Why You Can't Analyze RTOSs without Considering Applications and Vice Versa

Jörn Schneider

Dept. of Computer Science, Saarland University, Germany
E-mail: `js@cs.uni-sb.de`

Abstract

Traditionally WCET analysis tools are designed for the analysis of application code. The execution time of RTOS (Real-Time Operating System) services and the interaction between RTOS and application are usually not considered. When performing an RTOS aware schedulability analysis the WCETs of RTOS services are needed. At first sight the application of existing WCET analyzers on RTOS code should be straightforward and should deliver the same accuracy as for application code. The paper explains why this is not the case, and why the presence of an RTOS diminishes the accuracy of application code WCET-analysis.

In addition to explaining why RTOSs should not be analyzed without considering application code and vice versa, the underlying problems are identified as well as enlightened by some examples and possible solutions are sketched. Eventually a comprehensive approach for WCET- and schedulability-analysis is proposed.

1 Introduction

Current WCET analyzers [6, 2] aim at analyzing application code. The execution time of RTOS (Real-Time Operating System) services and the interaction between RTOS and application are usually not considered. The schedulability analysis is expected to consider the overhead due to the RTOS. However, the schedulability analysis needs at least the WCET of relevant RTOS services. Colin and Puaut made a first attempt to apply static program analysis to an RTOS [1]. They compute the WCET of some RTEMS [3] system calls and report several problems in applying their WCET analysis. For instance the loop bound of the RTEMS scheduler could not be derived because it depends on the number of task arrivals during its execution (the scheduler loops until no further arrivals are noticed). The average WCET overestimation reported is 86%. The reported problems originate mainly from a methodical weakness of their approach. As they regard the RTOS isolated from the application.

This paper explains why RTOSs should not be analyzed without considering application code and vice versa. The underlying problems are identified as well as explained by some examples and possible solutions are sketched.

2 Analyzing RTOSs

2.1 What are the problems?

The WCET of RTOS services is highly dependent on the application using them. Table 1 gives a systematic list of such dependences. Examples of such dependences can for instance be found in the RTEMS code, and in the code of osCAN (an OSEK [4] implementation by Vector Informatik).

Determining factors of the WCET of RTOS services	**Examples**
Non-constant call parameters in application code	Any service with call parameter dependent control flow
Static, application dependent configuration parameters	Any service with loop bounds depending on no. of RTOS objects (e. g. tasks, resources)
Cache state	Replacement of RTOS owned cache sets by application code
Calling history of RTOS services	Scheduler execution after disabling preemption
Calling context	Call to system service from task, interrupt or operating system level

Table 1. Sources of WCET variations of RTOS services.

2.2 How can these problems be addressed?

Non-constant calling parameters When system calls are analyzed as part of the application, any knowledge about parameter values (e. g. obtained by a value analysis [2]) can be used to derive sharper bounds on the WCET.

Static configuration parameters The configuration parameters (e. g. number of tasks and memory mapping of tasks) are fixed for a particular application. Therefore, they can be considered either manually or automatically by WCET analysis as well as schedulability analysis.

Cache state If the cache is not partitioned in a special way, application code or data might displace cache sets occupied by the RTOS. No isolated WCET analysis of the RTOS can therefore benefit from positive cache effects caused by previous runs of RTOS services. It might even be impossible to consider the positive intrinsic cache effects of RTOS services. RTOSs are usually designed to minimize the number and duration of non-interruptible code sections. It is impossible for an isolated analysis to predict the negative impact of application interrupts outside these few code sections, unless all positive cache effects are ignored. In the case of a combined analysis it is possible to bound the effect of application caused cache replacements as it has been shown in [5] for the application analysis.

This is not only a question of the schedulability analysis. Depending on the application it can be beneficial, and for certain modern CPU types even necessary, to consider the preemption related cache effects within the WCET analysis (cf. [5]).

Calling history of RTOS services It not only affects the WCET of RTOS services, but often has an immediate impact on the task response time as well (e. g. an RTOS service called to disable preemption eliminates the subsequent interference by other tasks). A good schedulability analysis should consider these effects. Therefore, the history information should be statically predicted anyway and can also be used by WCET analysis.

Calling context When using a combined analysis, the calling context can easily be regarded. The WCET analysis can for instance consider infeasible paths for the specific calling context.

3 Analyzing applications

This section discusses problems arising in presence of multitasking RTOSs. The two subsection treat the problems in the same order. First the problem domain of data values is considered. Issues that arise due to isolated analysis of application and RTOS code come second. Not all of these issues can be addressed by a mere integration of application and RTOS WCET-analysis. The last parts of either subsection and Section 4 cope with this enigma.

3.1 What are the problems?

The data values used in application code can play a large role in computing the WCET.[1] Examples are: loop bounds, addresses of memory references and infeasible paths. For the WCET-analysis to profit from this fact a static prediction of value ranges is necessary. The value analysis described in [2] provides this functionality for instance. However, tools of this kind are—like any available WCET tools—designed for sequential programs. The following issues arise in presence of a multitasking RTOS:

Shared application memory Accesses by other tasks may change the value of data in such areas.

RTOS data structures Any RTOS data structure not unique to the analyzed task might be changed by RTOS services called in other tasks. Even data structures unique to a task might be manipulated by other (user or RTOS) programs.

Memory mapped I/O The values read from those areas are mainly determined by the environment and accesses are non-cachable.

The WCET analysis of application code should consider the WCET of the system calls used. A seemingly attractive approach is to initially ignore the system calls within the application WCET analysis and thereafter add system call WCETs obtained by an isolated analysis of the RTOS. However, there are good reasons not to do so (see Table 2).

There is a significant difference between problem descriptions 1 through 3 and the classes of problems alluded to by description 4 of Table 2. The former difficulties occur also together with the isolated WCET-analysis of library functions, the latter not.

3.2 How can these problems be addressed?

Because of shared application memory and RTOS data structures, a value analysis has either to ignore such data, or has to be enhanced to a multi-task-analysis. The latter is not trivial. Memory mapped I/O areas have to be excluded from the value analysis since they are volatile.

[1]This holds for RTOS code also. Nevertheless the subject is discussed in this section because that is where WCET-analysis comes from and because applications are usually more data-driven than RTOSs.

No.	Problem description
1	RTOS WCETs are systematically overestimated (shown in Section 2)
2	Information about correlation of worst-case paths and number+context of RTOS calls is destroyed $\Rightarrow$ only a pessimistic approach can still deliver conservative WCETs
3	Cache and pipeline effects caused by RTOS calls cannot be considered in application WCET
4	It is impossible to consider positive effects of concepts existing only in presence of multi-tasking RTOSs, for instance RTOS calls dynamically raising the application priority (e. g. by disabling preemption or interrupts, or by occupying resources)

Table 2. Problems of analyzing applications isolated from the RTOS.

Section 2 shows that the WCET of RTOS services depends on the *call situation*. This call situation subsumes the factors given in Table 1. Some aspects of the call situation are not unique to applications running on RTOSs. These aspects can be identified already, when stand-alone applications with calls to library routines are considered (one could replace the word *RTOS* with *library* in the problem descriptions number 1 through 3 of Table 2 and the statements would still be true to some extent). Aspects like these can be addressed by embedding the analysis of library/system calls within the application WCET analysis. The embedding can be implicitly or explicitly. Embedding implicitly means that the calls are treated like an ordinary function call. The input data structure (e. g. the control flow graph) of the WCET analyzer contains all needed information to analyze such calls. If the WCET analyzer uses machine code as input (e. g. the one described in [2]), this can even be done without providing the user with the library/RTOS source code. Embedding explicitly means that two independent WCET analyzers (or two instances of the same analyzer) are used, one for the application and one for the RTOS. The RTOS WCET analyzer can be a black box that takes the code of the RTOS service as well as collected information about calling parameters, static configuration parameters, cache state and calling context as input (see Subsection 2.2).

However, introducing an RTOS in the considered scenario adds completely new qualities to the problem of WCET analysis (represented by item no. 4 of Table 2). These are issues that cannot be addressed by a mere integration of application and RTOS WCET analysis. Rather a high-level view is needed to consider them in WCET and schedulability analysis.

Several such high-level concepts can be identified that co-determine the temporal behavior of the tasks of an RTOS-based system. These concepts are for instance the effective priority of tasks, the RTOS mode (e. g. initialization or normal operation mode), application modes, task states, and the system level (task, interrupt or RTOS level). Those high-level concepts have a certain meaning when viewing the system as a whole rather than as a bunch of independent programs at a microarchitectural level. At run-time the properties of these concepts have a defined state at each point in time. We define the *meta-state* of a task to be the set of these states.

In a static approach, at best partial knowledge of the meta-state of a task can be obtained. To address the RTOS specific problem domain, partial knowledge can be collected which allows to compute the worst-case response time of tasks more accurately. This includes exploiting meta-state information to compute sharper bounds on WCETs of tasks as well as sharper bounds on microarchitecture-related preemption costs.

4 Proposal for a comprehensive WCET- and schedulability-analysis approach

The authors present work on a comprehensive WCET- and schedulability-analysis approach exploits meta-state information to compute sharper bounds on WCETs of jobs (tasks and interrupt service routines) and interesting code sections and on microarchitecture-related preemption costs. The framework exploits the following aspects of the meta-state of a job: effective priority of a job (determined by: locked interrupts, preemption lock, occupied resources), RTOS mode and current system level. The meta-state information is exploited as follows:

1. Extrinsic cache effects are considered by the cache analysis (which is a part of the WCET analysis) in dependence of the effective priority at each program point of the analyzed job.

2. Pipeline-related preemption costs are individually computed for each job, again in dependence of the effective priority, and are considered during the schedulability analysis.

3. The WCET of jobs and code sections is computed for the proper RTOS mode (initialization mode or normal operation mode) and for each RTOS mode a separate schedulability analysis is undertaken.

4. The current system level is considered when the WCET of system calls is computed.

Similar to the cache- and pipeline-sensitive schedulability analysis described in [5] the cache-related preemption costs are incorporated in the WCET while the pipeline-related preemption costs are explicitly considered during the schedulability analysis.

The above sketched framework uses the WCET analysis tool described in [2]. The WCET analyzer is loosely coupled with the surrounding tools. It is guided with the help of the obtained meta-state information in order to compute sharper bounds on the WCET and the microarchitecture-related preemption costs. A detailed explanation of this method is beyond the scope of this paper.

5 Conclusion

The paper showed that analyzing WCETs of RTOS-based real-time systems—whether of RTOS services or of application code—requires other than the established approaches. The underlying problems were explained by examples and classified. Additionally it was sketched how the individual problems can be addressed. Finally a comprehensive approach for WCET- and schedulability-analysis was proposed. The proposed approach shows how it is possible to overcome most of the obstacles obstructing the path toward comparative results of WCET-analysis for RTOS-based systems.

Acknowledgements

Many members of the compiler design group at the Universität des Saarlandes, especially the members of the USES (Universität des Saarlandes Embedded Systems) group, deserve acknowledgement. Reinhard Wilhelm, Daniel Kästner, and Stephan Diehl carefully read draft versions of this work and provided many valuable hints and suggestions.

I would like to thank the anonymous reviewers for their helpful comments.

References

[1] A. Colin and I. Puaut. Worst-Case Execution Time Analysis of the RTEMS Real-Time Operating System. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, pages 191–198, Delft, The Netherlands, June 2001.

[2] C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm. Reliable and Precise WCET Determination for a Real-Life Processor. In *Embedded Software Workshop*, Lake Tahoe, USA, Oct. 2001.

[3] On-Line Applications Research Corporation, Huntsville, AL, USA. RTEMS Applications C User's Guide. Edition 4.0, Oct 1998. `http://www.oarcorp.com/RTEMS/rtems.html`.

[4] OSEK/VDX – Open systems and the corresponding interfaces for automotive electronics. OSEK/VDX Operating System. Version 2.2, Sept. 2001. `http://www.osek-vdx.org`.

[5] J. Schneider. Cache and Pipeline Sensitive Fixed Priority Scheduling for Preemptive Real-Time Systems. In *Proceedings of the 21st IEEE Real-Time Systems Symposium 2000*, pages 195–204, Nov. 2000.

[6] J. Schneider and C. Ferdinand. Pipeline Behavior Prediction for Superscalar Processors by Abstract Interpretation. In *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems*, volume 34 of *ACM SIGPLAN Notices*, pages 35–44, May 1999.