BUFSIZ                         | 1285                               |
| atan2()                | 131                     | BUS_                           | 18                                 |
| atan2f                 | 131                     | byte-oriented stream           | 36                                 |
| atan2l                 | 131                     | byte-stream mode               | 1173                               |
| atanf()                | 129, 133                | bzero()                        | 154                                |
| atanh()                | 134                     | cabs()                         | 155                                |
| atanhf                 | 134                     | cabsf                          | 155                                |
| atanhl                 | 134                     | cabsl                          | 155                                |
| atanl()                | 129, 136                | cacos()                        | 156                                |
| atexit()               | 137                     | cacosf                         | 156                                |
| ATEXIT_MAX             | 137, 1473               | cacosh()                       | 157                                |
| atof()                 | 138                     | cacoshf                        | 157                                |
| atoi()                 | 139                     |                                |                                    |

## Index

|                                        |                 |                                                    |                       |
|----------------------------------------|-----------------|----------------------------------------------------|-----------------------|
| cacoshl.....                           | 157             | change owner and group of file.....                | 191                   |
| acosl() .....                          | 156, <b>158</b> | CHAR_MAX .....                                     | 697, 699              |
| calloc() .....                         | <b>159</b>      | chdir() .....                                      | <b>185</b>            |
| can.....                               | 1               | CHILD_MAX .....                                    | 395, 1473             |
| cancel-safe.....                       | 1133            | chmod() .....                                      | <b>187</b>            |
| cancelability state.....               | 1067, 1133      | chown() .....                                      | <b>190</b>            |
| cancelability states.....              | 54              | cimag() .....                                      | <b>193</b>            |
| cancelability type .....               | 1067, 1133      | cimagf.....                                        | 193                   |
| canceled execution of a thread.....    | 1022            | cimagl.....                                        | 193                   |
| canonical name .....                   | 427             | CLD_ .....                                         | 18                    |
| carg().....                            | <b>161</b>      | clearerr().....                                    | <b>194</b>            |
| cargf.....                             | 161             | clock tick.....                                    | 119, 1477, 1529       |
| cargl.....                             | 161             | clock ticks/second.....                            | 1473                  |
| casin() .....                          | <b>162</b>      | clock() .....                                      | <b>195</b>            |
| casinf .....                           | 162             | clock-resolution attribute .....                   | 76, 914               |
| casinh().....                          | <b>163</b>      | CLOCKS_PER_SEC .....                               | 195                   |
| casinhf.....                           | 163             | CLOCK_ .....                                       | 17                    |
| casinhl.....                           | 163             | clock .....                                        | 17                    |
| casinl().....                          | 162, <b>164</b> | clock_getcpuclockid() .....                        | <b>196</b>            |
| catan().....                           | <b>165</b>      | clock_getres().....                                | <b>197</b>            |
| catanf.....                            | 165             | clock_gettime .....                                | 197                   |
| catanh() .....                         | <b>166</b>      | CLOCK_MONOTONIC .....                              | 49, 201               |
| catanhf.....                           | 166             | clock_nanosleep() .....                            | <b>200</b>            |
| catanhl.....                           | 166             | CLOCK_PROCESS_CPUTIME_ID.....                      | 49                    |
| catanl() .....                         | 165, <b>167</b> | CLOCK_REALTIME.....                                | 49, 197, 201          |
| catclose() .....                       | <b>168</b>      | .....                                              | 823, 1088, 1259, 1521 |
| catgets().....                         | <b>169</b>      | clock_settime() .....                              | 197, <b>203</b>       |
| catopen() .....                        | <b>171</b>      | CLOCK_THREAD_CPUTIME_ID.....                       | 50                    |
| CBAUD .....                            | 19              | clog() .....                                       | <b>204</b>            |
| cbrt() .....                           | <b>173</b>      | clogf.....                                         | 204                   |
| cbrtf.....                             | 173             | clogl.....                                         | 204                   |
| cbrtl.....                             | 173             | close a file .....                                 | 207                   |
| ccos().....                            | <b>174</b>      | close() .....                                      | <b>205</b>            |
| ccosf.....                             | 174             | closedir() .....                                   | <b>208</b>            |
| ccosh() .....                          | <b>175</b>      | closelog().....                                    | <b>210</b>            |
| ccoshf .....                           | 175             | cmsg_ .....                                        | 16                    |
| ccoshl.....                            | 175             | CMSG_.....                                         | 18                    |
| ccosl() .....                          | 174, <b>176</b> | COLL_WEIGHTS_MAX .....                             | 1473                  |
| CD .....                               | <b>4</b>        | command interpreter                                |                       |
| ceil() .....                           | <b>177</b>      | portable .....                                     | 1592                  |
| ceilf .....                            | 177             | compare thread IDs.....                            | 1055                  |
| ceill.....                             | 177             | compilation environment .....                      | <b>13</b>             |
| cexp() .....                           | <b>179</b>      | condition variable initialization attributes ..... | 1041                  |
| cexpf.....                             | 179             | conforming application.....                        | 1389                  |
| cexpl .....                            | 179             | conforming application, strictly.....              | 119, 302              |
| cfgetispeed().....                     | <b>180</b>      | confstr() .....                                    | <b>214</b>            |
| cfgetospeed().....                     | <b>182</b>      | conj() .....                                       | <b>217</b>            |
| cfsetispeed() .....                    | <b>183</b>      | conjf.....                                         | 217                   |
| cfsetospeed() .....                    | <b>184</b>      | conjl .....                                        | 217                   |
| change current working directory ..... | 186             | connect() .....                                    | <b>218</b>            |
| change file modes.....                 | 189             | control data .....                                 | 38                    |

|                                              |                       |                                     |                                            |
|----------------------------------------------|-----------------------|-------------------------------------|--------------------------------------------|
| control-normal .....                         | 1173                  | ctanl() .....                       | 238, 240                                   |
| conversion descriptor.....                   | 297, 302, 590-593     | ctermid() .....                     | 241                                        |
| conversion specification .....               | 404, 435, 468         | ctime() .....                       | 243                                        |
| modified .....                               | 477, 1424, 1428, 1441 | ctime_r .....                       | 243                                        |
| conversion specifier                         |                       | CX.....                             | 4                                          |
| modified .....                               | 1442                  | c_.....                             | 17                                         |
| copysign() .....                             | 221                   | data key creation .....             | 1071                                       |
| copysignf .....                              | 221                   | data messages.....                  | 38                                         |
| copysignl .....                              | 221                   | data type.....                      | 80                                         |
| core .....                                   | 1593                  | DATEMSK.....                        | 495                                        |
| core file.....                               | 309                   | daylight .....                      | 245, 1549                                  |
| cos().....                                   | 222                   | DBL_MANT_DIG .....                  | 177, 377                                   |
| cosf.....                                    | 222                   | DBL_MAX_EXP.....                    | 177, 377                                   |
| cosh() .....                                 | 224                   | DBM.....                            | 246-247                                    |
| coshf .....                                  | 224                   | dbm_.....                           | 16                                         |
| coshl.....                                   | 224                   | DBM_.....                           | 18                                         |
| cosl() .....                                 | 222, 226              | dbm_clearerr().....                 | 246                                        |
| covert channel.....                          | 669                   | dbm_close .....                     | 246                                        |
| cpow() .....                                 | 227                   | dbm_delete .....                    | 246                                        |
| cpowf.....                                   | 227                   | dbm_error .....                     | 246                                        |
| cpowl.....                                   | 227                   | dbm_fetch .....                     | 246                                        |
| cproj() .....                                | 228                   | dbm_firstkey.....                   | 246                                        |
| cprojf .....                                 | 228                   | DBM_INSERT .....                    | 246                                        |
| cprojl.....                                  | 228                   | dbm_nextkey.....                    | 246                                        |
| CPT .....                                    | 4                     | dbm_open .....                      | 246                                        |
| creal() .....                                | 229                   | DBM_REPLACE .....                   | 246                                        |
| crealf.....                                  | 229                   | dbm_store .....                     | 246                                        |
| creall.....                                  | 229                   | DEAD_PROCESS.....                   | 284-285                                    |
| creat().....                                 | 230                   | DEFECHO .....                       | 19                                         |
| create a per-process timer.....              | 1522                  | deferred cancelability .....        | 1067                                       |
| create an interprocess channel.....          | 856                   | delay process execution.....        | 1388                                       |
| create session and set process group ID..... | 1317                  | DELAYTIMER_MAX.....                 | 1473, 1526                                 |
| creation-time attribute .....                | 76, 914               | dependency ordering.....            | 259                                        |
| CRYPT.....                                   | 232, 270, 1298        | descriptive name .....              | 426                                        |
| crypt() .....                                | 232                   | destroying a mutex .....            | 1077                                       |
| CS .....                                     | 4                     | destroying condition variables..... | 1034                                       |
| csin() .....                                 | 234                   | destructor functions.....           | 1071                                       |
| csinf .....                                  | 234                   | detaching a thread.....             | 1053                                       |
| csinh() .....                                | 235                   | difftime().....                     | 250                                        |
| csinhf.....                                  | 235                   | DIR .....                           | 80, 208, 845, 1179, 1181, 1219, 1245, 1512 |
| csinhl .....                                 | 235                   | directive.....                      | 404, 435, 468, 477, 1441                   |
| csinl().....                                 | 234, 236              | directory operations.....           | 846                                        |
| csqrt().....                                 | 237                   | dirent structure .....              | 846                                        |
| csqrtf.....                                  | 237                   | dirname() .....                     | 251                                        |
| csqrtl.....                                  | 237                   | div().....                          | 253                                        |
| ctan().....                                  | 238                   | dlclose().....                      | 254                                        |
| ctanf.....                                   | 238                   | dLError().....                      | 256                                        |
| ctanh() .....                                | 239                   | dlopen().....                       | 258                                        |
| ctanhf.....                                  | 239                   | dlsym() .....                       | 261                                        |
| ctanhl.....                                  | 239                   | dot.....                            | 846, 1216                                  |
|                                              |                       | dot-dot.....                        | 846, 1216                                  |

## Index

|                                |               |                 |          |
|--------------------------------|---------------|-----------------|----------|
| drand48()                      | 263           | encrypt()       | 270      |
| dup()                          | 265           | endgrent()      | 272      |
| dup2                           | 265           | endhostent()    | 274      |
| dynamic package initialization | 1109          | endnetent()     | 276      |
| d_                             | 16            | endprotoent()   | 278      |
| E2BIG                          | 22            | endpwent()      | 280      |
| EACCESS                        | 22            | endservent()    | 282      |
| EADDRINUSE                     | 22            | endutxent()     | 284      |
| EADDRNOTAVAIL                  | 22            | ENETDOWN        | 25       |
| EAFNOSUPPORT                   | 22            | ENETRESET       | 25       |
| EAGAIN                         | 22, 28        | ENETUNREACH     | 25       |
| EALREADY                       | 22            | ENFILE          | 25       |
| EBADF                          | 22            | ENOBUFS         | 25       |
| EBADMSG                        | 22            | ENODATA         | 25       |
| EBUSY                          | 23            | ENODEV          | 25       |
| ECANCELED                      | 23            | ENOENT          | 25       |
| ECHILD                         | 23            | ENOEXEC         | 25       |
| ECHOCTL                        | 19            | ENOLCK          | 25       |
| ECHOKE                         | 19            | ENOLINK         | 25       |
| ECHOPRT                        | 19            | ENOMEM          | 25       |
| ECONNABORTED                   | 23            | ENOMSG          | 25       |
| ECONNREFUSED                   | 23            | ENOPROTOPT      | 25       |
| ECONNRESET                     | 23            | ENOSPC          | 26       |
| ecvt()                         | 268           | ENOSR           | 26       |
| EDEADLK                        | 23            | ENOSTR          | 26       |
| EDESTADDRREQ                   | 23            | ENOSYS          | 26       |
| EDOM                           | 23            | ENOTCONN        | 26       |
| EDQUOT                         | 23            | ENOTDIR         | 26       |
| EEXIST                         | 23            | ENOTEMPTY       | 26       |
| EFAULT                         | 23            | ENOTSOCK        | 26       |
| EFBIG                          | 23            | ENOTSUP         | 26       |
| effective group ID             | 191, 304, 513 | ENOTTY          | 26       |
| effective user ID              | 98, 304, 669  | ENTRY           | 584      |
| EHOSTUNREACH                   | 23            | environ         | 287, 303 |
| EIDRM                          | 23            | envp            | 303      |
| Eighth Edition UNIX            | 1661          | ENXIO           | 26       |
| EILSEQ                         | 24, 37        | EOPNOTSUPP      | 26       |
| EINPROGRESS                    | 24, 42        | EOVERFLOW       | 26       |
| EINTR                          | 24, 57, 933   | EPERM           | 26       |
| EINVAL                         | 24, 920, 933  | EPIPE           | 26       |
| EIO                            | 24            | EPROTO          | 27       |
| EISCONN                        | 24            | EPROTONOSUPPORT | 27       |
| EISDIR                         | 24            | EPROTOTYPE      | 27       |
| ELOOP                          | 24            | erand48()       | 263, 288 |
| ELSIZE                         | 726           | ERANGE          | 27       |
| EMFILE                         | 24            | erf()           | 289      |
| EMLINK                         | 24            | erfc()          | 291      |
| EMPTY                          | 285           | erfcf           | 291      |
| EMSGSIZE                       | 24            | erfcl           | 291      |
| EMULTIHOP                      | 24            | erff()          | 289, 293 |
| ENAMETOOLONG                   | 25            | erfl            | 289, 293 |

|                                          |                                         |                          |                                  |
|------------------------------------------|-----------------------------------------|--------------------------|----------------------------------|
| EROFS .....                              | 27                                      | fchdir() .....           | 322                              |
| errno .....                              | 294                                     | fchmod() .....           | 323                              |
| error descriptions .....                 | 483                                     | fchown() .....           | 325                              |
| error numbers .....                      | 21                                      | fclose() .....           | 327                              |
| additional .....                         | 28                                      | fcntl() .....            | 329                              |
| ESPIPE .....                             | 27                                      | fcvt() .....             | 268, 336                         |
| ESRCH .....                              | 27                                      | FD .....                 | 4                                |
| EST5EDT .....                            | 1549                                    | fdatasync() .....        | 337                              |
| ESTALE .....                             | 27                                      | fdetach() .....          | 338                              |
| ETIME .....                              | 27                                      | fdim() .....             | 340                              |
| ETIMEDOUT .....                          | 27                                      | fdimf .....              | 340                              |
| ETXTBSY .....                            | 27                                      | fdiml .....              | 340                              |
| EWOULDBLOCK .....                        | 27                                      | fdopen() .....           | 342                              |
| examine and change blocked signals ..... | 1140                                    | fds_ .....               | 16                               |
| examine and change signal action .....   | 1348                                    | FD_ .....                | 16, 18                           |
| EXDEV .....                              | 28                                      | fd .....                 | 16                               |
| exec .....                               | 296                                     | FD_CLOEXEC .....         | 35, 171, 297, 329, 593           |
| of shell scripts .....                   | 303                                     | .....                    | 838, 846, 856, 876, 883, 1329    |
| exec family ...                          | 98, 207, 333, 374, 397, 979, 1306, 1592 | FD_CLR() .....           | 84, 974                          |
| execl .....                              | 296                                     | FD_ISSET .....           | 84, 974                          |
| execle .....                             | 296                                     | FD_SET .....             | 84, 974                          |
| execlp .....                             | 296                                     | FD_ZERO .....            | 84, 974                          |
| execute a file .....                     | 302                                     | feature test macro ..... | 13, 488                          |
| execution time monitoring .....          | 49                                      | _POSIX_C_SOURCE .....    | 14                               |
| execv .....                              | 296                                     | _XOPEN_SOURCE .....      | 14                               |
| execve .....                             | 296                                     | feclearexcept() .....    | 344                              |
| execvp .....                             | 296                                     | fegetenv() .....         | 345                              |
| exit() .....                             | 307                                     | fegetexceptflag() .....  | 346                              |
| EXIT_FAILURE .....                       | 307                                     | fegetround() .....       | 347                              |
| EXIT_SUCCESS .....                       | 307, 310                                | feholdexcept() .....     | 349                              |
| exp() .....                              | 312                                     | feof() .....             | 350                              |
| exp2() .....                             | 314                                     | feraiseexcept() .....    | 351                              |
| exp2f .....                              | 314                                     | ferror() .....           | 352                              |
| exp2l .....                              | 314                                     | fesetenv() .....         | 345, 353                         |
| expf .....                               | 312                                     | fesetexceptflag() .....  | 346, 354                         |
| expl .....                               | 312                                     | fesetround() .....       | 347, 355                         |
| expm1() .....                            | 316                                     | fetestexcept() .....     | 356                              |
| expm1f .....                             | 316                                     | feupdateenv() .....      | 358                              |
| expm1l .....                             | 316                                     | fflush() .....           | 360                              |
| EXPR_NEST_MAX .....                      | 1473                                    | ffs() .....              | 363                              |
| EXTA .....                               | 19                                      | fgetc() .....            | 364                              |
| EXTB .....                               | 19                                      | fgetpos() .....          | 366                              |
| extension                                |                                         | fgets() .....            | 368                              |
| CX .....                                 | 4                                       | fgetwc() .....           | 370                              |
| OH .....                                 | 7                                       | fgetws() .....           | 372                              |
| XSI .....                                | 11                                      | FIFO .....               | 758-759, 842, 1660               |
| extensions to setlocale .....            | 1300                                    | FILE .....               | 80, 194, 327, 342, 350, 352      |
| fabs() .....                             | 318                                     | .....                    | 360, 364, 366, 368, 370, 372     |
| fabsf .....                              | 318                                     | .....                    | 374-375, 390, 404, 416, 418, 420 |
| fabsl .....                              | 318                                     | .....                    | 422-423, 430, 435, 442, 445, 454 |
| fattach() .....                          | 319                                     | .....                    | 466-468, 475, 477, 486-487, 573  |

## Index

|                                         |                     |
|-----------------------------------------|---------------------|
| .....852, 862, 1148-1149, 1161, 1218    |                     |
| .....1285, 1327, 1409, 1531, 1559-1560  |                     |
| .....1577, 1579, 1581, 1583, 1585, 1587 |                     |
| file                                    |                     |
| locking .....                           | 332                 |
| file accessibility .....                | 98                  |
| file control .....                      | 332                 |
| FILE object.....                        | 34                  |
| file permission bits .....              | 98                  |
| file permissions.....                   | 98, 401, 1406       |
| file position indicator .....           | 34                  |
| fileno() .....                          | 374                 |
| FILESIZEBITS.....                       | 399                 |
| FIND.....                               | 584                 |
| find string token.....                  | 1454                |
| flockfile() .....                       | 375                 |
| floor() .....                           | 377                 |
| floorf .....                            | 377                 |
| floorl .....                            | 377                 |
| FLT_RADIX.....                          | 716                 |
| FLT_ROUNDS.....                         | 379                 |
| FLUSH.....                              | 18                  |
| FLUSHO .....                            | 19                  |
| FLUSHR.....                             | 614                 |
| FLUSHRW .....                           | 614                 |
| FLUSHW .....                            | 614                 |
| fma() .....                             | 379                 |
| fmaf.....                               | 379                 |
| fmal.....                               | 379                 |
| fmax() .....                            | 381                 |
| fmaxf .....                             | 381                 |
| fmaxl.....                              | 381                 |
| fmin() .....                            | 382                 |
| fminf .....                             | 382                 |
| fminl .....                             | 382                 |
| FMNAMESZ.....                           | 613                 |
| fmod().....                             | 383                 |
| fmodf.....                              | 383                 |
| fmodl.....                              | 383                 |
| fmtmsg() .....                          | 385                 |
| fnmatch() .....                         | 388                 |
| FNM_.....                               | 18                  |
| FNM_NOESCAPE.....                       | 388                 |
| FNM_NOMATCH.....                        | 388                 |
| FNM_PATHNAME.....                       | 388                 |
| FNM_PERIOD.....                         | 388                 |
| fopen() .....                           | 390                 |
| FOPEN_MAX.....                          | 342, 391, 862, 1531 |
| foreground .....                        | 1305                |
| fork handler .....                      | 980                 |
| fork() .....                            | 394                 |
| forkall .....                           | 397                 |
| format of entries.....                  | 11                  |
| fpathconf().....                        | 399                 |
| fpclassify() .....                      | 403                 |
| FPE_.....                               | 18                  |
| fprintf().....                          | 404                 |
| fputc().....                            | 416                 |
| fputs() .....                           | 418                 |
| fputwc() .....                          | 420                 |
| fputws().....                           | 422                 |
| FQDN.....                               | 528                 |
| FR .....                                | 4                   |
| fread() .....                           | 423                 |
| free().....                             | 425                 |
| freeaddrinfo().....                     | 426                 |
| freopen() .....                         | 430                 |
| frexp() .....                           | 433                 |
| frexpf .....                            | 433                 |
| frexpl .....                            | 433                 |
| FSC.....                                | 5                   |
| fscanf().....                           | 435                 |
| fseek() .....                           | 442                 |
| fseeko .....                            | 442                 |
| fsetpos() .....                         | 445                 |
| fstat().....                            | 447                 |
| fstatvfs() .....                        | 449                 |
| fsync().....                            | 452                 |
| ftell() .....                           | 454                 |
| ftello.....                             | 454                 |
| ftime().....                            | 456                 |
| ftok() .....                            | 458                 |
| ftruncate() .....                       | 460                 |
| ftrylockfile() .....                    | 375, 462            |
| FTW .....                               | 18, 829-830         |
| ftw() .....                             | 463                 |
| FTW_CHDIR.....                          | 829                 |
| FTW_D .....                             | 463, 829            |
| FTW_DEPTH.....                          | 829                 |
| FTW_DNR .....                           | 463, 829-830        |
| FTW_DP .....                            | 829                 |
| FTW_F .....                             | 463, 829            |
| FTW_MOUNT .....                         | 829                 |
| FTW_NS .....                            | 463, 829-830        |
| FTW_PHYS.....                           | 829                 |
| FTW_SL.....                             | 463, 829            |
| FTW_SLN.....                            | 829                 |
| fully-qualified domain name .....       | 528                 |
| functions .....                         | 13                  |
| implementation.....                     | 13                  |
| use.....                                | 13                  |
| funlockfile().....                      | 375, 466            |

|                                           |                   |                                     |                 |
|-------------------------------------------|-------------------|-------------------------------------|-----------------|
| <code>fwide()</code> .....                | <b>467</b>        | <code>getgrnam()</code> .....       | <b>510</b>      |
| <code>fwprintf()</code> .....             | <b>468</b>        | <code>getgrnam_r</code> .....       | 510             |
| <code>fwrite()</code> .....               | <b>475</b>        | <code>getgroups()</code> .....      | <b>512</b>      |
| <code>fwscanf()</code> .....              | <b>477</b>        | <code>gethostbyaddr()</code> .....  | <b>514</b>      |
| <code>f_</code> .....                     | 16                | <code>gethostbyname</code> .....    | 514             |
| <code>F_</code> .....                     | 18                | <code>gethostent()</code> .....     | 274, <b>516</b> |
| <code>F_DUPFD</code> .....                | 265, 329, 331-332 | <code>gethostid()</code> .....      | <b>517</b>      |
| <code>F_GETFD</code> .....                | 329, 331-332      | <code>gethostname()</code> .....    | <b>518</b>      |
| <code>F_GETFL</code> .....                | 329, 331-332      | <code>getitimer()</code> .....      | <b>519</b>      |
| <code>F_GETLK</code> .....                | 330-332           | <code>getlogin()</code> .....       | <b>521</b>      |
| <code>F_GETOWN</code> .....               | 329, 331          | <code>getlogin_r</code> .....       | 521             |
| <code>F_LOCK</code> .....                 | 705               | <code>getmsg()</code> .....         | <b>524</b>      |
| <code>F_RDLCK</code> .....                | 332               | <code>getnameinfo()</code> .....    | <b>528</b>      |
| <code>F_SETFD</code> .....                | 329, 331-333      | <code>GETNCNT</code> .....          | 1266-1267       |
| <code>F_SETFL</code> .....                | 329, 331-332      | <code>getnetbyaddr()</code> .....   | 276, <b>531</b> |
| <code>F_SETLK</code> .....                | 330-332           | <code>getnetbyname</code> .....     | 276, 531        |
| <code>F_SETLKW</code> .....               | 55, 330-332       | <code>getnetent</code> .....        | 276, 531        |
| <code>F_SETOWN</code> .....               | 329, 332          | <code>getopt()</code> .....         | <b>532</b>      |
| <code>F_TEST</code> .....                 | 705               | <code>getpeername()</code> .....    | <b>537</b>      |
| <code>F_TLOCK</code> .....                | 705               | <code>getpgid()</code> .....        | <b>538</b>      |
| <code>F_ULOCK</code> .....                | 705               | <code>getpgrp()</code> .....        | <b>539</b>      |
| <code>F_UNLCK</code> .....                | 330-331           | <code>GETPID</code> .....           | 1266-1267       |
| <code>F_WRLCK</code> .....                | 332               | <code>getpid()</code> .....         | <b>540</b>      |
| <code>gai_strerror()</code> .....         | <b>483</b>        | <code>getpmsg()</code> .....        | 524, <b>541</b> |
| <code>gcvt()</code> .....                 | 268, <b>484</b>   | <code>getppid()</code> .....        | <b>542</b>      |
| generation-version attribute .....        | 75, 914           | <code>getpriority()</code> .....    | <b>543</b>      |
| get configurable pathname variables ..... | 401               | <code>getprotent</code> .....       | 546             |
| get configurable system variables .....   | 1476              | <code>getprotobyname()</code> ..... | 278, <b>546</b> |
| get file status .....                     | 1406              | <code>getprotobynumber</code> ..... | 278, 546        |
| get process times .....                   | 1529              | <code>getprotoent</code> .....      | 278             |
| get supplementary group IDs .....         | 512               | <code>getpwent()</code> .....       | 280, <b>547</b> |
| get system time .....                     | 1519              | <code>getpwnam()</code> .....       | <b>548</b>      |
| get thread ID .....                       | 1131              | <code>getpwnam_r</code> .....       | 548             |
| get user name .....                       | 522               | <code>getpwuid()</code> .....       | <b>551</b>      |
| <code>getaddrinfo()</code> .....          | 426, <b>485</b>   | <code>getpwuid_r</code> .....       | 551             |
| <code>GETALL</code> .....                 | 1266              | <code>getrlimit()</code> .....      | <b>554</b>      |
| <code>getc()</code> .....                 | <b>486</b>        | <code>getrusage()</code> .....      | <b>557</b>      |
| <code>getchar()</code> .....              | <b>489</b>        | <code>gets()</code> .....           | <b>559</b>      |
| <code>getchar_unlocked()</code> .....     | 487, <b>490</b>   | <code>getservbyname()</code> .....  | 282, <b>560</b> |
| <code>getcontext()</code> .....           | <b>491</b>        | <code>getservbyport</code> .....    | 282, 560        |
| <code>getcwd()</code> .....               | <b>493</b>        | <code>getservent</code> .....       | 282, 560        |
| <code>getc_unlocked()</code> .....        | <b>487</b>        | <code>getsid()</code> .....         | <b>561</b>      |
| <code>getdate()</code> .....              | <b>495</b>        | <code>getsockname()</code> .....    | <b>562</b>      |
| <code>getdate_err</code> .....            | 495               | <code>getsockopt()</code> .....     | <b>563</b>      |
| <code>getegid()</code> .....              | <b>500</b>        | <code>getsubopt()</code> .....      | <b>566</b>      |
| <code>getenv()</code> .....               | 303, <b>501</b>   | <code>gettimeofday()</code> .....   | <b>570</b>      |
| <code>geteuid()</code> .....              | <b>504</b>        | <code>getuid()</code> .....         | <b>571</b>      |
| <code>getgid()</code> .....               | <b>505</b>        | <code>getutxent</code> .....        | 284             |
| <code>getgrent()</code> .....             | 272, <b>506</b>   | <code>getutxent()</code> .....      | <b>572</b>      |
| <code>getgrgid()</code> .....             | <b>507</b>        | <code>getutxid</code> .....         | 284, 572        |
| <code>getgrgid_r</code> .....             | 507               | <code>getutxline</code> .....       | 284, 572        |

## Index

|                            |                                     |                                       |                  |
|----------------------------|-------------------------------------|---------------------------------------|------------------|
| GETVAL.....                | 1266-1267                           | ifc.....                              | 16               |
| getwc().....               | 573                                 | ifra.....                             | 16               |
| getwchar().....            | 574                                 | ifru.....                             | 16               |
| getwd().....               | 494, 575                            | if.....                               | 16               |
| GETZCNT.....               | 1266-1267                           | IF.....                               | 18               |
| glob().....                | 576                                 | if_freenameindex().....               | 595              |
| globfree.....              | 576                                 | if_indextoname().....                 | 596              |
| GLOB.....                  | 18                                  | if_nameindex().....                   | 597              |
| GLOB_constants             |                                     | if_nametoindex().....                 | 598              |
| error returns of glob..... | 578                                 | ILL.....                              | 18               |
| used in glob.....          | 576                                 | ilogb().....                          | 599              |
| GLOB_ABORTED.....          | 578                                 | ilogbf.....                           | 599              |
| GLOB_APPEND.....           | 576-577                             | ilogbl.....                           | 599              |
| GLOB_DOOFFS.....           | 576-577                             | imaxabs().....                        | 601              |
| GLOB_ERR.....              | 576, 578                            | imaxdiv().....                        | 602              |
| GLOB_MARK.....             | 577                                 | implementation-defined.....           | 1                |
| GLOB_NOCHECK.....          | 577-578                             | IMPLINK.....                          | 18               |
| GLOB_NOESCAPE.....         | 577                                 | in6.....                              | 16               |
| GLOB_NOMATCH.....          | 578                                 | IN6.....                              | 18               |
| GLOB_NOSORT.....           | 577                                 | INADDR.....                           | 18               |
| GLOB_NOSPACE.....          | 578                                 | index().....                          | 603              |
| gl.....                    | 16                                  | inet.....                             | 16               |
| GMT0.....                  | 1549                                | inet_addr().....                      | 604              |
| gmtime().....              | 580                                 | inet_ntoa.....                        | 604              |
| gmtime_r.....              | 580                                 | inet_ntop().....                      | 606              |
| grantpt().....             | 582                                 | inet_pton.....                        | 606              |
| granularity of clock.....  | 456                                 | Inf.....                              | 123              |
| HALT.....                  | 386                                 | INF.....                              | 407, 471         |
| hcreate().....             | 584                                 | INFINITY.....                         | 407, 471         |
| hdestroy.....              | 584                                 | INFO.....                             | 386              |
| high resolution sleep..... | 823                                 | infu.....                             | 16               |
| host name.....             | 426                                 | inheritance attribute.....            | 76, 916          |
| htonl().....               | 587                                 | init.....                             | 310, 669         |
| htons.....                 | 587                                 | initialize a named semaphore.....     | 1255             |
| HUGE_VAL.....              | 134, 177, 224, 312, 314             | initialize an unnamed semaphore.....  | 1252             |
| .....                      | 316, 340, 377, 588, 677, 681, 708   | initializing a mutex.....             | 1077             |
| .....                      | 712, 714, 825, 827, 969, 1221, 1226 | initializing condition variables..... | 1034             |
| .....                      | 1230, 1385, 1449, 1485, 1516, 1619  | initstate().....                      | 608              |
| HUGE_VALF.....             | 377, 1221, 1449                     | INIT_PROCESS.....                     | 284-285          |
| HUGE_VALL.....             | 377, 1221, 1449                     | input and output rationale.....       | 1175             |
| hypot().....               | 588                                 | insque().....                         | 610              |
| hypotf.....                | 588                                 | international environment.....        | 1300             |
| hypotl.....                | 588                                 | Internet Protocols.....               | 66               |
| h.....                     | 16                                  | INT_MAX.....                          | 599              |
| h_errno.....               | 583                                 | INT_MIN.....                          | 94               |
| iconv().....               | 590                                 | in.....                               | 16               |
| iconv_close().....         | 592                                 | IN.....                               | 18               |
| iconv_open().....          | 593                                 | ioctl().....                          | 613              |
| ic.....                    | 16                                  | iov.....                              | 17               |
| IEEE Std 754-1985.....     | 3                                   | IOV.....                              | 18               |
| IEEE Std 854-1987.....     | 3                                   | IOV_MAX.....                          | 1186, 1473, 1664 |

- IP6 .....5
- IPC.....39, 806, 808, 811, 813, 1271, 1275, 1339, 1341
- ipc\_.....16
- IPC\_.....18
- IPC\_ constants
  - used in semctl .....1266
  - used in shmctl.....1337
- IPC\_CREAT.....807, 1269, 1340
- IPC\_EXCL.....807, 1269
- IPC\_NOWAIT.....809-810, 812-813, 1272
- IPC\_PRIVATE.....807, 1269, 1340
- IPC\_RMID.....805, 1267, 1337
- IPC\_SET.....805, 1267, 1337
- IPC\_STAT.....805, 1266, 1337
- IPPORT\_.....18
- IPPROTO\_.....18
- IPv4.....67
- IPv4-compatible address.....68
- IPv4-mapped address.....68
- IPv6.....67
  - compatibility with IPv4.....68
  - interface identification.....68
  - options.....69
- IPv6 address
  - anycast.....67
  - loopback.....68
  - multicast.....67
  - unicast.....67
  - unspecified.....68
- IPV6\_.....18
- IPV6\_JOIN\_GROUP.....69
- IPV6\_LEAVE\_GROUP.....69
- IPV6\_MULTICAST\_HOPS.....69
- IPV6\_MULTICAST\_IF.....69
- IPV6\_MULTICAST\_LOOP.....69
- IPV6\_UNICAST\_HOPS.....69
- IPV6\_V6ONLY.....69
- ip\_.....16
- IP\_.....18
- isalnum().....625
- isalpha().....626
- isascii().....627
- isastream().....628
- isatty().....629
- isblank().....630
- iscntrl().....631
- isdigit().....632
- isfinite().....633
- isgraph().....634
- isgreater().....635
- isgreaterequal().....636
- isinf().....637
- isless().....638
- islessequal().....639
- islessgreater().....640
- islower().....641
- isnan().....643
- isnormal().....644
- ISO C standard.....3, 119, 302, 332
  - .....488, 728, 1169, 1216, 1300, 1348
  - .....1372, 1519
- isprint().....645
- ispunct().....646
- isspace().....647
- isunordered().....648
- isupper().....649
- iswalnum().....650
- iswalpha().....651
- iswblank().....652
- iswcntrl().....653
- iswctype().....654
- iswdigit().....656
- iswgraph().....657
- iswlower().....658
- iswprint().....659
- iswpunct().....660
- iswspace().....661
- iswupper().....662
- iswxdigit().....663
- isxdigit().....664
- ITIMER\_PROF.....519
- ITIMER\_REAL.....519
- ITIMER\_VIRTUAL.....519
- it\_.....16-17
- I\_.....18
- I\_ATMARK.....619-620
- I\_CANPUT.....620
- I\_CKBAND.....620
- I\_FDINSERT.....616
- I\_FIND.....615
- I\_FLUSH.....613
- I\_FLUSHBAND.....614
- I\_GETBAND.....620
- I\_GETCLTIME.....620
- I\_GETSIG.....615
- I\_GRDOPT.....616, 1173
- I\_GWROPT.....618
- I\_LINK.....621
- I\_LIST.....619
- I\_LOOK.....613
- I\_NREAD.....616
- I\_PEEK.....615

## Index

|                     |                                       |                                    |                       |
|---------------------|---------------------------------------|------------------------------------|-----------------------|
| I_PLINK.....        | 622                                   | link().....                        | <b>683</b>            |
| I_POP.....          | 613                                   | LINK_MAX.....                      | 24, 399, 683, 1215    |
| I_PUNLINK.....      | 622                                   | LIO_.....                          | 16                    |
| I_PUSH.....         | 613                                   | lio_.....                          | 16                    |
| I_RECVFD.....       | 22, 619                               | lio_listio().....                  | <b>686</b>            |
| I_SENDFD.....       | 618-619                               | LIO_NOP.....                       | 686                   |
| I_SETCLTIME.....    | 205, 620                              | LIO_NOWAIT.....                    | 686                   |
| I_SETSIG.....       | 614-615                               | LIO_READ.....                      | 686                   |
| I_SRDOPT.....       | 615-616, 1173                         | LIO_WAIT.....                      | 686                   |
| I_STR.....          | 617                                   | LIO_WRITE.....                     | 686                   |
| I_SWROPT.....       | 618, 1658                             | list directed I/O.....             | 688                   |
| I_UNLINK.....       | 621                                   | listen().....                      | <b>689</b>            |
| j0().....           | <b>665</b>                            | llabs().....                       | 673, <b>691</b>       |
| j1.....             | 665                                   | lldiv().....                       | 679, <b>692</b>       |
| jn.....             | 665                                   | LLONG_MAX.....                     | 1457, 1625            |
| job control.....    | 310, 539, 669, 1305, 1317, 1476, 1592 | LLONG_MIN.....                     | 1457                  |
| rand48().....       | 263, <b>667</b>                       | llrint().....                      | <b>693</b>            |
| JST-9.....          | 1549                                  | llrintf.....                       | 693                   |
| kill().....         | <b>668</b>                            | llrintl.....                       | 693                   |
| killpg().....       | <b>671</b>                            | llround().....                     | <b>695</b>            |
| l64a().....         | 90, <b>672</b>                        | llroundf.....                      | 695                   |
| labs().....         | <b>673</b>                            | llroundl.....                      | 695                   |
| LANG.....           | 171                                   | load ordering.....                 | 259                   |
| last close.....     | 1333                                  | LOBLK.....                         | 19                    |
| LASTMARK.....       | 620                                   | localeconv().....                  | <b>697</b>            |
| lchown().....       | <b>674</b>                            | localtime().....                   | <b>702</b>            |
| lcong48().....      | 263, <b>676</b>                       | localtime_r.....                   | 702                   |
| LC_ALL.....         | 297, 699, 834, 1299, 1301             | lockf().....                       | <b>705</b>            |
| LC_COLLATE.....     | 576-577, 1299-1300, 1416              | locking.....                       | 332                   |
| .....               | 1464, 1603, 1637                      | advisory.....                      | 333                   |
| LC_CTYPE.....       | 153, 654, 736, 738                    | mandatory.....                     | 333                   |
| .....               | 740, 742-743, 745, 747, 1299-1300     | locking and unlocking a mutex..... | 1085                  |
| .....               | 1536-1540, 1598, 1614, 1629           | log().....                         | <b>708</b>            |
| .....               | 1639-1640, 1642-1643                  | log-full-policy attribute.....     | 74, 76, 916, 919, 937 |
| LC_MESSAGES.....    | 171, 1299-1300, 1422                  | log-max-size attribute.....        | 76, 917, 919          |
| LC_MONETARY.....    | 699, 1299-1300, 1425                  | log10().....                       | <b>710</b>            |
| LC_NUMERIC.....     | 268, 405, 435, 468, 477               | log10f.....                        | 710                   |
| .....               | 699, 1299-1300, 1425, 1449, 1619      | log10l.....                        | 710                   |
| LC_TIME.....        | 496, 834, 1299-1300                   | log1p().....                       | <b>712</b>            |
| ldexp().....        | <b>677</b>                            | log1pf.....                        | 712                   |
| ldexpf.....         | 677                                   | log1pl.....                        | 712                   |
| ldexpl.....         | 677                                   | log2().....                        | <b>714</b>            |
| ldiv().....         | <b>679</b>                            | log2f.....                         | 714                   |
| legacy.....         | 1                                     | log2l.....                         | 714                   |
| lfind().....        | <b>680</b> , 726                      | logb().....                        | <b>716</b>            |
| lgamma().....       | <b>681</b>                            | logbf.....                         | 716                   |
| lgammaf.....        | 681                                   | logbl.....                         | 716                   |
| lgammal.....        | 681                                   | logf().....                        | 708, <b>718</b>       |
| LIFO.....           | 57                                    | login shell.....                   | 302                   |
| LINE_MAX.....       | 1473                                  | LOGIN_NAME_MAX.....                | 521, 1473             |
| link to a file..... | 685                                   | LOGIN_PROCESS.....                 | 284-285               |

|                               |                         |                                 |                                |
|-------------------------------|-------------------------|---------------------------------|--------------------------------|
| logl.....                     | 708, 718                | mbsrtowcs() .....               | 743                            |
| LOG_.....                     | 18                      | mbstowcs() .....                | 745                            |
| LOG_constants in syslog.....  | 210                     | mbtowc().....                   | 747                            |
| LOG_ALERT .....               | 210                     | MB_CUR_MAX.....                 | 736, 738, 740, 747, 1598, 1640 |
| LOG_CONS.....                 | 211                     | MC1 .....                       | 5                              |
| LOG_CRIT .....                | 210                     | MC2 .....                       | 5                              |
| LOG_DEBUG .....               | 210                     | MC3 .....                       | 5                              |
| LOG_EMERG.....                | 210                     | MCL.....                        | 16                             |
| LOG_ERR.....                  | 210                     | MCL_CURRENT .....               | 772                            |
| LOG_INFO.....                 | 210                     | MCL_FUTURE.....                 | 772                            |
| LOG_LOCAL.....                | 210                     | memccpy().....                  | 749                            |
| LOG_NDELAY.....               | 211                     | memchr() .....                  | 750                            |
| LOG_NOTICE.....               | 210                     | memcmp() .....                  | 751                            |
| LOG_NOWAIT.....               | 211                     | memcpy().....                   | 752                            |
| LOG_ODELAY .....              | 211                     | MEMLOCK_FUTURE.....             | 779                            |
| LOG_PID.....                  | 211                     | memmove().....                  | 753                            |
| LOG_USER.....                 | 210-211                 | memory management.....          | 43                             |
| LOG_WARNING .....             | 210                     | memory protection option .....  | 778                            |
| longjmp() .....               | 719                     | memset().....                   | 754                            |
| LONG_MAX.....                 | 1457, 1625              | message catalog descriptor..... | 297, 302, 307                  |
| LONG_MIN .....                | 1457, 1625              | message parts .....             | 39                             |
| lrand48() .....               | 263, 721                | message priority .....          | 38                             |
| lrint().....                  | 722                     | high-priority .....             | 38                             |
| lrintf.....                   | 722                     | normal.....                     | 38                             |
| lrintl.....                   | 722                     | priority.....                   | 38                             |
| lround() .....                | 724                     | message-discard mode.....       | 1173                           |
| lroundf.....                  | 724                     | message-nondiscard mode.....    | 1173                           |
| lroundl .....                 | 724                     | MET-1MEST .....                 | 1549                           |
| lsearch().....                | 726                     | MF .....                        | 5                              |
| lseek() .....                 | 728                     | MINSIGSTKSZ.....                | 1351                           |
| lstat().....                  | 730                     | mkdir() .....                   | 755                            |
| l .....                       | 16                      | mkfifo() .....                  | 758                            |
| L_ctermid.....                | 241                     | mknod() .....                   | 761                            |
| l_sysid.....                  | 333                     | mkstemp() .....                 | 764                            |
| makecontext().....            | 732                     | mktemp() .....                  | 766                            |
| malloc() .....                | 734                     | mktime() .....                  | 768                            |
| manipulate signal sets.....   | 1354                    | ML.....                         | 6                              |
| mappings.....                 | 779                     | mlock() .....                   | 770                            |
| MAP_.....                     | 16, 18                  | mlockall().....                 | 772                            |
| MAP_FAILED .....              | 779                     | MLR.....                        | 6                              |
| MAP_FIXED.....                | 775, 778                | mmap().....                     | 774                            |
| MAP_PRIVATE.....              | 394, 775, 779, 783, 815 | MM_.....                        | 18                             |
| MAP_SHARED.....               | 397, 775-776            | MM_APPL.....                    | 385                            |
| max-data-size attribute ..... | 76, 919-920             | MM_CONSOLE.....                 | 385                            |
| MAX_CANON.....                | 399                     | MM_ERROR .....                  | 386-387                        |
| MAX_INPUT.....                | 399                     | mm_FIRM.....                    | 385                            |
| may .....                     | 2                       | MM_HALT.....                    | 386                            |
| mblen() .....                 | 736                     | MM_HARD.....                    | 385                            |
| mbrlen() .....                | 738                     | MM_INFO.....                    | 386                            |
| mbrtowc() .....               | 740                     | MM_NOCON.....                   | 386                            |
| mbsinit().....                | 742                     | MM_NOMSG .....                  | 386                            |

## Index

|                        |                 |                                      |                                    |
|------------------------|-----------------|--------------------------------------|------------------------------------|
| MM_NOSEV.....          | 386             | MST7MDT.....                         | 1549                               |
| MM_NOTOK.....          | 386             | msync().....                         | <b>815</b>                         |
| MM_NRECOV.....         | 385             | MS.....                              | 16, 18                             |
| MM_NULLMC.....         | 385             | MS_ASYNC.....                        | 776, 815                           |
| MM_OK.....             | 386             | MS_INVALIDATE.....                   | 815-816                            |
| MM_OPSYS.....          | 385             | MS_SYNC.....                         | 776, 815                           |
| MM_PRINT.....          | 385, 387        | multicast.....                       | 67                                 |
| MM_RECOVER.....        | 385             | munlock().....                       | 770, <b>818</b>                    |
| MM_SOFT.....           | 385             | munlockall().....                    | 772, <b>819</b>                    |
| MM_UTIL.....           | 385             | munmap().....                        | <b>820</b>                         |
| MM_WARNING.....        | 386             | mutex attributes.....                | 1092                               |
| modf().....            | <b>781</b>      | mutex initialization attributes..... | 1091                               |
| modff.....             | 781             | mutex performance.....               | 1092                               |
| modfl.....             | 781             | MUXID_ALL.....                       | 621-622                            |
| MON.....               | <b>6</b>        | MUXID_R.....                         | 18                                 |
| MORECTL.....           | 525             | MX.....                              | <b>6</b>                           |
| MOREDATA.....          | 525             | M.....                               | 18                                 |
| MPR.....               | <b>6</b>        | name information.....                | 528                                |
| mprotect().....        | <b>783</b>      | name space.....                      | <b>14</b>                          |
| MQ.....                | 16              | NAME_MAX.....                        | 25, 97, 171, 185, 187, 190         |
| mq.....                | 16              | .....                                | 300, 319, 338, 391, 399, 430, 449  |
| mq_close().....        | <b>785</b>      | .....                                | 458, 674, 683, 730, 755, 758, 762  |
| mq_getattr().....      | <b>786</b>      | .....                                | 791, 802, 830, 840, 845, 967, 1179 |
| mq_notify().....       | <b>788</b>      | .....                                | 1183, 1190, 1215, 1223, 1255, 1263 |
| mq_open().....         | <b>790</b>      | .....                                | 1330, 1333, 1404, 1470, 1542, 1562 |
| MQ_OPEN_MAX.....       | 1473            | .....                                | 1570, 1572                         |
| MQ_PRIO_MAX.....       | 796-797, 1473   | NaN.....                             | 123, 407, 471                      |
| mq_receive().....      | <b>793</b>      | nan().....                           | <b>822</b>                         |
| mq_send().....         | <b>796</b>      | nanf.....                            | 822                                |
| mq_setattr().....      | <b>798</b>      | nanl.....                            | 822                                |
| mq_timedreceive.....   | 793             | nanosleep().....                     | <b>823</b>                         |
| mq_timedreceive()..... | <b>800</b>      | NDEBUG.....                          | 21, 128                            |
| mq_timedsend.....      | 796             | nearbyint().....                     | <b>825</b>                         |
| mq_timedsend().....    | <b>801</b>      | nearbyintf.....                      | 825                                |
| mq_unlink().....       | <b>802</b>      | nearbyintl.....                      | 825                                |
| rand48().....          | 263, <b>804</b> | network interfaces.....              | 59                                 |
| MSG.....               | <b>6</b> , 18   | NEW_TIME.....                        | 284-285                            |
| msgctl().....          | <b>805</b>      | nextafter().....                     | <b>827</b>                         |
| msgget().....          | <b>807</b>      | nextafterf.....                      | 827                                |
| msgrcv().....          | <b>809</b>      | nextafterl.....                      | 827                                |
| msgsnd().....          | <b>812</b>      | nexttoward.....                      | 827                                |
| MSGVERB.....           | 386-387         | nexttowardf.....                     | 827                                |
| msg.....               | 16              | nexttowardl.....                     | 827                                |
| MSG.....               | 18              | nftw().....                          | <b>829</b>                         |
| MSG_ANY.....           | 524             | NGROUPS_MAX.....                     | 513, 1473                          |
| MSG_BAND.....          | 524, 1154       | nice().....                          | <b>832</b>                         |
| MSG_EOR.....           | 1393, 1395      | NLSPATH.....                         | 171                                |
| MSG_HIPRI.....         | 524, 1154       | NL.....                              | 18                                 |
| MSG_NOERROR.....       | 809-810         | NL_ARGMAX.....                       | 404, 435, 468, 477                 |
| msg_perm.....          | 40              | NL_CAT_LOCALE.....                   | 171                                |
| msqid.....             | 40              | nl_langinfo().....                   | <b>834</b>                         |

|                                  |                                    |                              |                                         |
|----------------------------------|------------------------------------|------------------------------|-----------------------------------------|
| nohup utility.....               | 303                                | PS .....                     | 7                                       |
| non-local jumps .....            | 1372                               | RS .....                     | 7                                       |
| non-volatile storage .....       | 452                                | RTS.....                     | 7                                       |
| rand48() .....                   | 263, <b>836</b>                    | SD.....                      | 7                                       |
| ntohl().....                     | 587, <b>837</b>                    | SEM.....                     | 8                                       |
| ntohs.....                       | 587, 837                           | SHM.....                     | 8                                       |
| NULL.....                        | 215, 241, 248, 256, 261, 779, 1181 | SIO .....                    | 8                                       |
| NUM_EMPL.....                    | 585                                | SPI.....                     | 8                                       |
| NZERO .....                      | 543, 832                           | SPN .....                    | 8                                       |
| n.....                           | 16                                 | SS.....                      | 8                                       |
| OB.....                          | <b>6</b>                           | TCT .....                    | 8                                       |
| obsolescent.....                 | 180                                | TEF.....                     | 8                                       |
| OF.....                          | 7                                  | THR .....                    | 9                                       |
| OH.....                          | 7                                  | TMO .....                    | 9                                       |
| OLD_TIME.....                    | 284-285                            | TMR.....                     | 9                                       |
| open a file .....                | 842                                | TPI.....                     | 9                                       |
| open a named semaphore.....      | 1255                               | TPP.....                     | 9                                       |
| open a shared memory object..... | 1330                               | TPS.....                     | 9                                       |
| open() .....                     | <b>838</b>                         | TRC.....                     | 9                                       |
| opendir() .....                  | <b>845</b>                         | TRI .....                    | 10                                      |
| openlog() .....                  | 210, <b>848</b>                    | TRL.....                     | 10                                      |
| OPEN_MAX .....                   | 171, 265, 280, 332, 391            | TSA .....                    | 10                                      |
| .....                            | 430, 463, 507, 510, 521, 551, 593  | TSF.....                     | 10                                      |
| .....                            | 790, 830, 840, 845, 856, 883, 886  | TSH.....                     | 10                                      |
| .....                            | 1473, 1531                         | TSP.....                     | 10                                      |
| optarg.....                      | 532, 849                           | TSS.....                     | 10                                      |
| opterr .....                     | 532, 849                           | TYM.....                     | 10                                      |
| optind.....                      | 532, 849                           | UP .....                     | 11                                      |
| option                           |                                    | XSR .....                    | 11                                      |
| ADV.....                         | 3                                  | optopt .....                 | 532, 535, 849                           |
| AIO .....                        | 3                                  | optstring .....              | 535                                     |
| BAR.....                         | 3                                  | orphaned process group ..... | 310                                     |
| BE.....                          | 4                                  | O .....                      | 18                                      |
| CD.....                          | 4                                  | O_ constants                 |                                         |
| CPT .....                        | 4                                  | used in open().....          | 838                                     |
| CS.....                          | 4                                  | used in posix_openpt() ..... | 873                                     |
| FD.....                          | 4                                  | O_ACCMODE.....               | 329                                     |
| FR.....                          | 4                                  | O_APPEND.....                | 41, 116, 246, 343, 838, 1656            |
| FSC.....                         | 5                                  | O_CREAT.....                 | 230, 790-791, 802, 838-840              |
| IP6 .....                        | 5                                  | .....                        | 1248, 1254, 1329-1331                   |
| MC1 .....                        | 5                                  | O_DSYNC .....                | 108, 838-839, 1173, 1657                |
| MC2 .....                        | 5                                  | O_EXCL .....                 | 791, 838-839, 1254, 1329-1330           |
| MC3 .....                        | 5                                  | O_NDELAY .....               | 1661                                    |
| MF.....                          | 5                                  | O_NOCTTY .....               | 839, 843, 873                           |
| ML.....                          | 6                                  | O_NONBLOCK.....              | 24, 205, 327, 360, 364                  |
| MLR.....                         | 6                                  | .....                        | 370, 416, 420, 443, 445, 525, 617       |
| MON .....                        | 6                                  | .....                        | 619, 791, 793, 796, 798, 839-841        |
| MPR.....                         | 6                                  | .....                        | 856, 859, 1155, 1172, 1657, 1660        |
| MSG.....                         | 6                                  | O_RDONLY.....                | 251, 790, 838-840, 843, 1329, 1331      |
| MX .....                         | 6                                  | O_RDWR .....                 | 322, 705, 790, 838-842, 873, 1329, 1331 |
| PIO.....                         | 7                                  | O_RSYNC .....                | 839, 1173                               |

## Index

- O\_SYNC .....108, 839, 1173, 1657
- O\_TRUNC.....230, 839, 841, 843, 1330-1331
- O\_WRONLY.....230, 322, 705, 790, 838-841, 843
- PAGESIZE .....43, 770, 816, 820, 986, 1475
- PAGE\_SIZE.....1475
- PATH.....215
- PATH environment variable .....305
- pathconf() .....399, **850**
- PATH\_MAX.....25, 97, 171, 185, 187, 190  
.....251, 300, 319, 338, 391, 399, 430  
.....449, 458, 494, 575, 674, 683, 730  
.....755, 758, 762, 791, 802, 830, 840  
.....845, 967, 1183, 1190, 1215, 1223  
.....1255, 1263, 1330, 1333, 1404, 1470  
.....1477, 1542, 1562, 1570, 1572
- pause().....**851**
- pclose() .....**852**
- pd .....16
- PENDIN.....19
- perror() .....**854**
- persistent connection (I\_PLINK).....**622**
- PF\_ .....18
- physical write .....452
- ph\_ .....16
- PIO .....7
- pipe .....396, 843, 1660
- pipe() .....**856**
- PIPE\_BUF .....399, 1657, 1660
- PIPE\_MAX .....1662
- plain characters.....1424
- POLL .....18
- poll() .....**858**
- POLLERR .....858
- POLLHUP .....858
- POLLIN .....858
- POLLNVAL .....859
- POLLOUT .....858
- POLLPRI.....858
- POLLRDBAND.....858
- POLLRDNORM.....858
- POLLWRBAND.....858
- POLLWRNORM.....858
- POLL\_ .....18
- popen() .....**862**
- portability.....**3**
- POSIX.....268
- POSIX.1 symbols.....13
- POSIX\_.....15, 17
- posix\_.....15, 17
- POSIX\_ALLOC\_SIZE\_MIN .....399
- posix\_fadvise() .....**864**
- POSIX\_FADV\_DONTNEED.....864
- POSIX\_FADV\_NOREUSE.....864
- POSIX\_FADV\_NORMAL.....864
- POSIX\_FADV\_RANDOM.....864
- POSIX\_FADV\_SEQUENTIAL.....864
- POSIX\_FADV\_WILLNEED.....864
- posix\_fallocate().....**866**
- posix\_madvise().....**868**
- POSIX\_MADV\_DONTNEED.....868
- POSIX\_MADV\_NORMAL.....868
- POSIX\_MADV\_RANDOM.....868
- POSIX\_MADV\_SEQUENTIAL.....868
- POSIX\_MADV\_WILLNEED.....868
- posix\_memalign().....**872**
- posix\_mem\_offset().....**870**
- posix\_openpt() .....**873**
- POSIX\_REC\_INCR\_XFER\_SIZE.....399
- POSIX\_REC\_MAX\_XFER\_SIZE.....399
- POSIX\_REC\_MIN\_XFER\_SIZE.....399
- POSIX\_REC\_XFER\_ALIGN.....399
- posix\_spawn() .....**875**
- posix\_spawnattr\_destroy().....**890**
- posix\_spawnattr\_getflags() .....**892**
- posix\_spawnattr\_getpgroup() .....**894**
- posix\_spawnattr\_getschedparam() .....**896**
- posix\_spawnattr\_getschedpolicy() .....**898**
- posix\_spawnattr\_getsigdefault().....**900**
- posix\_spawnattr\_getsigmask().....**902**
- posix\_spawnattr\_init() .....890, **904**
- posix\_spawnattr\_setflags() .....892, **905**
- posix\_spawnattr\_setpgroup() .....894, **906**
- posix\_spawnattr\_setschedparam().....896, **907**
- posix\_spawnattr\_setschedpolicy() .....898, **908**
- posix\_spawnattr\_setsigdefault().....900, **909**
- posix\_spawnattr\_setsigmask() .....902, **910**
- posix\_spawnp() .....875, **911**
- posix\_spawn\_file\_actions\_addclose().....**883**
- posix\_spawn\_file\_actions\_adddup2() .....**886**
- posix\_spawn\_file\_actions\_addopen().....883, **888**
- posix\_spawn\_file\_actions\_destroy() .....**889**
- posix\_spawn\_file\_actions\_init .....889
- POSIX\_SPAWN\_RESETEIDS.....876, 892
- POSIX\_SPAWN\_SETPGROUP.....876, 892, 894
- POSIX\_SPAWN\_SETSCHEDPARAM.....892, 896
- POSIX\_SPAWN\_SETSCHEDULER .....876, 892  
.....896, 898
- POSIX\_SPAWN\_SETSIGDEF.....877, 892, 900
- POSIX\_SPAWN\_SETSIGMASK.....892, 902
- POSIX\_TRACE\_ADD\_EVENTSET.....950
- POSIX\_TRACE\_ALL\_EVENTS.....944
- POSIX\_TRACE\_APPEND.....917, 937

- posix\_trace\_attr\_destroy() ..... **912**
- posix\_trace\_attr\_getclockres() ..... **914**
- posix\_trace\_attr\_getcreatetime ..... 914
- posix\_trace\_attr\_getgenversion ..... 914
- posix\_trace\_attr\_getinherited() ..... **916**
- posix\_trace\_attr\_getlogfullpolicy ..... 916
- posix\_trace\_attr\_getlogsize() ..... **919**
- posix\_trace\_attr\_getmaxdatasize ..... 919
- posix\_trace\_attr\_getmaxsystemevents ..... 919
- posix\_trace\_attr\_getmaxuserevents ..... 919
- posix\_trace\_attr\_getname() ..... 914, **922**
- posix\_trace\_attr\_getstreamfullpolicy() ..... 916, **923**
- posix\_trace\_attr\_getstreamsize() ..... 919, **924**
- posix\_trace\_attr\_init() ..... 912, **925**
- posix\_trace\_attr\_setinherited() ..... 916, **926**
- posix\_trace\_attr\_setlogfullpolicy ..... 916, 926
- posix\_trace\_attr\_setlogsize() ..... 919, **927**
- posix\_trace\_attr\_setmaxdatasize ..... 919, 927
- posix\_trace\_attr\_setname() ..... 914, **928**
- posix\_trace\_attr\_setstreamfullpolicy() ..... 916, **929**
- posix\_trace\_attr\_setstreamsize() ..... 919, **930**
- posix\_trace\_clear() ..... **931**
- posix\_trace\_close() ..... **933**
- POSIX\_TRACE\_CLOSE\_FOR\_CHILD ..... 916
- posix\_trace\_create() ..... **935**
- posix\_trace\_create\_withlog ..... 935
- POSIX\_TRACE\_ERROR trace event ..... 77
- posix\_trace\_event() ..... **939**
- posix\_trace\_eventid\_equal() ..... **941**
- posix\_trace\_eventid\_get\_name ..... 941
- posix\_trace\_eventid\_open() ..... 939, **943**
- posix\_trace\_eventset\_add() ..... **944**
- posix\_trace\_eventset\_del ..... 944
- posix\_trace\_eventset\_empty ..... 944
- posix\_trace\_eventset\_fill ..... 944
- posix\_trace\_eventset\_ismember ..... 944
- posix\_trace\_eventtypelist\_getnext\_id() ..... **946**
- posix\_trace\_eventtypelist\_rewind ..... 946
- posix\_trace\_event\_info structure ..... 74
- POSIX\_TRACE\_FILTER trace event ..... 77, 950
- POSIX\_TRACE\_FLUSH ..... 917
- posix\_trace\_flush() ..... 935, **947**
- POSIX\_TRACE\_FLUSHING ..... 73
- POSIX\_TRACE\_FULL ..... 72-74
- posix\_trace\_getnext\_event() ..... **953**
- posix\_trace\_get\_attr() ..... **948**
- posix\_trace\_get\_filter() ..... **950**
- posix\_trace\_get\_status() ..... 948, **952**
- POSIX\_TRACE\_INHERITED ..... 916
- POSIX\_TRACE\_LOOP ..... 73, 916-917, 937
- POSIX\_TRACE\_NOT\_FLUSHING ..... 73
- POSIX\_TRACE\_NOT\_FULL ..... 72-74
- POSIX\_TRACE\_NOT\_FULL ..... 931
- POSIX\_TRACE\_NOT\_TRUNCATED ..... 75, 954
- POSIX\_TRACE\_NO\_OVERRUN ..... 73-74, 948
- posix\_trace\_open() ..... 933, **956**
- POSIX\_TRACE\_OVERFLOW trace event ..... 77
- POSIX\_TRACE\_OVERRUN ..... 73-74
- POSIX\_TRACE\_RESUME trace event ..... 77
- posix\_trace\_rewind ..... 933, 956
- POSIX\_TRACE\_RUNNING ..... 72, 959
- POSIX\_TRACE\_SET\_EVENTSET ..... 950
- posix\_trace\_set\_filter() ..... 950, **957**
- posix\_trace\_shutdown() ..... 935, **958**
- POSIX\_TRACE\_START trace event ..... 77, 959
- posix\_trace\_start() ..... **959**
- posix\_trace\_status\_info structure ..... 72
- posix\_trace\_stop ..... 959
- POSIX\_TRACE\_STOP trace event ..... 77, 959
- POSIX\_TRACE\_SUB\_EVENTSET ..... 950
- POSIX\_TRACE\_SUSPENDED ..... 72-73, 959
- POSIX\_TRACE\_SYSTEM\_EVENTS ..... 944
- posix\_trace\_timedgetnext\_event() ..... 953, **961**
- posix\_trace\_trid\_eventid\_open() ..... 941, **962**
- POSIX\_TRACE\_TRUNCATED\_READ ..... 75, 954
- POSIX\_TRACE\_TRUNCATED\_RECORD ..... 75, 954
- posix\_trace\_trygetnext\_event() ..... 953, **963**
- POSIX\_TRACE\_UNTIL\_FULL ..... 73, 916-917, 937
- POSIX\_TRACE\_USER\_EVENT\_MAX ..... 939
- POSIX\_TRACE\_WOPID\_EVENTS ..... 944
- POSIX\_TYPED\_MEM\_ALLOCATE ..... 774-775
- ..... 870, 964, 966
- POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG ..... 774-775, 870, 964, 966
- posix\_typed\_mem\_get\_info() ..... **964**
- POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE ..... 820, 966
- posix\_typed\_mem\_open() ..... **966**
- pow() ..... **969**
- powf ..... 969
- powl ..... 969
- pread() ..... **972**, 1172
- predefined stream
  - standard error ..... 37
  - standard input ..... 37
  - standard output ..... 37
- preempted thread ..... 1038
- PRI ..... 18
- printf() ..... 404, **973**
- priority ..... 38
- PRIOR ..... 18
- PRIO\_INHERIT ..... 1088

## Index

- PRIO\_PGRP .....543
- PRIO\_PROCESS.....543
- PRIO\_USER .....543
- process
  - concurrent execution .....396
  - setting real and effective user IDs.....1313
  - single-threaded .....396
- process creation .....396
- process group
  - orphaned .....310
- process group ID.....539, 1305, 1317
- process ID.....310
- process lifetime .....670
- process scheduling .....44
- process shared memory .....1092
- process synchronization .....1092
- process termination.....309
- PROT .....16, 18
- PROT\_EXEC.....774, 783
- PROT\_NONE.....44, 774, 783
- PROT\_READ.....774, 783
- PROT\_WRITE .....774, 776, 778, 783
- PS.....7
- pselect().....974
- pseudo-random sequence generation functions....  
.....1169
- PST8PDT .....1549
- ps .....16
- PTHREAD .....16
- pthread .....16
- pthread\_atfork().....979
- pthread\_attr\_destroy().....981
- pthread\_attr\_getdetachstate().....984
- pthread\_attr\_getguardsize().....986
- pthread\_attr\_getinheritsched().....988
- pthread\_attr\_getschedparam() .....990
- pthread\_attr\_getschedpolicy().....992
- pthread\_attr\_getscope() .....994
- pthread\_attr\_getstack() .....996
- pthread\_attr\_getstackaddr().....998
- pthread\_attr\_getstacksize().....1000
- pthread\_attr\_init().....981, 1002
- pthread\_attr\_setdetachstate() .....984, 1003
- pthread\_attr\_setguardsize().....986, 1004
- pthread\_attr\_setinheritsched() .....988, 1005
- pthread\_attr\_setschedparam() .....990, 1006
- pthread\_attr\_setschedpolicy().....992, 1007
- pthread\_attr\_setscope() .....994, 1008
- pthread\_attr\_setstack() .....996, 1009
- pthread\_attr\_setstackaddr().....998, 1010
- pthread\_attr\_setstacksize() .....1000, 1011
- pthread\_barrierattr\_destroy().....1016
- pthread\_barrierattr\_getpshared() .....1018
- pthread\_barrierattr\_init() .....1016, 1020
- pthread\_barrierattr\_setpshared() .....1018, 1021
- pthread\_barrier\_destroy().....1012
- pthread\_barrier\_init.....1012
- PTHREAD\_BARRIER\_SERIAL\_THREAD .....1014
- pthread\_barrier\_wait() .....1014
- pthread\_cancel() .....1022
- PTHREAD\_CANCELED .....57, 1057
- PTHREAD\_CANCEL\_ASYNCHRONOUS.....54  
.....1132
- PTHREAD\_CANCEL\_DEFERRED 54, 1036, 1132
- PTHREAD\_CANCEL\_DISABLE .....54, 1132
- PTHREAD\_CANCEL\_ENABLE.....54, 1132
- pthread\_cleanup\_pop() .....1024
- pthread\_cleanup\_push.....1024
- pthread\_condattr\_destroy() .....1041
- pthread\_condattr\_getclock().....1043
- pthread\_condattr\_getpshared().....1045
- pthread\_condattr\_init().....1041, 1047
- pthread\_condattr\_setclock() .....1043, 1048
- pthread\_condattr\_setpshared().....1045, 1049
- pthread\_cond\_broadcast().....1029
- pthread\_cond\_destroy().....1032
- pthread\_cond\_init.....1032
- PTHREAD\_COND\_INITIALIZER.....1032
- pthread\_cond\_signal() .....1029, 1035
- pthread\_cond\_timedwait().....1036
- pthread\_cond\_wait.....1036
- pthread\_create().....1050
- PTHREAD\_CREATE\_DETACHED.....30, 984
- PTHREAD\_CREATE\_JOINABLE.....30, 984, 1067
- PTHREAD\_DESTRUCTOR\_ITERATIONS .....1064  
.....1069, 1475
- pthread\_detach().....1053
- pthread\_equal().....1055
- pthread\_exit() .....1056
- PTHREAD\_EXPLICIT\_SCHED.....988
- pthread\_getconcurrency() .....1058
- pthread\_getcpuclockid() .....1060
- pthread\_getschedparam().....1061
- pthread\_getspecific() .....1064
- PTHREAD\_INHERIT\_SCHED.....988
- pthread\_join() .....1066
- PTHREAD\_KEYS\_MAX.....1069, 1475
- pthread\_key\_create() .....1069
- pthread\_key\_delete() .....1073
- pthread\_kill() .....1075
- pthread\_mutexattr\_destroy().....1091
- pthread\_mutexattr\_getprioceiling() .....1096

- pthread\_mutexattr\_getprotocol() .....**1098**  
 pthread\_mutexattr\_getpshared() .....**1100**  
 pthread\_mutexattr\_gettype() .....**1102**  
 pthread\_mutexattr\_init() .....1091, **1104**  
 pthread\_mutexattr\_setprioceiling() .....1096, **1105**  
 pthread\_mutexattr\_setprotocol() .....1098, **1106**  
 pthread\_mutexattr\_setpshared() .....1100, **1107**  
 pthread\_mutexattr\_settype() .....1102, **1108**  
 PTHREAD\_MUTEX\_DEFAULT .....1084, 1102  
 pthread\_mutex\_destroy() .....**1076**  
 PTHREAD\_MUTEX\_ERRORCHECK...1084, 1102  
 pthread\_mutex\_getprioceiling() .....**1081**  
 pthread\_mutex\_init() .....1076, **1083**  
 PTHREAD\_MUTEX\_INITIALIZER .....1076, 1083  
 pthread\_mutex\_lock() .....**1084**  
 PTHREAD\_MUTEX\_NORMAL .....1084, 1102  
 PTHREAD\_MUTEX\_RECURSIVE .....1084  
 .....1102-1103  
 pthread\_mutex\_setprioceiling() .....1081, **1087**  
 pthread\_mutex\_timedlock() .....**1088**  
 pthread\_mutex\_trylock() .....1084, **1090**  
 pthread\_mutex\_unlock .....1084, 1090  
 pthread\_once() .....**1109**  
 PTHREAD\_ONCE\_INIT .....1109  
 PTHREAD\_PRIO\_INHERIT .....1098  
 PTHREAD\_PRIO\_NONE .....1098  
 PTHREAD\_PRIO\_PROTECT .....1085, 1098  
 PTHREAD\_PROCESS\_PRIVATE .....1018, 1045  
 .....1092, 1100, 1127, 1141  
 PTHREAD\_PROCESS\_SHARED .....1018, 1045  
 .....1092, 1100, 1127, 1141  
 pthread\_rwlockattr\_destroy() .....**1125**  
 pthread\_rwlockattr\_getpshared() .....**1127**  
 pthread\_rwlockattr\_init() .....1125, **1129**  
 pthread\_rwlockattr\_setpshared() .....1127, **1130**  
 pthread\_rwlock\_destroy() .....**1111**  
 pthread\_rwlock\_rdlock() .....1111, **1113**  
 pthread\_rwlock\_timedrdlock() .....**1115**  
 pthread\_rwlock\_timedwrlock() .....**1117**  
 pthread\_rwlock\_tryrdlock() .....1113, **1119**  
 pthread\_rwlock\_trywrlock() .....**1120**  
 pthread\_rwlock\_unlock() .....**1122**  
 pthread\_rwlock\_wrlock() .....1120, **1124**  
 PTHREAD\_SCOPE\_PROCESS .....52, 994  
 PTHREAD\_SCOPE\_SYSTEM .....52, 994  
 pthread\_self() .....**1131**  
 pthread\_setcancelstate() .....**1132**  
 pthread\_setcanceltype .....1132  
 pthread\_setconcurrency() .....1058, **1134**  
 pthread\_setschedparam() .....1061, **1135**  
 pthread\_setschedprio() .....**1136**  
 pthread\_setspecific() .....1064, **1138**  
 pthread\_sigmask() .....**1139**  
 pthread\_spin\_destroy() .....**1141**  
 pthread\_spin\_init .....1141  
 pthread\_spin\_lock() .....**1143**  
 pthread\_spin\_trylock .....1143  
 pthread\_spin\_unlock() .....**1145**  
 PTHREAD\_STACK\_MIN .....996, 998, 1000, 1475  
 pthread\_testcancel() .....1132, **1146**  
 PTHREAD\_THREADS\_MAX .....1050, 1475  
 ptsname() .....**1147**  
 putc() .....**1148**  
 putchar() .....**1150**  
 putchar\_unlocked() .....487, **1151**  
 putc\_unlocked() .....487, **1149**  
 putenv() .....**1152**  
 putmsg() .....**1154**  
 putpmsg .....1154  
 puts() .....**1158**  
 pututxline() .....284, **1160**  
 putwc() .....**1161**  
 putwchar() .....**1162**  
 pwrite() .....**1163**, 1656  
 pw\_ .....16  
 p\_ .....16  
 P\_ .....17  
 P\_ALL .....1595  
 P\_PGID .....1595  
 P\_PID .....1595  
 qsort() .....**1164**  
 queue a signal to a process .....1370  
 raise() .....**1166**  
 rand() .....**1168**  
 random() .....608, **1171**  
 RAND\_MAX .....1168  
 rand\_r .....1168  
 read from a file .....1175  
 read() .....**1172**  
 readdir() .....**1179**  
 readdir\_r .....1179  
 readlink() .....**1183**  
 readv() .....**1186**  
 real user ID .....98, 669  
 realloc() .....**1188**  
 realpath() .....**1190**  
 REALTIME .....105, 107-108, 110, 113-114  
 .....116, 197, 203, 337, 686, 770, 772  
 .....785-786, 788, 790, 793, 796, 798  
 .....802, 823, 1233-1237, 1240, 1248-1250  
 .....1252, 1254, 1257, 1261, 1263, 1265  
 .....1329, 1333, 1369, 1376, 1382, 1521

## Index

|                                     |                                   |                      |                          |
|-------------------------------------|-----------------------------------|----------------------|--------------------------|
| .....                               | 1524-1525                         | rindex()             | 1220                     |
| REALTIME THREADS .....              | 988, 992, 994                     | rint()               | 1221                     |
| .....                               | 1005, 1007-1008, 1061, 1081, 1087 | RLIMIT_.....         | 18                       |
| .....                               | 1096, 1098, 1105-1106, 1135-1136  | RLIMIT_AS.....       | 555                      |
| recv().....                         | 1192                              | RLIMIT_CORE.....     | 554                      |
| recvfrom().....                     | 1194                              | RLIMIT_CPU.....      | 554                      |
| recvmsg().....                      | 1197                              | RLIMIT_DATA.....     | 554                      |
| regcomp().....                      | 1200                              | RLIMIT_FSIZE.....    | 554                      |
| regerror.....                       | 1200                              | RLIMIT_NOFILE.....   | 554, 556                 |
| regexec.....                        | 1200                              | RLIMIT_STACK.....    | 554                      |
| regfree.....                        | 1200                              | rlim_.....           | 16                       |
| register fork handlers.....         | 979                               | RLIM_.....           | 18                       |
| REG.....                            | 18                                | RLIM_INFINITY.....   | 554-555                  |
| REG_constants                       |                                   | RLIM_SAVED_CUR.....  | 555                      |
| error return values of regcomp..... | 1202                              | RLIM_SAVED_MAX.....  | 555                      |
| used in regcomp.....                | 1200                              | rmdir().....         | 1223                     |
| REG_BADBR.....                      | 1202                              | RMSGD.....           | 616                      |
| REG_BADPAT.....                     | 1202                              | RMSGN.....           | 616                      |
| REG_BADRPT.....                     | 1202                              | RNORM.....           | 616                      |
| REG_EBRACE.....                     | 1202                              | round robin.....     | 46                       |
| REG_EBRACK.....                     | 1202                              | round().....         | 1226                     |
| REG_ECOLLATE.....                   | 1202                              | roundf.....          | 1226                     |
| REG_ECTYPE.....                     | 1202                              | roundl.....          | 1226                     |
| REG_EESCAPE.....                    | 1202                              | routing.....         | 59                       |
| REG_EPAREN.....                     | 1202                              | RPROTDAT.....        | 616                      |
| REG_ERANGE.....                     | 1202                              | RPROTDIS.....        | 616                      |
| REG_ESPACE.....                     | 1202                              | RPROTNORM.....       | 616                      |
| REG_ESUBREG.....                    | 1202                              | RS.....              | 7                        |
| REG_EXTENDED.....                   | 1200                              | RS_HIPRI.....        | 524, 615, 1154           |
| REG_ICASE.....                      | 1200                              | RTLD_.....           | 18                       |
| REG_NEWLINE.....                    | 1200                              | RTLD_DEFAULT.....    | 261                      |
| REG_NOMATCH.....                    | 1202                              | RTLD_GLOBAL.....     | 254, 258-259, 262        |
| REG_NOSUB.....                      | 1200                              | RTLD_LAZY.....       | 258, 261                 |
| REG_NOTBOL.....                     | 1201                              | RTLD_LOCAL.....      | 259                      |
| REG_NOTEOL.....                     | 1201                              | RTLD_NEXT.....       | 261-262                  |
| remainder().....                    | 1207                              | RTLD_NOW.....        | 258-259                  |
| remainderf.....                     | 1207                              | RTS.....             | 7                        |
| remainderl.....                     | 1207                              | RTSIG_MAX.....       | 1475                     |
| remove a directory.....             | 1224                              | RUSAGE_.....         | 18                       |
| remove directory entries.....       | 1564                              | RUSAGE_CHILDREN..... | 557                      |
| remove().....                       | 1209                              | RUSAGE_SELF.....     | 557                      |
| remque().....                       | 610, 1211                         | ru_.....             | 16                       |
| remquo().....                       | 1212                              | s6_.....             | 16                       |
| remquof.....                        | 1212                              | sa_.....             | 16                       |
| remquol.....                        | 1212                              | SA_.....             | 18                       |
| rename a file.....                  | 1216                              | SA_NOCLDSTOP.....    | 31, 1345, 1348           |
| rename().....                       | 1214                              | SA_NOCLDWAIT.....    | 307-308, 557, 1346, 1589 |
| rewind().....                       | 1218                              | SA_NODEFER.....      | 1346                     |
| rewinddir().....                    | 1219                              | SA_ONSTACK.....      | 297, 1345                |
| re_.....                            | 16                                | SA_RESETHAND.....    | 148, 1345-1346           |
| RE_DUP_MAX.....                     | 1475                              | SA_RESTART.....      | 148, 977, 1345, 1359     |

|                               |                                            |                                            |                      |
|-------------------------------|--------------------------------------------|--------------------------------------------|----------------------|
| SA_SIGINFO.....               | 1344-1345, 1348, 1369                      | SEM_FAILED .....                           | 1255                 |
| scalb() .....                 | <b>1228</b>                                | sem_getvalue() .....                       | <b>1250</b>          |
| scalbln().....                | <b>1230</b>                                | sem_init().....                            | <b>1252</b>          |
| scalblnf.....                 | 1230                                       | SEM_NSEMS_MAX.....                         | 1252, 1475           |
| scalblnl.....                 | 1230                                       | sem_open().....                            | <b>1254</b>          |
| scalbn .....                  | 1230                                       | sem_perm.....                              | 40                   |
| scalbnf.....                  | 1230                                       | sem_post().....                            | <b>1257</b>          |
| scalbnl .....                 | 1230                                       | sem_timedwait() .....                      | <b>1259</b>          |
| scanf().....                  | 435, <b>1232</b>                           | sem_trywait() .....                        | <b>1261</b>          |
| schedule alarm .....          | 119                                        | SEM_UNDO .....                             | 1272                 |
| scheduling documentation..... | 54                                         | sem_unlink().....                          | <b>1263</b>          |
| scheduling policy             |                                            | SEM_VALUE_MAX.....                         | 1254, 1475           |
| round robin.....              | 46                                         | sem_wait() .....                           | 1261, <b>1265</b>    |
| SCHED_.....                   | 16                                         | send().....                                | <b>1277</b>          |
| sched_.....                   | 16                                         | sendmsg() .....                            | <b>1279</b>          |
| SCHED_FIFO .....              | 42, 45, 53, 298, 394                       | sendto() .....                             | <b>1282</b>          |
| .....                         | 543, 832, 990, 992, 1061, 1096             | service name .....                         | 426                  |
| .....                         | 1113, 1238, 1257                           | session.....                               | 310, 669, 1305, 1317 |
| sched_getparam() .....        | <b>1234</b>                                | set cancelability state .....              | 1132                 |
| sched_getscheduler() .....    | <b>1235</b>                                | set file creation mask .....               | 1555                 |
| sched_get_priority_max()..... | <b>1233</b>                                | set process group ID for job control ..... | 1305                 |
| sched_get_priority_min.....   | 1233                                       | set-group-ID .....                         | 189, 304, 309, 334   |
| SCHED_OTHER .....             | 45, 48, 543, 992, 1061, 1238               | set-user-ID.....                           | 304, 309, 494, 669   |
| SCHED_RR.....                 | 42, 45-46, 53, 298, 394                    | SETALL .....                               | 1266, 1269           |
| .....                         | 543, 832, 990, 992, 1061, 1113, 1238, 1257 | setbuf() .....                             | <b>1285</b>          |
| sched_rr_get_interval() ..... | <b>1236</b>                                | setcontext() .....                         | 491, <b>1286</b>     |
| sched_setparam().....         | <b>1237</b>                                | setgid() .....                             | <b>1287</b>          |
| sched_setscheduler().....     | <b>1240</b>                                | setenv().....                              | <b>1288</b>          |
| SCHED_SPORADIC ....           | 42, 45-46, 1113, 1238, 1257                | seteuid() .....                            | <b>1290</b>          |
| sched_yield().....            | <b>1243</b>                                | setgid() .....                             | <b>1291</b>          |
| SCM_.....                     | 18                                         | setgrent() .....                           | 272, <b>1293</b>     |
| SCN.....                      | 18                                         | sethostent() .....                         | 274, <b>1294</b>     |
| SD.....                       | 7                                          | setitimer() .....                          | 519, <b>1295</b>     |
| security considerations ...   | 191, 309, 669, 1305, 1406                  | setjmp() .....                             | <b>1296</b>          |
| seed48().....                 | 263, <b>1244</b>                           | setkey().....                              | <b>1298</b>          |
| seekdir() .....               | <b>1245</b>                                | setlocale().....                           | <b>1299</b>          |
| SEEK_CUR .....                | 330, 442, 728                              | setlogmask() .....                         | 210, <b>1303</b>     |
| SEEK_END .....                | 330, 442, 728                              | setnetent() .....                          | 276, <b>1304</b>     |
| SEEK_GET .....                | 1218                                       | setpgid() .....                            | <b>1305</b>          |
| SEEK_SET .....                | 42, 110, 116, 330, 442, 728                | setpgrp() .....                            | <b>1307</b>          |
| SEGV_.....                    | 18                                         | setpriority() .....                        | 543, <b>1308</b>     |
| select().....                 | 974, <b>1247</b>                           | setprotoent() .....                        | 278, <b>1309</b>     |
| SEM.....                      | <b>8</b>                                   | setpwent() .....                           | 280, <b>1310</b>     |
| semctl().....                 | <b>1266</b>                                | setregid().....                            | <b>1311</b>          |
| semget() .....                | <b>1269</b>                                | setreuid().....                            | <b>1313</b>          |
| semid.....                    | 40                                         | setrlimit().....                           | 554, <b>1315</b>     |
| semop() .....                 | <b>1272</b>                                | setservent() .....                         | 282, <b>1316</b>     |
| SEM_.....                     | 16, 18                                     | setsid().....                              | <b>1317</b>          |
| sem_.....                     | 16                                         | setsockopt().....                          | <b>1319</b>          |
| sem_close().....              | <b>1248</b>                                | setstate() .....                           | 608, <b>1322</b>     |
| sem_destroy().....            | <b>1249</b>                                | setuid() .....                             | <b>1323</b>          |

## Index

- setutxent() .....284, **1326**
- SETVAL.....1266, 1269
- setvbuf().....**1327**
- shall.....2
- shell.....302, 310, 522, 539, 669, 1306, 1592
  - job.....**669**
  - login.....522
- shell scripts
  - exec.....303
- shell, login.....302
- SHM.....**8**, 18
- shmat().....**1335**
- shmctl().....**1337**
- shmdt().....**1339**
- shmget().....**1340**
- shmids.....40
- SHMLBA.....1335
- shm\_.....16
- SHM\_.....18
- shm\_open().....**1329**
- shm\_perm.....40
- SHM\_RDONLY.....1335
- SHM\_RND.....1335
- shm\_unlink().....**1333**
- should.....2
- shutdown().....**1342**
- SHUT\_.....18
- SIGABRT.....92
- sigaction().....**1344**
- sigaddset().....**1350**
- SIGALRM.....119, 519, 1388, 1551, 1568
- sigaltstack().....**1351**
- SIGBUS.....44, 776, 779, 1139
- SIGCANCEL.....1022
- SIGCHLD.....211, 307-308, 557, 582
  - .....1345, 1348, 1357, 1481, 1589, 1595
- SIGCLD.....1348
- SIGCONT.....34, 308, 310, 668-669
- sigdelset().....**1353**
- sigemptyset().....**1354**
- SIGEV\_.....16
- sigev\_.....16
- SIGEV\_NONE.....29, 42
- SIGEV\_SIGNAL.....29, 1521
- SIGEV\_THREAD.....29-30
- sigfillset().....**1356**
- SIGFPE.....1139, 1363
- sighold().....**1357**
- SIGHUP.....205, 308, 310
- sigignore.....1357
- SIGILL.....1139, 1363
- SIGINT.....396, 1481
- siginterrupt().....**1359**
- sigismember().....**1361**
- SIGKILL.....669, 1344, 1348, 1357
- siglongjmp().....**1362**
- signal generation and delivery.....28
  - realtime.....29
- signal handler.....1363
- signal().....**1363**
- signaling a condition.....1030
- signals.....28
- signbit().....**1365**
- sigpause().....1357, **1366**
- sigpending().....**1367**
- SIGPIPE.....327, 360, 416, 420, 443, 446, 1155, 1659
- SIGPOLL.....205, 614-615
- sigprocmask().....1139, **1368**
- SIGPROF.....519
- sigqueue().....**1369**
- SIGQUEUE\_MAX.....1369, 1475
- SIGQUIT.....1481
- sigrelse().....1357, **1371**
- SIGRTMAX.....29-30, 1369, 1376, 1380
- SIGRTMIN.....29-30, 1369, 1376, 1380
- SIGSEGV.....44, 555, 820, 986, 1139, 1363
- sigset.....1357, 1371
- sigsetjmp().....**1372**
- SIGSTKSZ.....1351
- SIGSTOP.....29, 1344, 1348, 1357
- sigsuspend().....**1374**
- sigtimedwait().....**1376**
- SIGTSTP.....29
- SIGTTIN.....29, 364, 370, 1174
- SIGTTOU.....29, 327, 360, 416, 420
  - .....443, 445, 1490, 1492, 1494, 1501
  - .....1504, 1659
- SIGURG.....615
- SIGVTALRM.....519
- sigwait().....**1380**
- sigwaitinfo().....1376, **1382**
- SIGXCPU.....554
- SIGXFSZ.....554, 1542
- SIG\_.....16, 18
- SIG\_BLOCK.....1139
- SIG\_DFL.....30, 297, 555, 1344, 1346, 1363
- SIG\_ERR.....148, 1363
- SIG\_HOLD.....1357
- SIG\_IGN.....30-31, 297, 303, 307-308, 557
  - .....1344, 1363, 1589
- SIG\_SETMASK.....1139
- SIG\_UNBLOCK.....1139

|                                    |                                  |
|------------------------------------|----------------------------------|
| sin()                              | 1383                             |
| sin6_                              | 16                               |
| sinf                               | 1383                             |
| sinh()                             | 1385                             |
| sinhf                              | 1385                             |
| sinhl                              | 1385                             |
| sinl()                             | 1383, 1387                       |
| sin_                               | 16                               |
| SIO                                | 8                                |
| SIOCATMARK                         | 1391                             |
| sival_                             | 16                               |
| SI_                                | 16, 18                           |
| si_                                | 16-17                            |
| SI_ASYNCIO                         | 32                               |
| SI_MESGQ                           | 32                               |
| SI_QUEUE                           | 32                               |
| SI_TIMER                           | 32                               |
| SI_USER                            | 32                               |
| sleep()                            | 1388                             |
| sl_                                | 16                               |
| SND                                | 18                               |
| SNDZERO                            | 618                              |
| snprintf()                         | 404, 1390                        |
| SO                                 | 18                               |
| socketmark()                       | 1391                             |
| socket I/O mode                    | 60                               |
| socket out-of-band data            | 61                               |
| socket owner                       | 60                               |
| socket queue limits                | 60                               |
| socket receive queue               | 61                               |
| socket types                       | 59                               |
| socket()                           | 1393                             |
| socketpair()                       | 1395                             |
| sockets                            | 58                               |
| address families                   | 58                               |
| addressing                         | 59                               |
| asynchronous errors                | 62                               |
| connection indication queue        | 62                               |
| Internet Protocols                 | 66                               |
| IPv4                               | 67                               |
| IPv6                               | 67                               |
| local UNIX connections             | 66                               |
| options                            | 63                               |
| pending error                      | 60                               |
| protocols                          | 59                               |
| signals                            | 62                               |
| SOCK_                              | 18                               |
| SOCK_DGRAM                         | 66, 1393, 1395                   |
| SOCK_RAW                           | 66                               |
| SOCK_SEQPACKET                     | 66, 1393, 1395                   |
| SOCK_STREAM                        | 66, 1393, 1395                   |
| SPI                                | 8                                |
| SPN                                | 8                                |
| sporadic server policy             |                                  |
| execution capacity                 | 46                               |
| replenishment period               | 46                               |
| sprintf()                          | 404, 1397                        |
| spurious wakeup                    | 1030                             |
| sqrt()                             | 1398                             |
| sqrtf                              | 1398                             |
| sqrtd                              | 1398                             |
| srand()                            | 1168, 1400                       |
| srand48()                          | 263, 1401                        |
| srandom()                          | 608, 1402                        |
| SS                                 | 8                                |
| sscanf()                           | 435, 1403                        |
| SSIZE_MAX                          | 793, 809, 1172, 1183, 1425, 1656 |
| ss_                                | 16-17                            |
| SS_                                | 18                               |
| SS_DISABLE                         | 1351-1352                        |
| SS_ONSTACK                         | 1351                             |
| stack size                         | 981                              |
| stat()                             | 1404                             |
| statvfs()                          | 449, 1408                        |
| stderr                             | 1409                             |
| STDERR_FILENO                      | 1409                             |
| stdin                              | 1409                             |
| STDIN_FILENO                       | 862, 1409                        |
| stdio locking functions            | 375                              |
| stdio with explicit client locking | 487                              |
| stdout                             | 1409                             |
| STDOUT_FILENO                      | 862, 1409                        |
| STR                                | 18                               |
| strcasecmp()                       | 1411                             |
| strcat()                           | 1412                             |
| strchr()                           | 1413                             |
| strcmp()                           | 1414                             |
| strcoll()                          | 1416                             |
| strcpy()                           | 1418                             |
| strcspn()                          | 1420                             |
| strdup()                           | 1421                             |
| STREAM                             | 617, 619, 1154, 1173, 1658       |
| stream                             |                                  |
| byte-oriented                      | 36                               |
| wide-oriented                      | 36                               |
| STREAM head/tail                   | 38                               |
| stream-full-policy attribute       | 72-73, 76, 917                   |
| stream-min-size attribute          | 76, 920                          |
| STREAMS                            | 22, 205, 319, 338, 524, 613      |
|                                    | 628, 840-841, 858, 974           |
| access                             | 39                               |
| multiplexed                        | 621                              |

## Index

|                                         |                     |                                          |                                    |
|-----------------------------------------|---------------------|------------------------------------------|------------------------------------|
| overview.....                           | 38                  | SYMLINK_MAX .....                        | 399, 1470                          |
| streams.....                            | 34                  | SYMLOOP_MAX .....                        | 147, 220, 320, 338, 391            |
| interaction with file descriptors ..... | 35                  | .....                                    | 431, 450, 458, 464, 674, 762, 830  |
| stream orientation .....                | 36                  | .....                                    | 1190, 1281, 1284, 1475, 1543, 1572 |
| STREAM_MAX.....                         | 342, 391, 862, 1475 | sync() .....                             | <b>1472</b>                        |
| strerror().....                         | <b>1422</b>         | synchronously accept a signal.....       | 1377                               |
| strerror_r .....                        | 1422                | sysconf().....                           | <b>1473</b>                        |
| strfmon().....                          | <b>1424</b>         | syslog().....                            | 210, <b>1480</b>                   |
| strftime().....                         | <b>1428</b>         | system crash .....                       | 452                                |
| strlen().....                           | <b>1434</b>         | System III.....                          | 191, 1557                          |
| strncasecmp().....                      | 1411, <b>1436</b>   | system interfaces .....                  | 83                                 |
| strncat().....                          | <b>1437</b>         | system name.....                         | 1557                               |
| strncmp().....                          | <b>1438</b>         | system trace event type definitions..... | 76                                 |
| strncpy().....                          | <b>1439</b>         | System V.....                            | 119, 191, 305, 310, 332-333        |
| strpbrk().....                          | <b>1440</b>         | .....                                    | 401, 539, 669, 756, 1224, 1317     |
| strptime().....                         | <b>1441</b>         | .....                                    | 1348, 1372, 1496, 1557             |
| strrchr().....                          | <b>1445</b>         | system().....                            | <b>1481</b>                        |
| strspn().....                           | <b>1446</b>         | s_.....                                  | 16                                 |
| strstr().....                           | <b>1447</b>         | S.....                                   | 18                                 |
| strtod().....                           | <b>1448</b>         | S_BANDURG .....                          | 615                                |
| strtof .....                            | 1448                | S_ERROR.....                             | 614                                |
| strtoimax().....                        | <b>1452</b>         | S_HANGUP.....                            | 615                                |
| strtok().....                           | <b>1453</b>         | S_HIPRI .....                            | 614                                |
| strtok_r.....                           | 1453                | S_IFBLK .....                            | 761                                |
| strtol().....                           | <b>1456</b>         | S_IFCHR.....                             | 761                                |
| strtold().....                          | 1448, <b>1458</b>   | S_IFDIR.....                             | 761                                |
| strtoll().....                          | 1456, <b>1459</b>   | S_IFIFO .....                            | 761                                |
| strtoul().....                          | <b>1460</b>         | S_IFREG.....                             | 761                                |
| strtoull.....                           | 1460                | S_INPUT.....                             | 614                                |
| strtoumax().....                        | 1452, <b>1463</b>   | S_IRGRP.....                             | 323, 447, 761                      |
| strxfrm().....                          | <b>1464</b>         | S_IROTH.....                             | 323, 447, 761                      |
| str_.....                               | 16                  | S_IRUSR.....                             | 323, 447, 761                      |
| st_.....                                | 16                  | S_IRWXG .....                            | 761                                |
| ST_.....                                | 18                  | S_IRWXO .....                            | 761                                |
| ST_NOSUID .....                         | 298, 449            | S_IRWXU .....                            | 761                                |
| ST_RDONLY .....                         | 449                 | S_ISGID.....                             | 187-189, 761, 1542, 1657           |
| sun_.....                               | 17                  | S_ISUID.....                             | 187-188, 761, 1542, 1657           |
| superuser.....                          | 98, 191, 685, 1564  | S_ISVTX .....                            | 187, 761, 1215, 1224, 1562         |
| supplementary groups.....               | 191, 512            | S_IWGRP .....                            | 323, 447, 761                      |
| SVID .....                              | 1372                | S_IWOTH .....                            | 323, 447, 761                      |
| SVR4.....                               | 778, 823            | S_IWUSR .....                            | 323, 447, 761                      |
| sv_.....                                | 16                  | S_IXGRP .....                            | 761                                |
| SV_.....                                | 18                  | S_IXOTH .....                            | 761                                |
| swab().....                             | <b>1466</b>         | S_IXUSR .....                            | 761                                |
| swapcontext().....                      | 732, <b>1467</b>    | S_MSG.....                               | 614                                |
| swprintf().....                         | 468, <b>1468</b>    | S_OUTPUT .....                           | 614                                |
| swscanf().....                          | 477, <b>1469</b>    | S_RDBAND.....                            | 614-615                            |
| SWTCH.....                              | 19                  | S_RDNORM .....                           | 614                                |
| symbols                                 |                     | S_WRBAND .....                           | 614                                |
| POSIX.1.....                            | 13                  | S_WRNORM .....                           | 614                                |
| symlink().....                          | <b>1470</b>         | TABSIZE.....                             | 150, 726                           |

|                                   |                                                                                                       |
|-----------------------------------|-------------------------------------------------------------------------------------------------------|
| tan()                             | 1485                                                                                                  |
| tanf                              | 1485                                                                                                  |
| tanh()                            | 1487                                                                                                  |
| tanhf                             | 1487                                                                                                  |
| tanhL                             | 1487                                                                                                  |
| tanl()                            | 1485, 1489                                                                                            |
| tcdrain()                         | 1490                                                                                                  |
| tcflow()                          | 1492                                                                                                  |
| tcflush()                         | 1494                                                                                                  |
| tcgetattr()                       | 1496                                                                                                  |
| tcgetpgrp()                       | 1498                                                                                                  |
| tcgetsid()                        | 1500                                                                                                  |
| TCIFLUSH                          | 1494                                                                                                  |
| TCIOFF                            | 1492                                                                                                  |
| TCIOFLUSH                         | 1494                                                                                                  |
| TCION                             | 1492                                                                                                  |
| TCOFLUSH                          | 1494                                                                                                  |
| TCOOFF                            | 1492                                                                                                  |
| TCOON                             | 1492                                                                                                  |
| TCP_                              | 18                                                                                                    |
| TCSADRAIN                         | 1503                                                                                                  |
| TCSAFLUSH                         | 1503                                                                                                  |
| TCSANOW                           | 1503                                                                                                  |
| tcsendbreak()                     | 1501                                                                                                  |
| tcsetattr()                       | 1503                                                                                                  |
| tcsetpgrp()                       | 1506                                                                                                  |
| TCT                               | 8                                                                                                     |
| tdelete()                         | 1508                                                                                                  |
| TEF                               | 8                                                                                                     |
| telldir()                         | 1512                                                                                                  |
| tempnam()                         | 1513                                                                                                  |
| terminal access control           | 1496, 1504                                                                                            |
| terminal device name              | 1546                                                                                                  |
| terminate a process               | 309                                                                                                   |
| terminology                       | 1                                                                                                     |
| termios structure                 | 1496                                                                                                  |
| tfind()                           | 1508, 1515                                                                                            |
| tgamma()                          | 1516                                                                                                  |
| tgammaf                           | 1516                                                                                                  |
| tgammaL                           | 1516                                                                                                  |
| THR                               | 9                                                                                                     |
| thread creation                   | 1051                                                                                                  |
| thread creation attributes        | 981                                                                                                   |
| thread ID                         | 1055                                                                                                  |
| thread IDs                        | 51                                                                                                    |
| thread mutexes                    | 51                                                                                                    |
| thread scheduling                 | 52                                                                                                    |
| thread termination                | 1056                                                                                                  |
| thread-safety                     | 50, 375                                                                                               |
| thread-specific data key creation | 1071                                                                                                  |
| thread-specific data key deletion | 1073                                                                                                  |
| thread-specific data management   | 1065                                                                                                  |
| threads                           | 50                                                                                                    |
| regular file operations           | 58                                                                                                    |
| time()                            | 1518                                                                                                  |
| timer ID                          | 1523                                                                                                  |
| TIMER_                            | 17-18                                                                                                 |
| timer_                            | 17                                                                                                    |
| TIMER_ABSTIME                     | 49, 200, 1525                                                                                         |
| timer_create()                    | 1521                                                                                                  |
| timer_delete()                    | 1524                                                                                                  |
| timer_getoverrun()                | 1525                                                                                                  |
| timer_gettime                     | 1525                                                                                                  |
| TIMER_MAX                         | 1475                                                                                                  |
| timer_settime                     | 1525                                                                                                  |
| times()                           | 1528                                                                                                  |
| timezone()                        | 1530                                                                                                  |
| TMO                               | 9                                                                                                     |
| tmpfile()                         | 1531                                                                                                  |
| tmpnam()                          | 1533                                                                                                  |
| TMP_MAX                           | 1513, 1532-1533                                                                                       |
| TMR                               | 9                                                                                                     |
| tms_                              | 16                                                                                                    |
| tm_                               | 17                                                                                                    |
| toascii()                         | 1535                                                                                                  |
| tolower()                         | 1536                                                                                                  |
| TOSTOP                            | 327, 360, 416, 420, 443, 445, 1658                                                                    |
| toupper()                         | 1537                                                                                                  |
| towctrans()                       | 1538                                                                                                  |
| tolower()                         | 1539                                                                                                  |
| towupper()                        | 1540                                                                                                  |
| TPI                               | 9                                                                                                     |
| TPP                               | 9                                                                                                     |
| TPS                               | 9                                                                                                     |
| trace event, POSIX_TRACE_ERROR    | 77                                                                                                    |
| trace event, POSIX_TRACE_FILTER   | 77, 950                                                                                               |
| trace event, POSIX_TRACE_OVERFLOW | 77                                                                                                    |
| trace event, POSIX_TRACE_RESUME   | 77                                                                                                    |
| trace event, POSIX_TRACE_START    | 77, 959                                                                                               |
| trace event, POSIX_TRACE_STOP     | 77, 959                                                                                               |
| trace functions                   | 79                                                                                                    |
| trace-name attribute              | 76, 914                                                                                               |
| TRACE_EVENT_NAME_MAX              | 939, 941                                                                                              |
| TRACE_SYS_MAX                     | 936                                                                                                   |
| TRACE_USER_EVENT_MAX              | 939, 941                                                                                              |
| TRACING                           | 912, 914, 916, 919, 922-931<br>933, 935, 939, 941, 943-944, 946-948<br>950, 952-953, 956-959, 961-963 |
| TRAP_                             | 18                                                                                                    |
| TRC                               | 9                                                                                                     |
| TRI                               | 10                                                                                                    |
| TRL                               | 10                                                                                                    |

## Index

|                                   |            |                              |                     |
|-----------------------------------|------------|------------------------------|---------------------|
| trunc()                           | 1541       | UTC                          | 1549                |
| truncate()                        | 1542       | utime()                      | 1570                |
| truncation-status attribute       | 939        | utimes()                     | 1572                |
| truncf                            | 1541       | utim_                        | 17                  |
| truncf()                          | 1544       | uts_                         | 17                  |
| truncl                            | 1541, 1544 | ut_                          | 17                  |
| TSA                               | 10         | va_arg()                     | 1574                |
| tsearch()                         | 1508, 1545 | va_copy                      | 1574                |
| TSF                               | 10         | va_end                       | 1574                |
| TSH                               | 10         | va_start                     | 1574                |
| TSP                               | 10         | VDISCARD                     | 19                  |
| TSS                               | 10         | VDSUSP                       | 19                  |
| ttyname()                         | 1546       | Version 7                    | 119, 191, 669, 1557 |
| ttyname_r                         | 1546       | vfork()                      | 1575                |
| TTY_NAME_MAX                      | 1475, 1546 | vfprintf()                   | 1577                |
| tv_                               | 16-17      | vfscanf()                    | 1578                |
| twalk()                           | 1508, 1548 | vwprintf()                   | 1579                |
| TYM                               | 10         | vwscanf()                    | 1580                |
| tzname                            | 1549       | VISIT                        | 1508, 1548          |
| TZNAME_MAX                        | 1475       | VLNEXT                       | 19                  |
| tzset()                           | 1549       | vprintf()                    | 1577, 1581          |
| t_uscalar_t                       | 616        | VREPRINT                     | 19                  |
| ualarm()                          | 1551       | vscanf()                     | 1578, 1582          |
| uc_                               | 16-17      | vsprintf()                   | 1577, 1583          |
| UINT_MAX                          | 119, 1389  | vsprintf                     | 1577, 1583          |
| UIO_MAXIOV                        | 17         | vsscanf()                    | 1578, 1584          |
| ulimit()                          | 1553       | VSTATUS                      | 19                  |
| ULLONG_MAX                        | 1461       | vswprintf()                  | 1579, 1585          |
| ULONG_MAX                         | 1461, 1632 | vswscanf()                   | 1580, 1586          |
| UL_                               | 17         | VWERASE                      | 19                  |
| UL_GETFSIZE                       | 1553       | vwprintf()                   | 1579, 1587          |
| UL_SETFSIZE                       | 1553       | vwscanf()                    | 1580, 1588          |
| umask()                           | 1555       | wait for process termination | 1592                |
| uname()                           | 1557       | wait for thread termination  | 1067                |
| undefined                         | 2          | wait()                       | 1589                |
| underlying function               | 36         | waitid()                     | 1595                |
| ungetc()                          | 1559       | waiting on a condition       | 1037                |
| ungetwc()                         | 1560       | waitpid()                    | 1589, 1597          |
| unicast                           | 67         | WARNING                      | 386                 |
| unlink()                          | 1562       | warning                      |                     |
| unlockpt()                        | 1566       | OB                           | 6                   |
| unsetenv()                        | 1567       | OF                           | 7                   |
| unspecified                       | 2          | WCONTINUED                   | 1589, 1595          |
| UP                                | 11         | wcrtomb()                    | 1598                |
| US-ASCII                          | 627        | wcscat()                     | 1600                |
| user ID                           |            | wcschr()                     | 1601                |
| real and effective                | 1313       | wcscmp()                     | 1602                |
| setting real and effective        | 1313       | wcscoll()                    | 1603                |
| user trace event type definitions | 79         | wcscopy()                    | 1604                |
| USER_PROCESS                      | 284-285    | wcscspn()                    | 1605                |
| usleep()                          | 1568       | wcsftime()                   | 1606                |

|                                |                                                  |
|--------------------------------|--------------------------------------------------|
| wcslen()                       | 1608                                             |
| wcsncat()                      | 1609                                             |
| wcsncmp()                      | 1610                                             |
| wcsncpy()                      | 1611                                             |
| wcspbrk()                      | 1612                                             |
| wcsrchr()                      | 1613                                             |
| wcsrtombs()                    | 1614                                             |
| wcsspn()                       | 1616                                             |
| wcsstr()                       | 1617                                             |
| wcstod()                       | 1618                                             |
| wcstof                         | 1618                                             |
| wcstoimax()                    | 1621                                             |
| wcstok()                       | 1622                                             |
| wcstol()                       | 1624                                             |
| wcstold()                      | 1618, 1627                                       |
| wcstoll()                      | 1624, 1628                                       |
| wcstombs()                     | 1629                                             |
| wcstoul()                      | 1631                                             |
| wcstoumax()                    | 1621, 1634                                       |
| wcswcs()                       | 1635                                             |
| wcswidth()                     | 1636                                             |
| wcsxfrm()                      | 1637                                             |
| wctob()                        | 1639                                             |
| wctomb()                       | 1640                                             |
| wctrans()                      | 1642                                             |
| wctype()                       | 1643                                             |
| wcwidth()                      | 1645                                             |
| WEOF                           | 82, 650-651, 653-654, 656-663<br>1539-1540, 1560 |
| WEXITED                        | 1595                                             |
| WEXITSTATUS                    | 1590                                             |
| we_                            | 17                                               |
| wide-oriented stream           | 36                                               |
| WIFCONTINUED                   | 1590                                             |
| WIFEXITED                      | 1590                                             |
| WIFSIGNALED                    | 1590                                             |
| WIFSTOPPED                     | 1590, 1593                                       |
| wmemchr()                      | 1646                                             |
| wmemcmp()                      | 1647                                             |
| wmemcpy()                      | 1648                                             |
| wmemmove()                     | 1649                                             |
| wmemset()                      | 1650                                             |
| WNOHANG                        | 1348, 1589, 1595                                 |
| WNOWAIT                        | 1595                                             |
| wordexp()                      | 1651                                             |
| wordfree                       | 1651                                             |
| wprintf()                      | 468, 1655                                        |
| WRDE_                          | 18                                               |
| WRDE_APPEND                    | 1652                                             |
| WRDE_BADCHAR                   | 1653                                             |
| WRDE_BADVAL                    | 1653                                             |
| WRDE_CMDSUB                    | 1653                                             |
| WRDE_DOOFFS                    | 1652                                             |
| WRDE_NOCMD                     | 1652                                             |
| WRDE_NOSPACE                   | 1653                                             |
| WRDE_REUSE                     | 1652                                             |
| WRDE_SHOWERR                   | 1652                                             |
| WRDE_SYNTAX                    | 1653                                             |
| WRDE_UNDEF                     | 1652                                             |
| write to a file                | 1660                                             |
| write()                        | 1656                                             |
| writev()                       | 1664                                             |
| wscanf()                       | 477, 1666                                        |
| WSTOPPED                       | 1595                                             |
| WSTOPSIG                       | 1590                                             |
| WTERMSIG                       | 1590                                             |
| WUNTRACED                      | 1589, 1592                                       |
| XSI                            | 11                                               |
| XSI interprocess communication | 39                                               |
| XSR                            | 11                                               |
| X_OK                           | 98                                               |
| y0()                           | 1667                                             |
| y1                             | 1667                                             |
| yn                             | 1667                                             |
| zombie process                 | 307                                              |