seL4

Formal Verification of an OS Kernel

Gerwin Klein
June Andronick
Dhammika Elkaduwe
Michael Norrish

Kevin Elphinstone
David Cock
Kai Engelhardt
Thomas Sewell
Simon Winwood

Gernot Heiser
Philip Derrin
Rafal Kolanski
Harvey Tuch
1 microkernel
8,700 lines of C
0 bugs\(^*\)
\[\text{qed}\]

\(^*\text{conditions apply}\)
An exception 06 has occurred at 0028:C11B3ADC in VxD DiskTSD(03) + 00001660. This was called from 0028:C11B40C8 in VxD voltrack(04) + 00000000. It may be possible to continue normally.

* Press any key to attempt to continue.
* Press CTRL+ALT+RESET to restart your computer. You will lose any unsaved information in all applications.

Press any key to continue
The Problem
Small kernels

Small trustworthy foundation

- hypervisor, microkernel, nano-kernel, virtual machine, separation kernel, exokernel ...
- High assurance components in presence of other components

seL4 API:
- IPC
- Threads
- VM
- IRQ
- Capabilities

Untrusted

Trusted

Legacy Apps

Sensitive App

Linux Server

Trusted Service

Hardware
Small Kernels

Small trustworthy foundation

- hypervisor, microkernel, nano-kernel, virtual machine, separation kernel, exokernel ...
- High assurance components in presence of other components

seL4 API:
- IPC
- Threads
- VM
- IRQ
- Capabilities

Untrusted
- Legacy Apps
- Linux Server

Trusted
- Sensitive App
- Trusted Service

seL4

Hardware
The Proof
The Proof
Functional Correctness

Proof

Specification

Code
Functional Correctness

What

Proof

Specification

definition
  schedule :: unit s_monad where
  schedule ≡ do
    threads ← allActiveTCBs;
    thread ← select threads;
    switch_to_thread thread
  od
  OR switch_to_idle_thread
22 Threads and TCBs

theory Tcb_A
imports CSpace_A ArchVSpace_A Schedule_A Ipc_decls_A
begin

constdefs set_thread_state :: obj_ref \Rightarrow thread_state \Rightarrow unit s_monad
set_thread_state ref ts ≡ do
tcb ← assert_opt_get t get_tcb ref;
set_object ref yTCB ytcb (tcb_state := ts) z
od

defs suspend_def:
suspend lazy thread ≡ do
ipc_cancel thread;
set_thread_state thread Inactive
od

constdefs restart :: obj_ref \Rightarrow unit s_monad
restart thread ≡ do
state ← get_thread_state thread;
when y ¬ runnable state t do
ipc_cancel thread;
end

NICTA Confidential

definition
schedule :: unit s_monad where
schedule ≡ do
threads ← allActiveTCBs;
thread ← select threads;
switch_to_thread thread
od
OR switch_to_idle_thread

end
*conditions apply
*conditions apply
*conditions apply

- Specification
- Proof
- Code
- Assumptions
- Expectation
*conditions apply

Assume correct:
- compiler + linker (wrt. C op-sem)
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)
Implications

Execution always defined:
- no null pointer de-reference
- no buffer overflows
- no code injection
- no memory leaks/out of kernel memory
- no div by zero, no undefined shift
- no undefined execution
- no infinite loops/recursion

Not implied:
- “secure” (define secure)
- zero bugs from expectation to physical world
- covert channel analysis
Proof Architecture

- Specification
- Design
- C Code
Proof Architecture

Specification

Design

C Code
Proof Architecture

- Access Control Spec
- Specification
- Design
- C Code
- Confinement
definition

schedule :: unit s_monad where
schedule ≡ do
  threads ← allActiveTCBs;
  thread ← select threads;
  switch_to_thread thread
od
OR switch_to_idle_thread
Proof Architecture

- Access Control Spec
- Confinement
- Specification
- Design
- C Code

```
schedule :: Kernel ()
schedule = do
  action <- getSchedulerAction
  case action of
    ResumeCurrentThread -> return ()
    ChooseNewThread -> do
      chooseThread
      setSchedulerAction ResumeCurrentThread
      SwitchToThread t -> do
        switchToThread t
        setSchedulerAction ResumeCurrentThread

chooseThread :: Kernel ()
chooseThread = do
  r <- findM chooseThread' (reverse [minBound .. maxBound])
  when (r == Nothing) $ switchToIdleThread
  where
```
Proof Architecture

Access Control Spec ➔ Confinement

Specification

Design

C Code

```c
void schedule(void) {
    switch ((word_t)ksSchedulerAction) {
        case (word_t)SchedulerAction_ResumeCurrentThread:
            break;

        case (word_t)SchedulerAction_ChOOSE_NEW_THREAD:
            chooseThread();
            ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
            break;

        default: /* SwitchToThread */
            switchToThread(ksSchedulerAction);
            ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
            break;
    }
}

void chooseThread(void) {
    prio_t prio;
    tcb_t *thread, *next;
```
System Model

States:
User, Kernel, Idle

Events:
Syscall, Exception, IRQ, VM Fault
States:
User, Kernel, Idle

Events:
Syscall, Exception, IRQ, VM Fault
Kernel Design for Verification
Kernel Design for Verification
Two Teams

Formal Methods Practitioners

Kernel Developers
Two Teams

Formal Methods Practitioners

Kernel Developers

The Power of Abstraction

(Liskov 09)

Exterminate All OS Abstractions!

(Engler 95)
Iterative Design and Formalisation

- Whiteboard
- Formal Specification
- Formal Design
- Haskell Prototype
- C Code
Iterative Design and Formalisation

- Whiteboard
- Formal Design
- Formal Specification
- Haskell Prototype
- C Code
Iterative Design and Formalisation

- Whiteboard
- Haskell Prototype
- Formal Design
- Formal Specification
- C Code
Iterative Design and Formalisation

- Whiteboard
- Haskell Prototype
- Formal Design
- Formal Specification
- C Code

© NICTA 2009
Reducing Complexity

Hardware
- drivers outside kernel

Concurrency
- event based kernel
- limit preemption

Code
- derive from functional representation

```c
void schedule(void) {
    switch ((word_t)ksSchedulerAction) {
        case (word_t)SchedulerAction_ResumeCurrentThread:
            break;
        case (word_t)SchedulerAction_GenerateNewThread:
            chooseThread();
            SchedulerAction = SchedulerAction_ResumeCurrentThread;
            break;
    } /* SwitchToThread */
    switchToThread(ksSchedulerAction);
    SchedulerAction = SchedulerAction_ResumeCurrentThread;
    break;
}
```
C subset

Everything from C standard

- pointers, casts, pointer arithmetic
- data types
- structs, padding
- pointers into structs
- precise finite integer arithmetic

- goto, switch fall-through
- reference to local variable
- side-effects in expressions
- function pointers (restricted)
- unions

• including:
  - pointers, casts, pointer arithmetic
  - data types
  - structs, padding
  - pointers into structs
  - precise finite integer arithmetic

• minus:

• plus compiler assumptions on:
  - data layout, encoding, endianess
Did you find any Bugs?

Bugs found

during testing: 16

during verification:
• in C: 160
• in design: ~150
• in spec: ~150

460 bugs

Effort

- Haskell design: 2 py
- First C impl.: 2 weeks
- Debugging/Testing: 2 months
- Kernel verification: 12 py
- Formal frameworks: 10 py
- Total: 25 py

Cost

- Common Criteria EAL6: $87M
- L4.verified: $6M

© NICTA 2009
Did you find any Bugs?

Bugs found

- **during testing:** 16
- **during verification:**
  - in C: 160
  - in design: ~150
  - in spec: ~150

460 bugs

Effort

- Haskell design: 2 py
- First C impl.: 2 weeks
- Debugging/Testing: 2 months
- Kernel verification: 12 py
- Formal frameworks: 10 py
- Total: 25 py

Cost

- Common Criteria EAL6: $87M
- L4.verified: $6M
Summary

Formal proof all the way from spec to C.

- 200kloc handwritten, machine-checked proof
- ~460 bugs (160 in C)
- Verification on code, design, and spec
- Hard in the proof $\rightarrow$ Hard in the implementation

Formal Code Verification up to 10kloc:

- It works.
- It’s feasible. (It’s fun, too.
- It’s cheaper. And we’re hiring..)
Thank You
Thank You