An exception 06 has occurred at 0028:C11B3ADC in \xD DiskTSD(03) + 00001660. This was called from 0028:C11B40C8 in \xD 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
Trust Without Trustworthiness
What’s Next?
Our Vision: Trustworthy Systems

- We will change industry’s approach to the design and implementation
- of embedded software, resulting in systems which are **trustworthy**.

*Trustworthy* means *truly dependable* in that there are *hard guarantees* about the *security, safety* or *reliability* of the software.
Approach: Microkernel Technology

- Protect critical components by sandboxing complex components
- Provide tightly-controlled communication channels
- *Trustworthy microkernel* enforces security/safety policies
- Microkernel becomes core of trusted computing base
  - System trustworthiness only as good as microkernel
Trustworthy Systems Agenda

1. Ensure microkernel trustworthiness (seL4)
   - Proof of functional correctness
   - Proof of safety/security properties

2. Lift microkernel guarantees to whole system
   - Use safety/security guarantees
   - Prove correctness of balance of trusted computing base
   - Prove safety/security of complete system

Ingredients:
   - Functional correctness
   - Isolation / non-interference / information flow
   - Timeliness / worst-case latency guarantees
   - Energy management
seL4: Designing and Formalising

- Design & Specify
- Formal Model
- Proof
- High-Performance C implementation
- Safety Theorem
Two Mentalities

Formal Methods Practitioners vs Kernel Developers
Standard Kernel Design

Kernel Hacker View

- Design & Specify
- High-Performance C implementation
- Prototype on Real Hardware
- White-board
- Step 2
- Formal Model
- Safety Theorem
- Proof
- Proof

From imagination to impact
Formal Design

Formal Methods

View

Design in Theorem Prover

Design & Specify

Formal Model

Proof

Proof

High-Performance C implementation

Safety Theorem

Step 2
Iterative Design and Formalisation

- **Design & Specify**
- **Formal Model**
- **Safety Theorem**

- **Haskell Prototype**
  - inspired by existing code
  - prototype kernel executes native binaries on simulator
  - exposes usability issues early
  - tight formal design integration

- **High-Performance C implementation**

NICTA Copyright 2010
Kernel Design for Verification

• Main objective: minimise complexity
  – global invariants must be proven for each state change
  – must prove pre- and post-conditions for statements/blocks
  – effort determined by complexity of conditions and state change

• … without sacrificing performance

• Affects design in many ways
  – global variables, side effects
  – kernel memory management
  – concurrency and non-determinism
  – I/O
Global Variables

- Not a difficulty per se, but potential source of complexity
- Eg: scheduler queue as doubly-linked list
  - show that
    - all pointers are to valid nodes
    - front- and back-pointers are consistent
    - nodes point to TCBs
- Requires proof that any pointer operation maintains invariants
- Challenge is temporary violation
  - eg adding a node
  - requires ensuring atomicity
Kernel Memory

- seL4 kernel memory management model pushes policy to userland
  - aids verification
  - need to ensure strict hierarchy
  - capability derivation tree
- Challenge is re-use
  - most difficult part of verification!
  - use derivation tree to detect all references
  - global data structure that requires invariants in all parts of the system
Concurrency

- Proofs about concurrent programs are inherently hard!
- seL4 strictly limits concurrency to the bare minimum
  - single processor
  - user-level device drivers
  - non-preemptible, event-based
    - single kernel stack
  - interrupt points to limit real-time latencies
    - poll interrupt status
    - insert new kernel event (ahead of user)
    - return to user boundary and re-enter kernel
    - allows maintaining all invariants
Concurrency

• Preempting object destruction
• Keep one cap as zombie during object cleanup
  – only retained to reference partially cleaned-up object
  – stores state of cleanup, maintaining invariants
  – attempt by preemptor to remove zombie can just execute

• Exceptions in kernel
• Prevent memory exceptions
  – ensure kernel page tables are complete
  – map into every address space
• Disallow other exceptions
  – verification is its own friend
I/O

- Mostly a non-issue
  - user-level drivers
    - IOMMU support for DMA security
  - non-preemptible kernel
- Exception is timer tick
  - essentially a source of interrupts
  - handled in-kernel as separate event
  - no real complication
Lessons for Kernel Design

- Need to reduce complexity forced simple and clean design
  - beneficial even with traditional validation
  - does not necessarily impact performance
- Some design decision beneficial for other reasons too
  - single kernel stack for memory footprint
  - interrupt handling by polling has performance advantages
seL4 Status

• seL4 operational on ARM11 and x86
  – runs Linux etc...
• Performance in line with other L4 kernels
  – 300 cycle IPC (ARM11 with IPC fastpath in C)
  – Exception handling slow (no fastpath yet)
    • Virtualization performance sub-optimal
• Multiprocessor version in progress
• Userland being developed
  – Eg memory-management
Proof of Functional Correctness

- Access Control Model (300)
  - on-going
- Abstract Model (4,900)
  - 117,000 lop
- Executable Model (13,000)
  - 50,000 lop
- C Code (8,700)
- Confinement (10)
  - 3,000 lop

- Manual System Specification (Isabelle/HOL)

- Haskell Prototype (5,700)

- High Performance Implementation (C/asm)
  - Hardware model

Formal proof: concrete behaviour captured at abstract level

NICTA Copyright 2010
From imagination to impact

Proof of Functional Correctness

- Access Control Model (300)
- Abstract Model (4,900)
- Executable Model (13,000)
- C Code (8,700)

Confinement (10)

- Haskell Prototype (5,700)

Formal proof:
- concrete behaviour captured at abstract level.

Manual System Specification (Isabelle/HOL)

High Performance Implementation (C/asm)

Hardware model

3,000 lop
117,000 lop
50,000 lop
(92% done)

NICTA Copyright 2010
Formal Verification Summary

Kinds of properties proved

• Behaviour of C code is fully captured by abstract model
• Behaviour of C code is fully captured by executable model
  • Can prove many interesting properties on higher-level models
• Kernel never fails, behaviour is always well-defined
  • Assertions never fail
  • Will never de-reference null pointer
  • Cannot be subverted by misformed input
• All syscalls terminate, reclaiming memory is safe, ...
• Well typed references, aligned objects, kernel always mapped...
• Access control is decidable

Effort:
• Average 6 people over 5.5 years
### Verification vs Certification

#### Common Criteria: Military-Strength Security

<table>
<thead>
<tr>
<th>Evaluation Level</th>
<th>Requirements</th>
<th>Functional Specification</th>
<th>Top Down Design</th>
<th>Implementation</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAL1</td>
<td>Informal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EAL2</td>
<td>Informal</td>
<td></td>
<td>Informal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EAL3</td>
<td>Informal</td>
<td></td>
<td>Informal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EAL4</td>
<td>Informal</td>
<td></td>
<td>Informal</td>
<td>Informal</td>
<td></td>
</tr>
<tr>
<td>EAL5</td>
<td>Semi-formal</td>
<td>Semi-formal</td>
<td>Semi-formal</td>
<td>Informal</td>
<td></td>
</tr>
<tr>
<td>EAL6</td>
<td>Formal</td>
<td>Semi-formal</td>
<td>Semi-formal</td>
<td>Informal</td>
<td>1K/LoC</td>
</tr>
<tr>
<td>EAL7</td>
<td>Formal</td>
<td>Formal</td>
<td>Formal</td>
<td>Informal</td>
<td></td>
</tr>
<tr>
<td>seL4</td>
<td>Formal</td>
<td>Formal</td>
<td>Formal</td>
<td>Formal</td>
<td>0.6K/LoC</td>
</tr>
</tbody>
</table>
Phase Two: Full-System Guarantees

- Achieved: Verification of microkernel (8,700 LOC)

- Next step: Guarantees for real-world systems (1,000,000 LOC)
Overview of Approach

- Build system with minimal TCB
- Formalize and prove security properties about architecture
- Prove correctness of trusted components
- Prove correctness of setup
- Prove temporal properties (isolation, WCET, …)
- Maintain performance
Proof of Concept: Secure Access Controller
Provider A & B should not be able to leak info between each other even if they actively cooperate.
Linux-based Router minimal device access

Solution Overview

Windows
Network A
Router (Linux)
Network Interface A
Terminal Network Interface
Terminal
User

Linux
Network B
Web Server (Linux)
Network Interface B
Control Interface
Control Network

Not Connected

NICTA Copyright 2010
Specifying Security Architecture
Component Side-Channel Mitigation

- Component response time variability exposes secrets.
- Add a response time control policy.
- Real-time scheduling gives a mechanism.
- seL4 endpoints give a framework.
Trusted Synthesized Drivers

• Correct driver synthesis
  – given model of driver interface, basic behaviour, and hardware
  – driver is automatically generated
  – performance as good as hand-knitted

• Challenge: device spec

• Vision:
  – automatically extract hardware model from HDL description
  – potential impact beyond our immediate agenda

Formal OS interface spec

HDL

Formal device spec

driver.c
Kernel Worst-Case Execution Time

• seL4 is small as an OS kernel (9 kLOC)
  – … but large as an object for WCET analysis
  – … and full of performance optimisations

• However, we know a lot about it (in a very formal way!)
  – Plenty of invariants proved during verification
  – E.g. loop iteration counts, non-interference

• Can make use of this for WCET analysis
  – Collaboration with WCET experts at NUS (Abhik Roychoudhury)
Power Analysis
Energy Management

Koala: accurate model of CPU and memory energy use

• Driven by performance counters
• Model predicts energy consumption and relative execution speed
  – at present frequency
  – at different frequencies
• Accurately predicts energy- and performance response to DVFS
  – within a few %
• Can use this for informed energy-management decisions
• Developed OS-level policy framework
Accuracy of Approach

![Graph showing the relationship between requested minimum performance and actual performance for different benchmarks. The graph compares memory-bound and CPU-bound performance.](image_url)

**Memory-bound**

**CPU-bound**

© NICTA 2010

From imagination to impact 37
How About Peripherals?

Exploring approach similar to DVFS work:

• Assume we can predict device energy from known workload
• General energy model of device parameterised by I/O events
  – Observable by device driver
  – Device-class specific, device-independent
• Model gauged through (random) characterisation work load
  – Varying data rates and packet sizes
  – Using linear regression on model parameters
• Evaluated with evaluation benchmark suite
Example: Network Devices

Energy model:

\[ P = P_0 + E_1 b_r + E_2 b_w + E_3 p_r + E_4 p_w + P_i \]

- \( b_r, b_w \): data (byte) read, write rate
- \( p_r, p_w \): packet read, write rate
- \( i \): 1=active, 0=idle (accounts for idle state)
AX88772 USB Ethernet

Predicted Energy (mJ)

Measured Energy (mJ)
Several USB Ethernet Devices

![Graph showing measured vs predicted energy for Quokka USB Ethernet devices. The x-axis represents measured energy (mJ), and the y-axis represents predicted energy (mJ). The graph includes data points for different configurations labeled as pld, plf, asf, alf, and xlf.](image-url)
Device Power

- Device class family driver collects energy stats
- Initial results look very promising
  - USB-connected network devices
    - Ethernet, WiFi
  - Also did (earlier) Flash storage devices
    - Less mature approach, re-doing now
- Work in progress, planning to look at other device classes
  - Disks (incl head movement)
  - Audio (speakers)
Multicore Energy Management

• Additional degree of freedom
  – DVFS + sleep states + core shutdown
  – Hypervisor supports transparent, temporary consolidation of cores
    • Unneeded cores turned off to reduce power

• Different tradeoffs
  – Performance vs power close to linear

• Important to manage cores globally
  – In average more cores off than with per-guest management
    • Can use deeper sleep state
    • Less overall energy use
UNSW OS Students Pipeline

Smart students → UNSW OS + AdvOS + ToR + Thesis → OS-trained graduates

OS-trained graduates → Universities Google VMware Apple...

World-class PhDs → Experienced PhD students

NICTA ERTOS → Open Kernel Labs