A Formally Verified Runtime

Bernhard Egger

2013/02/01

Computer Systems and Platforms Laboratory
School of Computer Science and Engineering
Seoul National University

A Smart New World

- everything is "smart" these days
  - most noticeable feature:
    user-generated content on devices
  - less noticeable:
    increased complexity of hardware
    increased complexity of software

CSAP Computer Systems and Platforms Laboratory

A Smart New World – And a Host of New Problems

- we run more and more security- and safety-critical stuff on our smart devices
    - mobile banking
    - smart wallets
    - personal health monitoring

- and then there's the issue of privacy

I Won’t Be Evil – Just Trust Me

- prevalent model for smartphones is based on trust

- the trust model of Android and iOS
  - limited privileges per default
  - safe programming language (Android - Java)
  - walled gardens (app stores), app permissions

- lots of problems with this approach
  - root exploits
  - native code
  - verification, permission difficulties

I Can’t Be Evil – Here’s The Mathematical Proof

- ideal world: entire system is verified
  - not (yet) possible
- realistic world: use a verified base to run software on
  - formally verified kernels/hypervisors
    - Verisoft, seL4, SecVisor
  - formally verified compilers
    - CompCert C compiler, C0 compiler
  - run secure/insecure S/W alongside

Pioneers – Secure Unix, PSOS, KIT

- UCLA Secure Unix (1980)
  - microkernel-like structure
  - implemented in Pascal
  - used (an early variant of) formal refinement
  - proof for kernel only, but “tedious and painful”
  - performance up to an order of magnitude below par
- Provably Secure Operating System (1973-80)
  - considers the entire OS
  - layered approach

Pioneers – Secure Unix, PSOS, KIT

- KIT (kernel for isolated tasks, 1989)
  - first formally verified kernel
  - ~600 lines of assembler source code
  - provides (static) task switching, async I/O, exceptions, message-passing

Verisoft / VerisoftXT

- Verisoft (2003-2007)
  - government funded (~15m euro, 250 억원), partly confidential
  - pervasive
    - does not rely on the correctness of the ISA or compiler
    - unbroken chain from H/W to applications
  - layered approach similar to PSOS
  - theorem prover: Isabelle/HOL

Verisoft / VerisoftXT

- Verisoft (2003-2007)
  - hardware layer: VAMP processor
    - formally verified down to the gate level
  - kernel layers:
    - CVM (Communicating Virtual Machines)
    - VAMOS
  - user mode layers
    - SOS (Simple Operating System) – privileged process
    - applications

Verisoft / VerisoftXT

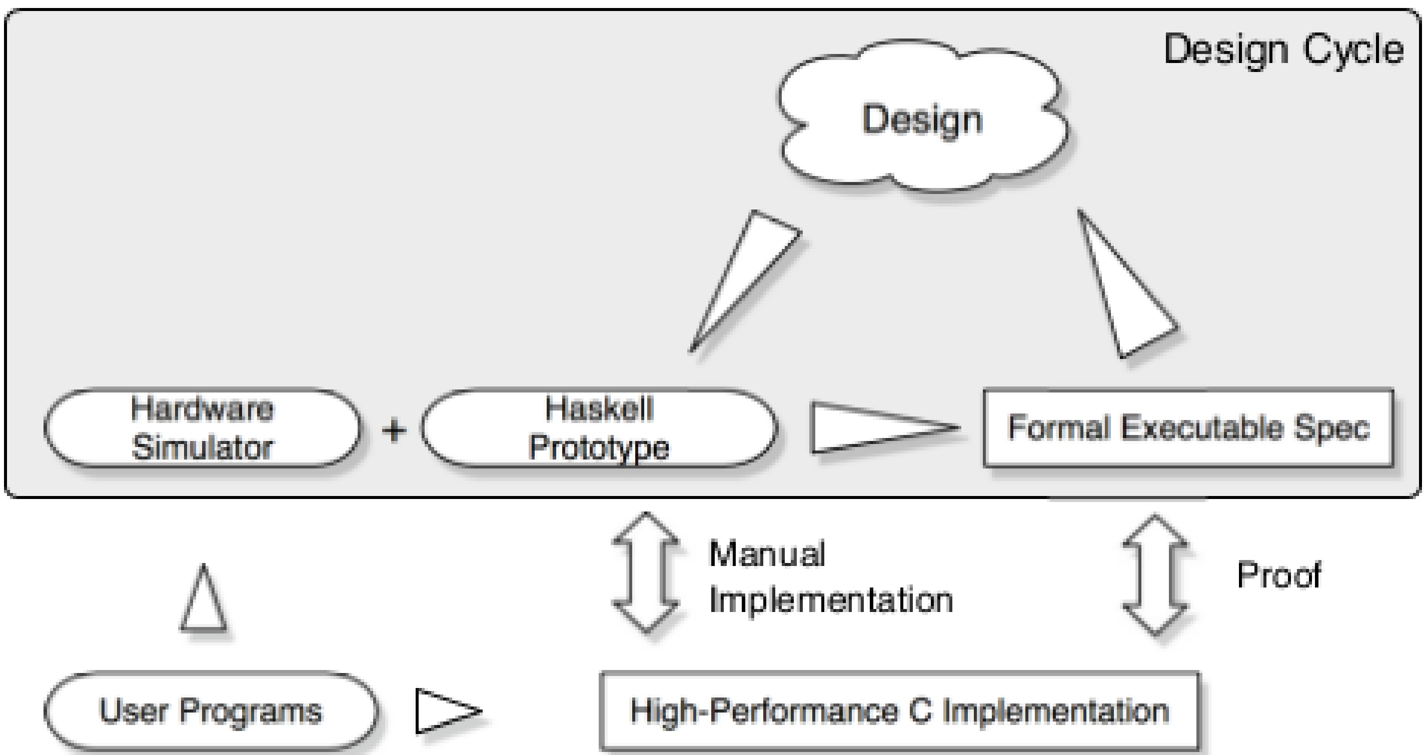
- Verisoft (2003-2007)
  - verified compiler tool chain
    - C0, C0A language
    - bootstrapped
  - code vs proof:
    - compiler source: 1’500 lines of C0 code
    - proof: 85’000 lines of Isabelle code
- successor: VerisoftXT (2007-2010)

seL4

- ongoing joint project of people at NICTA, UNSW, and Open Kernel Labs
- production-quality, commercialized general-purpose microkernel
- formally verified from abstract specification down to C implementation
  - assumes correctness of
    - hardware, boot code, assembly code, compiler
  - everything else is proven

seL4

- interesting from the design/proof/implementation point of view
  - Haskell prototype
    - intermediate target suitable for formal methods and implementation

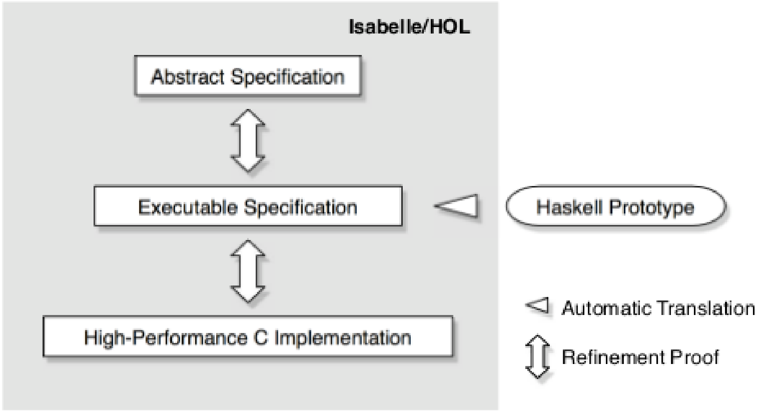


source: Klein, 2009

seL4

- formal verification
  - interactive, machine-assisted, machine-checked proof
  - theorem prover used: Isabelle/HOL

  - refinement proofs
    establish correspondence between high-level and low-level representations of a system
    - correspondence: Hoare logic properties hold for both levels



source: Klein, 2009

seL4

- formal verification

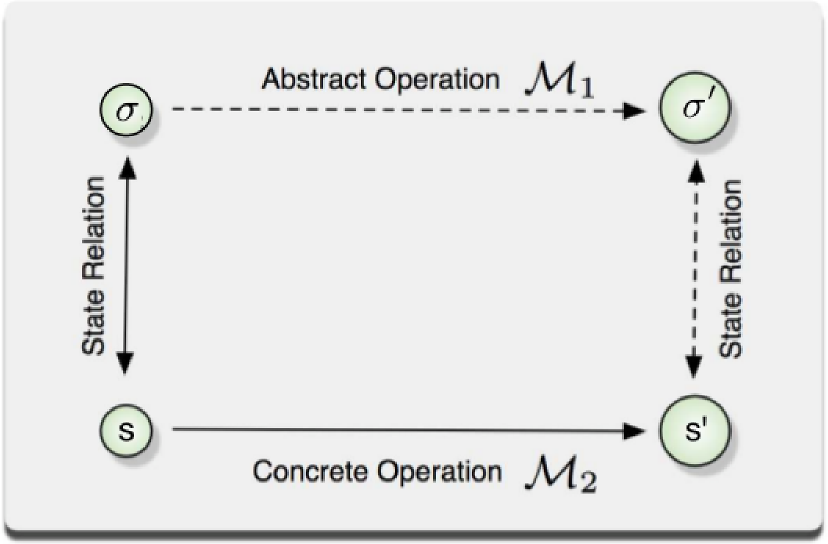
```
schedule ≡ do
  threads ← all_active_tcbs;
  thread ← select threads;
  switch_to_thread thread
od OR switch_to_idle_thread
```

```
schedule = do
  action <- getSchedulerAction
  case action of
    ChooseNewThread -> do
      chooseThread
      setSchedulerAction ResumeCurrentThread
      ...
chooseThread = do
    r <- findM chooseThread' (reverse [minBound .. maxBound])
    when (r == Nothing) $ switchToIdleThread
chooseThread' prio = do
     q <- getQueue prio
     liftM isJust $ findM chooseThread'' q
chooseThread'' thread = do
     runnable <- isRunnable thread
     if not runnable then do
             tcbSchedDequeue thread
             return False
     else do
             switchToThread thread
             return True
```

source: Klein, 2009

seL4

- the proof of functional correctness
  - reduction of refinement to forward simulation
  - for each transition in M_2: s → s' show that there exists a corresponding transition on the abstract side M_1: σ → σ'.
  - find a relation R that holds for each possible transition between the states s and σ, and s' and one in σ'.



source: Klein, 2009

seL4

- kernel design and implementation
  - reduced use of global variables
    - simplifies proof
  - memory management in user level
    - separate proof
  - concurrency
    - single processor
    - no exceptions
    - no yielding
    - interrupts disabled; poll-rollback-restart model

seL4

- implementation and verification effort
  - Haskell prototype: 5’700 LOC, 2 person years
  - C implementation: 8’700 LOC, 2 person months
  - proof: 200’000 LOP, > 20 person years

CSAP
Computer Systems and Platforms Laboratory

Okay, So What Now?

- There is still a lot of work to do
  - verified boot code, assembly code, and compiler
  - support for true concurrency (multiple cores)
  - support for H/W accelerators

References

- Alkassar, E., Hillebrand, M.A., Paul, W.J., and Petrova, E. “Automated Verification of a Small Hypervisor”, Lecture Notes in Computer Science 6217, pp. 40-54, 2010.
- Bevier, W.R. “Kit: A study in operating system verification”, IEEE Transactions on Software Engineering 15(11), pp. 1382-1396, 1989.
- Hillebrand, M.A., and Paul, W.J. “On the architecture of System Verification Environments”, Lecture Notes in Computer Science 4899, pp. 153-168, 2008.
- Klein, G. “Operating System Verification – An Overview”, Sadhana, Springer 34(1), pp. 27-69, 2009.
- Klein, G., Andronick, J., Elphinstone, K., Heiser, G., et al. “seL4: Formal Verification of an Operating-System Kernel”, Communications of the ACM 53(6), pp. 107-115, 2010.
- Neumann, P.G., and Feiertag, R.J. “PSOS revisited”, Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC’03), 2003.
- Seshadri, A., Luk, M., Qu. N., and Perrig, A. “SecVisor: A Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity OSes”, Proceedings of the biennial ACM Symposium on Operating Systems Principles (SOSP’07), 2007.
- Walker, B.J., Kemmerer, R.A., and Popek, G.J. “Specification and verification of the UCLA Unix security kernel”, Communications of the ACM 23(2), pp. 118-131, 1980.