LOCAL OPERATING SYSTEMS R&D AND OPPORTUNITIES FOR STUDENTS

COMP9242
2006/S2 Week 14

OVERVIEW

Cool systems stuff happening at:

- UNSW
  - Gelato@UNSW
  - Linux scalability and performance on Itanium
- National ICT Australia (NICTA)
  - Embedded, Real-Time and Operating Systems (ERTOS) Program
  - world-class research agenda on embedded operating systems
- Open Kernel Labs, Inc
  - brand-new ERTOS spinout with a global business
  - microkernels for millions of people
- Opportunities
  - summer projects
  - theses
  - employment

NATIONAL ICT AUSTRALIA (NICTA)

- National Centre of Excellence in Information and Communications Technology (ICT)
- Created in 2003 by Australian Government
  - members: UNSW, ANU, NSW and ACT governments
  - partners: USyd, UMelb, UQ, QUT, Griffith, QLD+Vic governments
  - locations: Sydney (Kensington, ATP), Canberra, Melbourne, Brisbane
- Aim: change the Australian ICT landscape
  - conduct world-class research
  - improve quality of Australian ICT PhDs
  - commercialise research outputs
  - achieve real impact
  - become one of the top-ten ICT research institutions in the world

NICTA STRUCTURE

- Presently ≈ 300 researchers, 250 PhD students
- Most researchers belong to Research Programs
  - aligned with discipline areas (≈ 5–10 researchers)
  - ERTOS is one of them (currently largest)
  - medium- to long-term vision
- Projects focused on specific outcomes
  - collaborative or client-focused
  - 1–20 people, 1–5 years
- International Science Advisory Group
  - J Vuillemin (VP, INRIA), D Rombach (Head, Fraunhofer IESE), R Newton (Dean UCB), R Brooks (Head MIT CSAIL), S Feldman (VP, IBM)
- International Business Advisory Group
  - D Zitzner, (ret exec VP, HP), N Murthy (Chairman, Infosys), C Mudge (Dir, Macq Innov), B Bishop (V. Chairman, SGI), H Killen (MP Hemisphere Capital)
**ERTOS Agenda**

**ERTOS Vision**
To make highly reliable, safe and secure embedded systems a widely-deployed reality.

**ERTOS Mission**
To establish ERTOS-developed embedded operating systems as de-facto industry standards.

---

**Embedded System**

Computer system that is part of a larger system.

---

**General-Purpose vs. Embedded**

**Traditional view:**
- **General-Purpose System:**
  - Applications
  - File System
  - Virtual Memory
  - Low-level I/O
  - Network Stack
  - Scheduling
  - Device Drivers
  - Interrupt Handler
- **Embedded System:**
  - Minimal
  - No OS at all or small “real-time executive”
  - No protection
**Security Challenges**

- Growing functionality
  - Increasing software complexity
  - Millions LOC on phone handsets
  - Gigabytes of code in cars
  - Increased number of faults
  - Increased likelihood of security faults

- Wireless connectivity
  - Subject to attacks from outside (crackers)

- Downloaded contents (entertainment)
  - Subject to attacks from inside (viruses, worms)

- Increasing dependence on embedded systems
  - Increased exposure to embedded-systems security weaknesses

**Embedded Systems Software**

Present approaches 1: Real-time executives

- Small, simple operating system
  - Optimised for fast real-time response
  - Suitable for systems with very limited functionality

- No internal protection
  - Every small bug/failure is fatal
  - No defence against viruses, limited defence against crackers

**Embedded Systems Software**

Present Approaches 2: Linux, Windows Embedded, ...

- Scaled-down version of desktop operating system
  - Operating system protected from application misbehaviour
  - Excessive code base for small embedded system
  - Too much code on which security of system is dependent

- Dubious or non-existent real-time capabilities
  - Unsuitable for hard real-time systems

**Embedded Systems Requirements**:

Reliability, trustworthiness, security:

- Achieved by:
  - Exhaustive testing?
  - Systematic code inspection?
  - Formal methods?
  - Scale poorly (few 1000 loc)

- Requires minimal trusted computing base (TCB):
  - TCB: The part of system that must be relied on for the correct operation of the system

- Why minimal TCB?
  - Minimise exposure to bugs/faults
  - Minimise exposure to attacks (internal and external)
  - Support poorly-scaling verification methods
**Trusted Computing Base**

What does the TCB contain?

- Kernel
  - (part of system that executes in privileged mode)
  - everything running in privileged mode can bypass security
- Device drivers
  - DMA-capable devices can bypass protection
  - drivers can mount DoS attacks
- Services that control resources
  - resource owner can deny resource
  - resource owner can leak/corrupt data
- Everything on MPU-less processors
  - no memory-protection hardware ➔ no memory protection

---

**Minimising the Size of the TCB**

... means first of all:

- Use an MPU — microcontrollers are out!
- Minimise the size of the kernel!

Minimising kernel size:

- Reduce kernel to what is essential for supporting secure systems
- What does not require privileged mode must not be in the kernel
- This is the definition of a microkernel
- Minimal TCB required ➔ microkernel required!

---

**A Sample System**

- Sensitive part of system has small TCB
- Standard API supported by de-privileged Linux server ➔ full binary compatibility with native Linux
- Compromised legacy system cannot interfere with trusted part
WOMBAT PERFORMANCE: LMBENCH LAT CTX

![Graph showing latency vs number of processes for Vanilla Linux and Wombat/L4.]

WOMBAT PERFORMANCE: LMBENCH PIPE

![Graph showing bandwidth vs size for Vanilla Linux and Wombat/L4.]

WOMBAT PERFORMANCE: OTHER BENCHMARKS

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Linux</th>
<th>Wombat/L4</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>lmbench latencies, $[\mu s]$, smaller is better</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lat_fifo</td>
<td>510</td>
<td>49</td>
<td>10.4</td>
</tr>
<tr>
<td>lat_pipe</td>
<td>509</td>
<td>49</td>
<td>10.3</td>
</tr>
<tr>
<td>lat_unix</td>
<td>1015</td>
<td>77</td>
<td>13.3</td>
</tr>
<tr>
<td>lat_sem</td>
<td>199</td>
<td>14</td>
<td>14.6</td>
</tr>
</tbody>
</table>

AIM7 multitasking benchmark (jobs/min/task)

<table>
<thead>
<tr>
<th></th>
<th>Linux</th>
<th>Wombat/L4</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 task</td>
<td>45.15</td>
<td>43.62</td>
<td>0.97</td>
</tr>
<tr>
<td>2 tasks</td>
<td>23.35</td>
<td>22.62</td>
<td>0.97</td>
</tr>
<tr>
<td>3 tasks</td>
<td>15.79</td>
<td>15.30</td>
<td>0.97</td>
</tr>
</tbody>
</table>

SEL4: MICROKERNEL MECHANISMS FOR SECURE SYSTEMS

SEL4: Microkernel for secure embedded systems:
- Security requirements for embedded systems:
  - Integrity: protecting data from damage
  - Availability: ensuring system operation
  - Privacy: protecting sensitive data from loss
  - IP Protection: controlling propagation of valuable data
- Issue: Old L4 API unsuitable for highly-secure systems
  - Inefficient information flow control mechanisms
  - Present mechanisms double or triple IPC costs
  - Insufficient resource isolation (kernel memory pool)
  - Applications can force kernel to run out of memory
  - Present countermeasures are inflexible
- Note: Interim solutions for those issues presently in place
  - SEL4 working on clean and general model
  - Production kernel API adapts continuously (and gently)
SEL4: Microkernel Mechanisms for Secure Systems

Communications control:
- Targeting confinement, including covert storage channels
- Capability-based IPC authorisation
- Aim: control communication between arbitrary subsystems
- Session-based communications (no resume cap)
  - reply cap passed explicitly on each IPC, or
  - passed once on session establishment

Resource management:
- Aiming at complete control of kernel memory allocation
  - Verification requires static kernel implementation
  - Allocation policy dependent on application
- Dual systems (Linux + RT) have completing policy requirements
- Initially have cap to untyped memory (unused kernel memory)
- Provide model for synthesising new kernel objects (+ caps)
- Possible due to L4 simplicity, small number of objects:
  - TCBs
  - Physical frames for virtual memory
  - Synchronous endpoints (like ports with no resume caps)
  - Asynchronous notification objects
  - Capability nodes
  - Few more for interrupt controllers, page tables, synchronisation

Project status:
- Semi-formal API specification in literate Haskell
  - Automatic generation of API documentation from source
  - Draft available from http://ertos.sics.uu.se/research/seL4
- Paper proof of separation properties
  - Suitable for confinement, DRM
- Prototype implementation in Haskell, integrated with ISA simulator
  - Rapid prototyping: API changes implemented in hours/days
  - Can build and execute apps using standard build tools
  - Used for porting user-level software
- C implementation in progress
  - Prototype in Dec '06

L4 verifying: Formal verification of Kernel

- Leverage small size of kernel to prove correctness
- Part 1: Pilot project (Jan '04 – Mar '05)
  - Verified thin “slice” of API all the way to source code
  - Memory-management functions
  - > 10% of kernel code, > 20% of kernel complexity
  - 1.5 person years
  - Did almost complete formalisation of present L4 API
- Part 2: Main project (Apr '05 – Mar '08)
  - Refinement approach using the Isabelle/HOL theorem prover
  - Importing seL4 API specification (Haskell)
  - Importing C/assembler implementation to be proved
  - Result to be usable in existing deployments
  - No sacrificing of performance for verifiability
  - On track...
**POTOROO: COMPLETE TEMPORAL MODEL OF KERNEL**

- Prerequisite for complete real-time analysis of whole system
  - strict worst-case execution times (WCET)
  - probabilistic WCET
- Essential for trustworthy real-time systems
  - RT analysis of applications pointless without timing model of kernel
- Measurement-based approach augmented by static analysis
  - measure execution-time profiles of basic blocks
  - convolute into overall execution-time profile
  - static analysis to ensure worst case observed
  - static analysis to reduce pessimism

**CAMKES: COMPONENT ARCHITECTURE FOR MICROKERNEL-BASED EMBEDDED SYSTEMS**

- Aim: approach for highly-componentised embedded software
  - ✓ reduce software cost by enforcing modularity
  - ✓ deliver on fault isolation, hot upgrades, security enforcement, ...
- Ultimate goal:
  - Full system verification
  - kernel-enforced component boundaries
  - ✓ can verify components individually
  - ✓ model composition?
  - ✓ Distant future...
- Status: static prototype
- Working on dynamic system, performance, non-functional properties

**HIGH-PERFORMANCE USER-LEVEL DRIVERS**

- L4 device drivers are *always* outside the kernel (at user level)
  - Interrupts delivered to driver as IPC from kernel
- Potentially higher communication overhead
  - past experience with user-level drivers:
    - ≥50% performance degradation
- L4 IPC performance is very high
  - with well-designed driver interfaces can achieve good performance

**USER-LEVEL DEVICE DRIVERS ON LINUX**
**User-Level Device Driver Performance**

Gigabit Ethernet echo on 900MHz Itanium-2 with 66MHz 64-bit PCI

**User-Level Drivers: Ongoing Work**

- Complete driver framework and methodology
  - ease development of high-performance drivers
  - reduce driver complexity
  - drivers portable between systems (L4 and Linux)
- Integration with I/O system
  - Linux VFS layer integration
  - user-level network protocol stacks
  - componentised protocol stacks
- Driver encapsulation
  - use hardware mechanisms to limit DMA
  - use software mechanisms to limit trust in drivers
  - goal: untrusted device drivers
- Collaboration between NICTA and UNSW Gelato project
  - 1 PhD student

**Present State**

- Pistachio: Mature microkernel
  - 10,000 lines of code (shrinking)
  - highly efficient
- Iguana: Core OS services
  - naming, protection, memory...
  - device drivers
  - optional Linux server
- Multiple architectures
  - on ARM, MIPS, x86
- Commercially deployed
  - new base of Qualcomm CDMA chip firmware
  - other deployments in pipeline

**Open Kernel Labs (OKL)**

- Startup company for commercialising ERTOS technology
- Created Sep 2006 (still in setup process)
  - Steve Subar, CEO
  - startup veteran
  - Gernot Heiser, CTO
  - ca 15 engineers, growing 1-2 per month
    - probably largest group of top kernel hackers outside major multinationals
  - US HQ, Sydney-based engineering
- Projects with 5 large multinationals, several others in pipeline
  - mobile communication chipsets and phone handsets
  - multimedia
  - some huge stuff we can’t talk about
**Open Kernel Labs — A Unique Approach**

OKL-NICTA Joint Venture:
- OKL provides services
- NICTA/ERTOS does research
- Outcomes industrialised and commercialised by OKL

OKL/NICTA Ongoing Relationship:
- Students move into either OKL or ERTOS ➔ working on similar stuff
- ERTOS staff move into OKL with their projects ➔ efficient industrialisation/commercialisation
- OKL staff move back to ERTOS ➔ do PhD on research issues identified by OKL

**Student Opportunities**

Would you like to work on cool systems people actually use???
- Gelato — kernel work for supercomputers ➔ with significant research issues
- BLUEsat — L4 in space! ➔ but first it needs an OS!
- ERTOS research — trustworthy embedded systems ➔ ... will change the industry!
- Open Kernel Labs — microkernels in billions of devices ➔ hot startup building cool systems