

# A Verified POSIX Flash File Storage System

*Leo Freitas, University of York  
ONFI Seminar @ Intel, Hillsboro U.S.  
April, 2nd, 2009*

## Outline

### *Verification GC: context, motivation, goals*

- *what makes a pilot project?*
- *pilot projects: past and current*
- *players and achievements*

### *Verified POSIX flash file store*

- *original motivation: JPL's Mars rover FS anomaly*
- *mechanisation (e.g., theorem proving) of various models*
- *strands of work and related projects*
- *embedded OS kernels and security hypervisors*
- *certification + evaluation processes*

## Outline (cont.)

### *Results summary*

- *file store: POSIX (func.) + CICS (fault-tolerance)*
- *hardware: ONFI (hw standard) + NVMeHCI (device drivers)*
- *OS kernel: Kernel for embedded devices + Xenon hypervisor*

### *Conclusions*

- *work in progress*
- *future collaboration / proposals*
- *suggestions + criticisms + discussion*

## Hoare's Verification Grand Challenge (GC6)

**A mature scientific discipline should set its own agenda and pursue ideals of purity, generality, and accuracy far beyond current needs**

### what should we do?

- *achieve a significant body of verified programs*
- *precise external specifications*
- *complete internal specifications*
- *machine-checked proofs of correctness*
- *a collection of verified programs, e.g.,:*
  - *6,000,000 lines (e.g., Cousot's Airbus A380 FCSW)*
  - *replacing existing unverified ones (e.g., UNIX-POSIX)*

## Goals of the Repository

1. *accelerate development of verification technology*
2. *provide focus for verification community*
3. *provide open access*
4. *collect challenging applications*
5. *identify key metrics*
6. *enumerate challenge problems*
7. *standardise formats*
8. *define quality standards*
9. *curate gathered information*

<http://vsr.sourceforge.net/gc6index.htm> **[York]**

<http://qpq.csl.sri.com> **[SRI]**

## Deliverables — VSTTE

1. **theory: a comprehensive theory of programming**
  - *covering features to build practical and reliable programs*
  - *linking theories for tool reuse*
2. **tools: a coherent toolset**
  - *automating the theory and scaling up to large codes*
3. **experiments: a collection of verified programs**
  - *e.g., Mondex Electronic Purse*
  - *e.g., space flight file store*
  - *e.g., variety of verified components*

## Pilot projects

- *Mondex*
- *Verified File Store*
- *Operating System Kernels*
- *Xenon High Assurance Hypervisor*
- *Tokeneer ID Station*
- *Radio Spectrum Auctions*
- *Cardiac Pacemaker*
- *Free RTOS*
- *and others: TCP/IP, pointer-rich C-code analysis, Apache, etc.*

**\* over 80 publications (20 journal, 60 conference) [Jun/08]**

## The POSIXcompliant file-store

- Amir Pnueli's challenge: **verify the Linux Kernel**
- Joshi & Holzmann restricted this to

*verify a small subset of the POSIX interface suitable for flash-memory hardware with strict fault-tolerance requirements to be used by forthcoming NASA missions*

- **robustness requirements for fault-tolerance**
  - *no corruption from unexpected power loss*
  - *recovery from faults specific to flash hardware*
- *NAND flash hardware has interesting issues*
  - *physical wear-and-tear of device*
  - *space reclaim and wear-levelling algorithms*

## Roadmap strands of work — overview + achievements

- POSIX functionality
  - ✓ Open group IEEE POSIX stand.
  - ✓ legacy (formal Z) models of APIs
    - *security, access control, and concurrency features*
- fault tolerance
  - ✓ IBM CICS — file transaction processing
    - reclaim and garbage collection algorithms
    - flash translation layer — hardware abstraction
    - recovery on unexpected power loss
      - \* in hardware — spare areas
      - \* in software — FSM

## Strands of work in the roadmap — overview

- architecture

- Intel's (8) layers — orthogonal composition:

- ✓ file system functionality (JPL subset)

- \* various degrees of fault tolerance

- \* hardware interface/interaction models

- \* real-time operating system concerns

- top-down development from API to code:

- linking functionality and fault tolerance

- hardware

- Open NAND flash interface (ONFi) standard

- flash hardware modelling — FSM on hardware behaviour

- NVMHCI standard — NAND device drivers

## Verified file store (the players) — [2007–]

- *Joshi & Holzmann (JPL) — Spin*
  - *original problem, serious development effort*
- *Woodcock & Freitas (York) — Z/Eves*
  - *specifications, refinements, analyses in Z*
- *Butler (Southampton) — Event-B + Rodin*
  - *Event-B, Rodin toolset, hierarchical file system*
- *Oliveira (Minho) — VDM-tooset*
  - *hardware models in VDM*
- *Jackson (MIT) — Alloy*
  - *model & analysis of flash file system*

## 1800 APIs, where shall we start?

*Proposal: orthogonally layered approach (Intel inspired)*

- *functionality modelling*
- *hardware interfacing*
- *fault-tolerance guarantees*
- *risk/hazard analysis*

### *JPL minimal interface*

*create, open, close, unlink, read, write, truncate, ftruncate,  
stat, fstat, mkdir, rmdir, rename, opendir, readdir, rewinddir,  
closedir, format, mount, unmount*

## What to avoid? — JPL suggestions

- *user/group/other file permissions*
- *hard- or symbolic-links*
- *sockets, pipes, networking (i.e., have just plain files)*

## In the spare time . . .

- *user-level operations*
  - *encryption*
  - *directory contents listing*
  - *operations with regular expressions*
- *multi-threading and real-time*
- *generic interface for most NAND devices*

## What is our ambition?

- *initials scope: POSIX subset enough for flash memory*
- *whole XNFS (network file system) interface*
- *as much user utilities as possible*
- *top-down development from user-level API's to flash hardware*

## Previously opened questions:

- **is the whole POSIX to be a target?** (e.g., yes)
- **standard for UNIX domain modelling?** (e.g., Yeow's work)

## What is JPL's ambition? — fault-tolerance issues

- *NAND flash hardware faults*
  - *bad blocks and read errors*
- *reset robustness*

**“no corruption in the presence of unexpected power loss”**

- *JPL multi-threading disclaimer: performance traded for simplicity*

**“we make the very conservative guarantee that the result of executing concurrent filesystem operations is equivalent to executing them in some serial order”**

## What do we want?

### *Formal model of POSIX/UNIX*

- *functional model*
- *hardware requirements*
- *risk analysis and fault-tolerance dependability*
- **do we intend to aim at coding/prototyping?**

### *JPL using our work*

- *minimal subset first*
- *fault-tolerance in hostile environment*
- *flash memory hardware issues (i.e., balance levelling)*

## Some results over the last two years

- **abstract specification** of POSIX interface in Z/Eves
  - mechanisation of Morgan & Sufrin's Unix filing store (*S*)
  - mechanisation of Patrick Place's specification of POSIX1003.21 Real-time distributed systems communication (1/2 *S*)
  - IEEE POSIX Working Group interface requirements
- **concrete implementation** in Z hashmaps lifted from JML
- **abstract file maps, B<sup>+</sup>-tree implementation** (e.g., Z-to-VDM)
- domain modelling for flash file stores
  - specification patterns
  - desirable (overall) properties
  - intricate data structures

## Introduction to CICS — IBM Hursley's Reports

- *IBM's transaction processing system providing:*
  - *high availability, integrity, and performance*
  - *reliability, and scalability*
- *most important of services are spread across sets of APIs:*
  - *continuous operation and parallel execution from multiple users*
  - *connectivity with database management systems*
  - *built-in facilities for ensuring data integrity*
  - *failure recovery and transaction back out*
- *they have been formally specified in Z*
- *original work won a Queen's Award for design excellence*

## CICS within formalisation of POSIX file-stores

- *part of next GC pilot project suggested by NASA's JPL*
- *POSIX documentation guideline*
  - *standards with formal specification*
  - *Morgan & Sufrin's UNIX file store*
- *thinking of CICS as a transaction processing interface to a file store*
- *cherry-picking CICS APIs that are related to file systems*
  - *mechanisation of the APIs using **Z/Eves***

**MSc project for whole APIs: to be used in the GC POSIX pilot project**

**TODO: refinement proof of their link with lower level specs.**

## CICS File Control API — mechanisation

- *mechanisation in Z/Eves*
  - *syntactic and type checked File Control specification*
  - *domain checked  $\Rightarrow$  well defined*
  - *calculated precondition for all operations*
- *discussion on the original specification*
  - *few typos and type bugs, and missing free type constants*
  - *we kept faithful to original spec. style*
    - \* *more difficult to mechanise*
    - \* *yet gave insight in some murky areas*
    - \* *reusable from other challenges (i.e.,  $\mu$  —  $\theta$ -one-point)*

## CICS file control API — experimental (student) results (2006)

- 223 paragraphs, 84 operation (split in their cases)
- 73 precondition proofs (11 still to proof)
- 118 rewriting rules and theorems
- all discharged with 1213 proof commands
  - 56% no creativity, but just button clicks
  - 26% some knowledge of which rewrite rule to choose/click
  - 18% more creative steps (i.e., existential instantiations)
- **5 months work by an MSc student - no previous experience!**
- quite nice learning curve when compared with other (Z) provers
- **one SCP journal paper on the main results**

## CICS task control API — experimental (student) results (2007)

- 218 paragraphs, 23 operation (split in their cases)
- 13 precondition proofs (10 still to proof)
- 121 rewriting rules and theorems
- all discharged with 1416 proof commands
  - 58% no creativity, but just button clicks
  - 21% some knowledge of which rewrite rule to choose/click
  - 21% more creative steps (i.e., existential instantiations)
- **4 months work by an MSc student - no previous experience!**
- quite nice learning curve when compared with other (Z) provers
- **one conference paper on the main results**

## What have we learned?

### *General theorem proving laws*

- *reuse of laws from Mondex and hashmaps from POSIX spec.*
- *great similarity in proofs and strategies (e.g., similar domain)*
- *reuse of (general) laws  $\Rightarrow$  greater automation*
  - *finiteness: sets and schema bindings*
  - *free types: injectivity of constructors*
  - *schema calculus: surgical expansion*

### *Strengthen the open source **Z/X!***

- *Canadian government negotiation for release of **EVES'** licence*
- **looking for potential backend engines for Z/X**
- *recent talks with ACL2 community in Austin, Texas*

## Overall architecture

- *Intel's flash memory top-level design / APIs*
- *corroborated with JPL's interests / goals*
- *some work with research group in Brazil (still incipient, though)*
- *APIs as (refinement) targets for the various mechanisations*
- *implementing the API's is still to be done*
- **Comments, suggestions, objections?**

**see Intel's architecture document...**

## Flash device drivers and ONFI (2008)

- **mechanised model of flash memory standard (ONFI)**
  - *pages, blocks, logical units, targets, devices*
  - *memory addressing, defect marking*
  - *mandatory command set: reset, read, write, protect, page program, change read/write column, block erase*
- **device driver correctness (NVMHCI)**
  - *C programs — found 5 issues (possibly errors?)*
  - *discovering assertions using Daikon (MIT)*
  - *inspiring fresh work using ILP (inductive logic programming)*

## Hardware and kernel-level functionality

- *verification of NAND device driver's C code (2000 lines)*
- *continue verification of ONFi standard*
- *integration between top-level models and actual (C/VHDL) code*
- *wear-levelling algorithms minimise failure rate*
- *modelling of reclaim algorithms*
- *flash devices prone to unrecoverable block failure*
- *fault tolerance must be organised by host*
  - **workload-related aging**

## Current work @ York

*three MSc in 2008.1 / .2*

- *analysis of 2 flash (production) device drivers*
- *operating system kernel formalisation*
- *file system structures domain modelling*

*two PhD from 2009.1*

- *continued work on flash device driver verification*
- *use of Denali's MMV and ModelSim tools*
- *invariant detection and run-time exception freedom techniques*

## Operating system kernels — [2008–]

- *Pnueli's original pilot project suggestion: the Linux kernel*
- *possible departure points*
  - *Practical File System Design (Dominic Gianpaolo)*
  - *Formal Refinement for Operating System Kernels (Iain Craig)*

### *Related Grand Challenge pilot projects*

- *free RTOS (open-source, but no specification)*
  - *widely used real-time operating system kernel*
  - *pointer-rich implementation*
- *Xenon hypervisor (high-assurance open-source separation kernel)*
  - *modelling/verification of abstract security policy using Circus*
  - *modelling/verification of critical concrete C-code parts*

## A verified simple OS kernel

- **pilot project for the grand challenge in verified software**
- *Iain Craig (Northampton)*
  - *operating system kernel domain expert*
  - *modeller*
- *Leo Freitas & Jim Woodcock (York)*
  - *verifiers*
- *exploratory phase*
- *verified domain models*

**see Iain's book...**

## Operating system kernels

- **kernel:** *central component of most operating systems*
- *manages hardware/software resource interface: lowest-level abstraction layer for memory, processors and I/O devices*
- *provides facilities to applications processes through inter-process communication mechanisms and system calls*

## *Kernel features*

- *low-level scheduling of processes (dispatching)*
- *inter-process communication*
- *process synchronisation*
- *context switching*
- *manipulation of process control blocks*
- *interrupt handling*
- *process creation and destruction*
- *process suspension and resumption*
- *targeting embedded devices*

## *Kernel development*

- **very difficult and complex programming task**
- *critical component*
  - *correct functionality and good performance*
- *can't use many abstractions that simplify programming*
- *typical for embedded and real-time systems*

## OS kernel pilot project

- *Iain D. Craig Formal Refinement for Operating System Kernels, Springer, 330pp, 2007*
- **objectives: demonstrate**
  - *feasibility of top-down OS kernels development*
    - \* *formal specification, refinement, implementation*
    - \* *Z notation, Dijkstra's guarded-command language*
    - \* *hand-written proofs*
- **pilot project objectives: investigate**
  - *tractability of mechanising all models*
  - *tractability of formalising all proofs*
  - **feasibility of a tool chain**
    - \* *Z/Eves  $\longrightarrow$  ZRC-Refine/Gabriel  $\longrightarrow$  Spec#-Boogie/PL*
  - *curation of results in verified software repository*

## Z/Eves “waterfall”

- *prop. calc. + congruence closure + linear arithmetic*
- *types + type prescription (grules) + proof context*
- *forward and backward chaining + compound recognisers*
- *unconditional + conditional rewrite rules*
- *normalisation + subsumption + equality substitution*
- *expansion is up to the user + left-to-right reasoning*
- *not much heuristics over what to do with certain terms*
- *great handling of schema calculus and quantifiers*
- *axiomatised (non-computable) set-theoretical abstractions*
- *limited / restricted induction schemes*

## Z/Eves architecture

- *front-end for Spivey's Z for Lisp (EVES) FOL back engine*
  - *ZF set theory to FOL with quantifiers*
  - *tailored for the Z schema calculus*
- *typechecking + domain checking + interactive proof*
- *axiomatic, generic, abstract type definitions ( $\text{\LaTeX}$ + XML)*
- *Python, Emacs, and shell interfaces: Win, Linux, Mac, Sparc*
- *fast and easy-to-use (e.g., student projects ex.)*
- *support for modules for specification and proofs*
- *questionably sound, yet rigourously built*
- *lack of exposure due to irrelevant secrecy from its contractor*

## Proving theorems with Z/Eves

- *can handle pretty large (industrial-scale) specifications*
- *abstract (general) + concrete (finely guided) proof scripts*
- *highly automated handling of Z toolkit expressions*
- *nicely spread proof effort*
- *70% blind (easy); 15% aware (medium); 15% creative (hard)*
- *easy to learn: students with no previous experience take sizeable experiments in around four months*
- *largest specs known: Mondex*
  - *200 definitions, operations, schemas*
  - *320 lemmas, theorems, automation rules*
  - *4700 proof steps for the 320 proofs*
  - *runs all just over 2 hours*

## Z/Eves case studies: 2002—2009

- 2003: *Specialised automata theory (225/205/?)*
- 2004: *Refinement calculus automation (86/91/?)*
- 2005: *Circus model checker + op. semantics (200/150/?)*  
*programmable link with Java and CZT (2Kloc)*
- 2006: *Mondex smartcards (200/220/4700)*  
*IBM CICS file (216/120/?) + task (218/121/1416) APIs*
- 2007: *Unix file stores (200/100/1200) + part of POSIX XNFS Std (150/100/1000)*  
*Hashmap +  $B^+$ -tree + JML interfaces + journaling flash FS +*  
*ONFI flash hardware interface (58/64/?) + NVCHMI device drivers (2 Kloc)*
- 2008: *OS kernels embedded systems (?) + Xenon high-assurance hypervisor (?)*
- 2009: *Tokeneer ID station*
- 2002-2009: *set-theoretical col. of lemmas (-/118/?)*

## Conclusions and discussion

- *GC work instigates experiments and usage of tools*
- *community effort is necessary for success*
- *good opportunity for papers on tools and experimental work*
- *CbyC achievements via modelling and refinement*
- *work at various levels (i.e., from requirements down to code)*
- *engaging the research community*
- *work refereed and exposed to various venues*
- *good body of specified material + proved properties of interest*

## Future collaboration

- *how could we work together?*
  - *student internships*
  - *research collaboration*
  - *support for research proposal*
  - *evaluation of particular results*
- *information exchange + foreseeable issues*
- **any problem suggestions?**
- **comments, suggestions, observations?**

<http://vsr.sourceforge.net> **[York]**

<http://qpq.csl.sri.com> **[SRI]**