seL4: Formal Verification of an OS Kernel

Marek.Sapota@students.mimuw.edu.pl

December 15, 2010

What is seL4?

- Secure Embedded L4
- Formally verified kernel from the L4 family
- Runs on ARMv6 and x86 architectures (there is no proof for x86)
- “Complete, general-purpose operating-system kernel”
- 8‘700 lines of C and 600 lines of assembler
- Microkernel that provides address spaces, threads, IPC and authorisation capabilities

What is proved?

- Behaviour precisely specified at abstract level
- Implementation is proved to follow abstract level specification and do nothing more
- Proof of termination and execution safety

What is not proved?

- Abstract specification making any sense
- Device drivers (not being part of the kernel)
- Memory allocation
- Several low level parts implemented in assembler
- Boot code
- Compiler correctness
- Hardware correctness
- Theorem prover correctness

What does it give us?

- It's impossible to make a buffer overflow attack
- No NULL pointer dereferencing
- No accidental uses of pointers to wrong type of data
- No memory leaks
- No arithmetic overflows and exceptions in kernel code (eg. no division by zero errors)

About proof

Proof is done using Isabelle/HOL theorem prover. There are several proof assumptions:

- Valid compiler (GCC)
- Hardware correctness
- Theorem prover is correct

Proof was only done for ARMv6 architecture with one CPU.

About proof

The proof and implementation is split into three phases:

- Abstract specification — system interface
- Executable specification — Haskell prototype
- C implementation

Abstract specification

Defines the system interface, describes what the system does, but not how. Every implementation refining this specification will be binary compatible with seL4.
Properties:

- 4‘900 lines of Isabelle proof
- Enough detail to specify outer interface of the kernel
- No implementation details, data structures are high level (sets, lists, trees, etc.)

Haskell prototype

Executable specification fills in details left by abstract specification and describes how the system works.
Prototype properties:

- 5‘700 lines of code, 13‘000 lines of Isabelle proof
- Runnable with modified QEMU
- Deterministic (only left non-determinism is hardware)
- Proved to implement abstract specification
- Automatically translated to Isabelle script

Haskell prototype

There are some restrictions that make automatic translation to Isabelle possible:

- No substantial use of laziness
- Restricted use of type classes
- All functions are proved to terminate

C implementation

C implementation is a manual reimplementation of Haskell code.

- Haskell runtime is much bigger than the kernel itself and would be hard to be proved correct
- Garbage collection is not suitable for real-time environments
- C allows low level optimisations
- Automatic translation would be easier to prove, but it wouldn't allow any low level optimisations

Properties:

- 8'700 lines of code, 15'000 lines of Isabelle proof

Assembler parts

For some direct interaction with hardware assembler is used. For example cache and TLB flushes require direct instructions to hardware. This low level parts of code are not proved correct, but they are traditionally tested.

Lines of code

- 5‘700 lines of Haskell
- 8‘700 lines of C
- 600 lines of assembler
- Total of 200‘000 lines of Isabelle script (including generated proofs)

Time

- Abstract specification - 4 person months
- Haskell prototype - 2 person years
- C implementation - 3 person months
- Total of 2.2 person years
- SLOCCount estimates cost of seL4 implementation at 4 person years, in comparison it took 6 person years to implement Pistachio kernel
- Cost of proof - 20 person years (9 person years for tools, frameworks and automation, 11 person years for seL4 specific proof)
- All together — design, documentation, implementation and verification — about 25 – 30 person years, to do this again it would be about 10 person years

Conclusions

- Formal verification is achievable for microkernels
- Performance does not need to be sacrificed for verification
- Detour via Haskell improved productivity

What next?

- Verification of assembly parts
- Version of kernel for multi-CPU machines