

Towards a Practical, Verified Kernel

Kevin Elphinstone and Gerwin Klein, *National ICT Australia and the University of New South Wales*
 Philip Derrin, *National ICT Australia*
 Timothy Roscoe, *ETH Zürich*
 Gernot Heiser, *National ICT Australia, the University of New South Wales, and Open Kernel Labs*

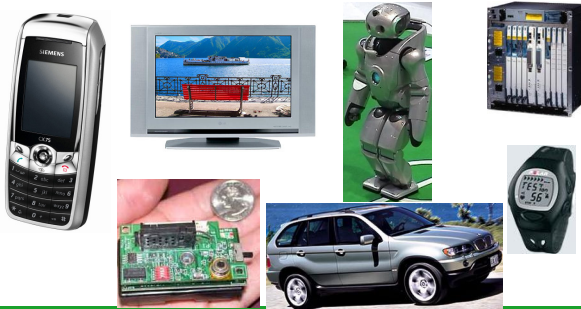


And the rest...

Dharmika Elkaduwe
 David Cock
 Thomas Sewell
 Simon Winwood
 Harvey Tuch
 Michael Norrish
 Jeremy Dawson
 Rafal Kolanski
 Jia Meng

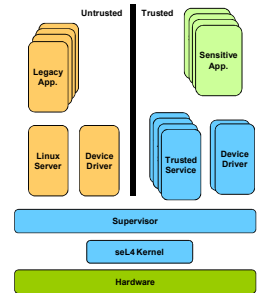
Feature-Rich Embedded Systems

Are they trustworthy?



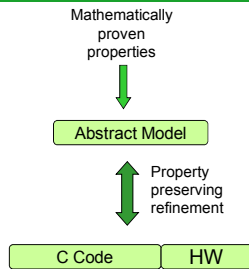
Small Kernels

- Smaller, more trustworthy foundation
 - Hypervisor, microkernel, nano-kernel, virtual machine monitor, isolation kernel, partitioning kernel, exokernel.....
 - Fault isolation, fault identification, IP protection, modularity.....
 - High assurance components in presence of other components
- Only as trustworthy as the foundation
 - Code reviews?
 - Testing?
 - Static analysis?
 - Sound and unsound



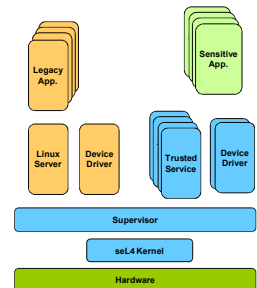
Our Goal – A Formally Verified Kernel

- A formally specified API (abstract model)
 - Proven properties such as spatial partitioning.
- High-performance implementation in C
 - A formal refinement between model and *implementation*
- Guaranteed correct **implementation!!!!**

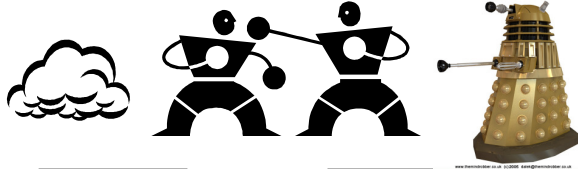


Success creates problems....

- Verified kernel is rigid
 - Changes invalidate proofs
- Kernel
 - "Mechanism not policy"
 - Flexible distribution of authority
 - Capabilities (KeyKOS/EROS)
 - Address spaces
 - Memory objects implemented at user level (L4/Xen/Nemesis)
 - Partitioning to hold within the kernel
 - No implicit memory allocation in kernel (Cambridge CAP)
 - Event-based/Atomic (Fluke)



Kernel Developers Versus Formal Methods Practitioners



Abstract Model C Code HW

Bridging the gap

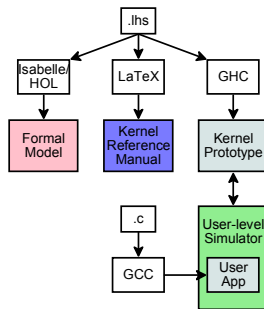


Modelling?

- Well defined semantics
- Readily formalisable
- Exposed implementation details
- Programming language

Our Prototyping Approach

- Model the kernel in detail
- Literate Haskell to model
 - Pure functional programming language
 - Embedded documentation
 - Close to Isabelle/HOL
- Executable
 - Supports running user-level code

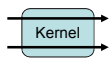


Kernel Modelling in Haskell

Kernel is event-based (single stack) mostly atomic API.

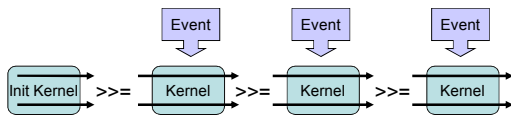
- Kernels are big state machines with events as input
 - Imperative
 - Rely on side-effects all the time
 - P(s), make_runnable(tcb)
- Kernels manipulate the low-level machine
 - Interrupts, TLBs, caches
- Preemption required
 - Kernels can't always perform operations to completion

Kernel Code in a State Monad



- State monads are *units* of computation which consume and produce state
 - State transformers
- Kernel monad encapsulates a state transformer of the kernel and machine

- Monads can be bound together using the *bind* operator
 - Sequencing the computation
 - Connects the plumbing to pass the state along

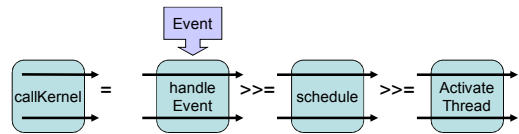


Kernel Code in a Monad

```

type Kernel = StateT KernelState MachineMonad
callKernel :: Event -> Kernel ()
callKernel ev =
  handleEvent ev >>= (\x -> schedule >>=
    (\y -> activateThread))

```

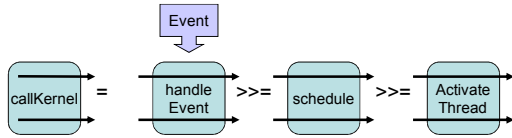


Kernel Code in a Monad

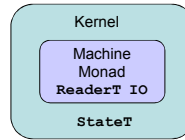
```

type Kernel = StateT KernelState MachineMonad
callKernel :: Event -> Kernel ()
callKernel ev = do
  handleEvent ev
  schedule
  activateThread
    
```

Imperative in "style"
Lowers barrier to entry for kernel developers



Kernel Monad



- Machine monad contains state to interface to simulator
- Kernel contains the state of physical memory

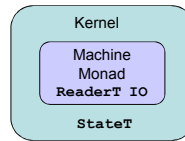
Machine Monad – Lowest Level of Model

```

• getMemoryTop :: MachineMonad (PPtr ())
• getDeviceRegions :: MachineMonad [(PPtr (), Int)]
• loadWord :: PPtr Word -> MachineMonad Word
• storeWord :: PPtr Word -> Word -> MachineMonad ()
• insertMapping :: PPtr Word -> VPtr -> Int -> Bool -> MachineMonad ()
• flushCaches :: MachineMonad ()
• getActiveIRQ :: MachineMonad (Maybe IRQ)
• maskInterrupt :: Bool -> IRQ -> MachineMonad ()
• ackInterrupt :: IRQ -> MachineMonad ()
• waitForInterrupt :: MachineMonad IRQ
• configureTimer :: MachineMonad IRQ
• resetTimer :: MachineMonad ()
    
```

- Foreign Function Interface (FFI)
- Approximate machine-level C functions
- Close to "real" as possible
 - Forces us to manage "hardware"

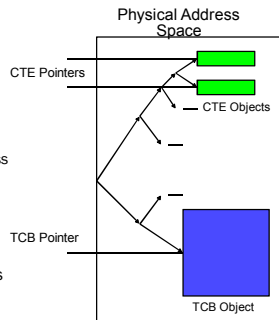
KernelState Monad



- Statically allocated global kernel data
 - Current thread
 - Scheduler queues
- Physical Memory

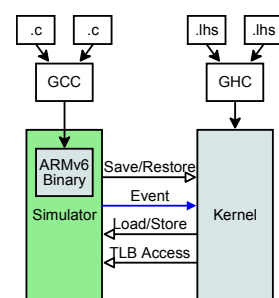
KernelState Monad

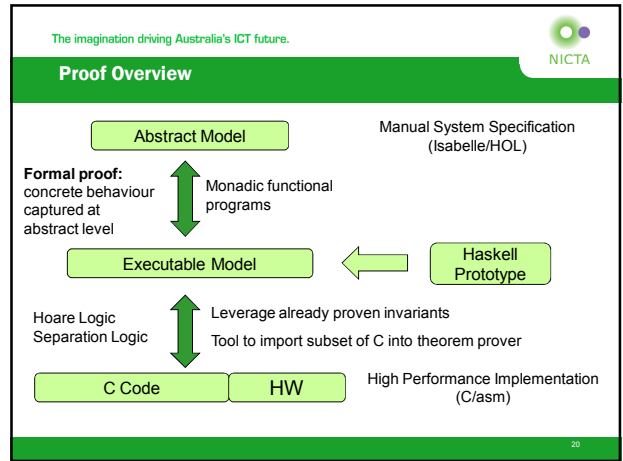
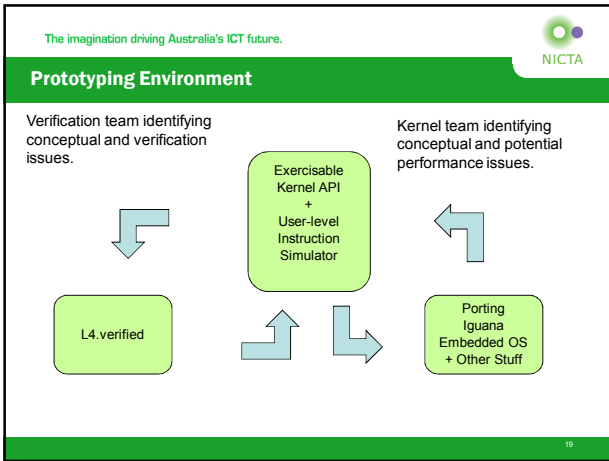
- Physical memory model
 - Contents of dynamically-allocated memory
 - Typed data used by the kernel
 - Thread control blocks
 - Capability and page tables
 - ...
 - Indexed by physical memory address
- Forces us to model memory management (30% of kernel)
- Reduces the gap to C
 - Pointers, not Haskell's heap
- Still provides strongly typed pointers



User-level Simulation

- User-level CPU simulator
 - M5 Alpha simulator
 - Locally-developed ARMv6 simulator
 - QEMU
- Executes compiled user-level binaries
- Sends events to the Haskell kernel





- The imagination driving Australia's ICT future.
- NICTA
- ### The Proof - Summary
- Statistics:
 - 3.5kloc abstract, 7kloc concrete spec (about 3k Haskell)
 - 53kloc proof so far (estm. 60kloc final, about 10kloc/month)
 - 37 patches to Haskell kernel, 110 to abstract spec
 - Proven lots of invariants:
 - well typed references, aligned objects
 - thread states and port queues
 - well formed current thread, scheduler queues
 - Proof of termination
 - Expect to prove "does not fail"
 - Pen-paper proof of spatial partitioning
 - Identified set of invariants required of supervisor
- 21

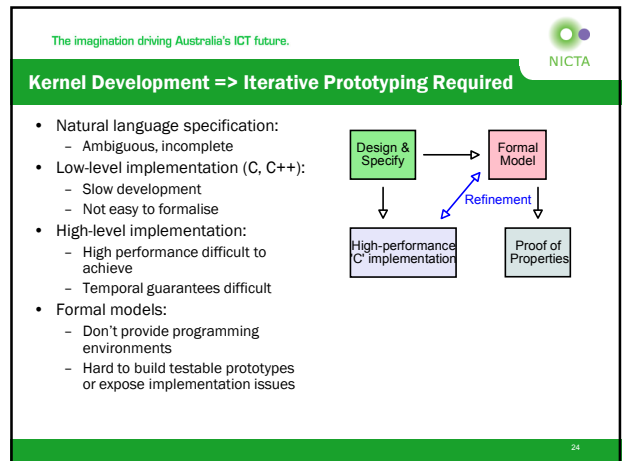
- The imagination driving Australia's ICT future.
- NICTA
- ### Conclusions
- Significant gap between kernel development and formal verification
 - Haskell (Functional Programming) middle ground
 - Programming language
 - Model low-level details
 - Readily formalisable
 - Make proving easier - explicitly manage side-effects
- Overall Status
- Very close to a verified detailed executable kernel model
 - Immature 'C' version of the design
- 22

The imagination driving Australia's ICT future.

NICTA

Questions

23



Kernel Code in a Monad

```
type Kernel = StateT KernelState MachineMonad
callKernel :: Event -> Kernel ()
callKernel ev = do
  runErrorT $ handleEvent ev
  `catchError` \irq -> withoutPreemption $
    handleInterrupt irq
  schedule
  activateThread
```

