Operating Systems Research at
UNSW and NICTA

and
Opportunities for Students
COMP9242 2004/S2 Week 14

National ICT Australia (NICTA)

- National Centre of Excellence in Information and Communications Technology (ICT)
- Created in October 2002 by Australian Government
- Members:
  - University of New South Wales (UNSW), Sydney
  - Australian National University (ANU), Canberra
  - NSW and ACT governments
- Locations:
  - Sydney (UNSW campus and ATP technology park)
  - Canberra
  - Melbourne (University of Melbourne Campus)
  - Brisbane (in progress)

NICTA’s Four Pillars

- Research
  - conduct world-class ICT research that makes an impact
  - ‘use-inspired fundamental research’
- Education
  - produce world-class PhD graduates
  - students enrolled at one of the member/partner universities
- Linkages
  - collaborate with top research institutions around the world
- Commercialisation
  - turn research into products
  - focus on
    - local small and medium-sized enterprises (SMEs)
    - multinational corporations (MNCs)

NICTA Structure

- Presently ~ 150 researchers, 150 PhD students
- Researchers belong to Research Programs
  - aligned with discipline areas (= 5–10 researchers)
  - lifetime 5–10 years
- Priority Challenges drive research
  - PC1: trusted wireless networks
  - PC2: from data to knowledge
- Projects focused on specific outcomes
  - collaborative or client-focused
  - lifetime 1–5 years
- International Science Advisory Group
  - Richard Newton (UCB), Shankar Sastry (UCB), Raj Reddy (CMU), Rodney Brooks (MIT), Jeff Ullman (Stanford), Gunnar Bjurel (SIKS), Gilles Kahn (INRIA), Ya-Qin. Zhang (Microsoft), ...

Embedded, Real-Time and Operating Systems (ERTOS) Program

- One of presently 16 Research Programs in NICTA
  - 8 FTE PhD-qualified researchers (1 vacant)
  - 11 FTE research engineers/research assistants (2 PhDs)
  - 20 PhD students (10 core OS topics)
- Competencies in
  - operating systems, microkernels
  - networking
  - real-time systems
  - reconfigurable computing
  - programming languages and compiler front-ends

Embedded Systems in Australia

- Australian embedded systems industry landscape
  - little industrial research
  - innovation concentrated in SMEs
  - little confidence in locally-developed technology
  - operating in niche markets
- Implications:
  - no scope for ASICs, use COTS hardware components
  - main focus is on software
  - reconfigurable hardware will be increasingly important
  - prototyping
  - small series
  - flexibility

NICTA Priority Challenge:
Trusted Wireless Networks

To enable greater confidence, freedom, and capability through improved efficiency, reliability, and security of all wireless environments.

- Strongly based on embedded systems technology
- Issues relevant to ERTOS:
  - efficiency
  - reliability
  - security
  - cost
ERTOS VISION

To develop methodologies, tools, components and systems that will deliver reliable, trustworthy and inexpensive embedded systems software

ERTOS research is to be driven by applications

- to identify common challenges
- to provide generic systems software

ERTOS COMPUTER SYSTEMS

Computer system that is part of a larger system

GENERAL-PURPOSE VS. EMBEDDED

TRADITIONAL VIEW:

<table>
<thead>
<tr>
<th>General-Purpose System</th>
<th>Embedded System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications</td>
<td>Device Drivers</td>
</tr>
<tr>
<td>File System</td>
<td>Application</td>
</tr>
<tr>
<td>Virtual Memory</td>
<td>Hardware</td>
</tr>
<tr>
<td>Low-Level OS</td>
<td>Hardware</td>
</tr>
<tr>
<td>Interrupts</td>
<td></td>
</tr>
<tr>
<td>Scheduling</td>
<td></td>
</tr>
<tr>
<td>Device Drivers</td>
<td></td>
</tr>
<tr>
<td>Interrupt Handler</td>
<td></td>
</tr>
<tr>
<td>Hardware</td>
<td></td>
</tr>
</tbody>
</table>

- Minimal
- No OS at all or small "real-time executive"
- No protection

SECURITY CHALLENGES

- Growing functionality
  - Increasing software complexity
  - Increased number of faults
  - Increased likelihood of security faults
- Wireless connectivity
  - Subject to attacks from outside (crackers)
  - 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

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
- Dubsicous 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
  - 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!

TRUSTED COMPUTING BASE

<table>
<thead>
<tr>
<th>System: traditional embedded code</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCB: all code</td>
</tr>
<tr>
<td>Linux/Windows 100,000's loc</td>
</tr>
<tr>
<td>Microkernel-based 10,000's loc</td>
</tr>
</tbody>
</table>

Small is beautiful:
- Small kernel ⇒ potentially small TCB
- Small TCB ⇒ more trustworthy TCB!

Challenge: Can we guarantee the trustworthiness of the TCB?

MICROKERNEL BENEFITS

- Fault isolation
- Other components protected from fault
- Hot swapping / hot upgrade
- Can replace servers on running system
- Software engineering techniques
- Componentised system
- Enforcement of security policies
- Kernel controls communication even between system services
- Resource management for system services
- OS servers are like normal user code
- Formal verification
- Hardware-enforced isolation reduces complexity

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

L4: NICTA L4-EMBEDDED API

SIMPLER/SMALLER IMPLEMENTATION
- L4Ka::Pistachio issues
  - Optimised for high-end systems (servers)
  - Large memory footprint (presently 100kB)
  - Should fit on smartcard
- Microkernel provided protection of all components from each other
- Making code hard to validate implementation (formally or informally)
- Makes it hard to establish worst-case latencies

Iguana: OS PERSONALITY FOR EMBEDDED SYSTEMS

- Remember, L4 is (almost) a “strict” microkernel:
  - Does not provide any services
  - Does not provide policies (or only very few)
  - Provides mechanisms
- L4 aspires to be a generic kernel, suitable for all kinds of systems
- Almost any system requires a set of core services:
  - Process management
  - Memory management
  - Security management
  - Based on some system-wide policies
- Iguana provides these (or at least more tools for providing them)
- Designed for use in embedded systems
- Designed to minimise trusted computing base

WHAT DOES IGUANA PROVIDE?

- Convenient way of using L4 primitives
- OO-style method invocations instead of explicit IPC calls
- IDL compiler for automatic generation of stubs
- Protection framework for access rights management
  - Capability-based, flexible
  - Able to model most standard security models
- Virtual memory management
  - Allocation, deallocation, sharing, ... 
  - Single-address-space view, supporting FASS on ARM
- Protection-domain (process) management
- Thread management
SAMPLE IGUANA SYSTEM

IGUANA: BASIC APPROACH

- Basic idea: single address space (SAS)
  - eases sharing of data
  - minimises copying
  - no problems with pointers

- Per-process protection domains
  - enforce security policy
  - any access is subject to access control
  - do not interfere with sharing

- Advantages
  - works with MMU-less microprocessors
  - cheap context switches on ARM7/ARM9
  - Based on UNSW Mungi system

IGUANA CONCEPTS

- Memory section
  - unit of VM allocation and protection
  - can be an encapsulated object with methods and data

- Thread
  - execution abstraction, as in L4

- Server
  - thread associated with memory section
  - invoked through methods with well-defined interfaces

- Protection domain
  - defines access and resource rights of a thread
  - corresponds to a process in traditional OS

IGUANA PHILOSOPHY

- Small and lightweight
  - geared towards embedded systems
  - allow optimal utilisation of hardware

- Strong yet unintrusive protection
  - hide protection machinery from most apps
  - able to emulate most standard protection models

- Support for resource management
  - in principle, although it isn’t implemented yet!

- Legacy support
  - designed to run Linux server

- Code and concept re-use
  - utilise what we’ve leaned in and developed for Mungi project

IGUANA OBJECTS

- Six kinds of objects
  1. memory sections
  2. threads
  3. protection domains (PDs)
  4. sessions
  5. resource tokens (restoks)
  6. external spaces
     - not full Iguana objects
     - serve as proxies for non-Iguana objects

- Access controlled by capabilities

OBJECTS: COMMONALITIES

- Objects have a unique name — object ID (OID)
  - OIDs are addresses in Iguana’s SAS
  - only for memory sections does this address correspond to actual memory

- Objects have methods that can be invoked
  - one method that exists for all objects: destroy
  - each kind of object has a set of pre-defined methods

- Objects are created by invoking constructor on a PD:
  - kind zap = pd->new(kind(args));

- Methods are grouped into interfaces
  - interfaces also have unique IDs (IIDs) that are OID + interface number
  - interfaces have capabilities
  - grant rights to invoke an interface’s methods
  - all pre-defined methods belong to separate interfaces
  - i.e., access is individually protected
**Differences to Mungi**
- SAS name space extended to all objects
  - not just memory
  - supports unified access control and resource management
- Secondary storage not part of SAS
  - meant to run on 32-bit hardware
  - meant to run on embedded systems without disk
- Provision for management of all resources (i.e. objects)
  - only half worked out yet
  - one honours student working on details
- Client-server model instead of migrating-threads model
  - more familiar to potential users
- Avoids some (as yet) unresolved resource management issues
- Aimed at commercial deployment → more conventional
  - Mungi is a pure research system

**Iguana Capabilities**
- A capability is a token that confers access right(s)
- Two kinds of capabilities in Iguana:
  - master capability
    - created when an object is created
    - confers rights on all methods of object
    - allows creation of further capabilities
  - invocation capability
    - created when an interface is created
    - confers right to invoke methods of a single interface
- Capabilities are only active if stored in PD's capability lists
  - as in Mungi

**Memory Sections**
- Memory sections represent virtual memory
  - allocation of a certain amount of virtual memory:
    ```
    main_cap = pd->new_word((size));
    ```
- Memory sections are the only objects that support user-defined methods
  - others have pre-defined (standard) methods only
- Used to provide encapsulated services:
  - service = memory (data) + server (thread) + methods

**Memory Sections...**
- To create a service:
  - register a server thread on memory section
    ```
    base->new_server(thread_id);
    ```
  - base is the base address (OID) of the memory section
  - register interfaces (user-defined methods)
    ```
    base = iid->new_cap();
    ```
    - iid refers to number of new interface
  - Registering interfaces supports user-defined methods
    - remember: each interface can have one or more methods
    - interface number only interpreted by server
      - similarly, the method number is an opcode delivered to the interface
    - IIDs and method numbers allocated by system implementor
      - part of the service's interface protocol

**Memory Sections: Pseudo Methods**
- Read (R), write (W), execute (X) are logically considered methods
  - subjects them to same protection mechanisms as other methods
  - no actual methods exist corresponding to those operations
- Further pseudo-method is `clist(C)`
  - allows holder to add an object to the Clist array

**Threads**
- Iguana threads are essentially L4 threads:
  - threads within same PD operated on by plain L4 syscalls
  - corresponding to local L4 threads (i.e., same L4 AS)
  - direct IPC to non-local threads is not allowed
  - use method invocations (corresponding to server thread)
  - presently not enforced by Iguana
  - requires enhancements to L4 (forthcoming API) to do efficiently
  - will provide attribute to ensure enforcement (at a cost)
- Certain operations require privileges
  - e.g. thread creation and deletion done by privileged L4
    - `ThreadControl()` call
  - Done by Iguana on invocation of appropriate methods

**Thread Operations**
- Thread creation:
  ```
  thread_cap = pd->new_thread(id, thread_id);
  ```
  - returns two kinds of thread IDs
    - Iguana thread ID (iid), part of the thread_cap
    - used for protection and other Iguana-specific purposes
    - L4 thread ID (l4_id)
      - used for L4 syscalls
  - New thread created inactive
    - can be activated by:
      ```
      l4syscall ExchangeRegisters() (local threads only)
      Iguana method tid->start(ip,ep)
      ```

**Thread Operations...**
- Obtain L4 thread ID
  ```
  l4tid = tid->l4_thread();
  ```
- Obtain own thread ID
  ```
  tid = myself();
  ```
- Obtain protection domain of thread
  ```
  pd = tid->domain();
  ```
- Obtain and modify scheduling parameters
  ```
  tid->schedule_info(info);
  ```
**SESSIONS**

- Sessions reduce authentication overheads of repeated calls
- Prior to invoking methods on a service, must establish session
  
  ```
  session = pd->new_session(server);
  ```

  establishes session between target PD and server
  - `server` is a PD ID

  **Note:** This is likely to change

  - Iguana informs the server by invoking its notification method
    
    ```
    server->session_created(pd);
    ```

  - Iguana notifies remaining partners if the session is destroyed
    
    ```
    pd or server->session_destroyed(session);
    ```

**Iguana Capabilities**

- Iguana capabilities are user-level objects
  - `password capability`, consisting of OID and password
    
    | object ID | password |
    |-----------|----------|

  - Length of password is configurable (normally ≥ 64 bits)
  - Same model as in Mungi
    - Implicit presentation
    - Two-level structure (Clists caps array, Clists)
    - Same confinement approach

**EXTERNAL SPACES**

- External spaces are “raw” L4 address spaces
  - not part of Iguana SAS

  - Provided to deal with restrictions of Iguana model
    - 32-bit address space may not be large enough to share between all protection domains
    - legacy support (e.g. strict `fork()` semantics) may require separate address spaces

  - External spaces come at a cost
    - unable to make full use of fast address-space switching on ARM
    - not well integrated with Iguana world
      - no fine-grained access control provided by Iguana capabilities
      - not allowed to communicate with any PD other than creator
      - not even with Iguana — cannot invoke methods
        - enforced via L4 redirectors

**EXTERNAL Spaces — Operations**

- Creation requires explicit specification of KIP and UTCB address
  
  ```
  es = pd->new_es(kip, utcb_area);
  ```

  - Thread creation also requires arguments similar to L4
    
    ```
    ltid = es->new_thread(pager, scheduler, starter, utcb);
    ```

**Hardware Access**

- Device drivers need to access raw hardware features

  - Iguana provides a (static) `hardware` object for this
    
    ```
    hardware->back_mem(adr, p_adr, caching);
    ```

    - maps the memory section (`adr`) to the specified physical address with
      specified caching attributes

  - Interrupt association:
    
    ```
    hardware->register_interrupt(tid, irq);
    ```

    - registers the specified thread as the handler of the specified interrupt

**Resource Tokens**

- Iguana’s resource management mechanism
  - Note: presently this only exists conceptually
    
    - details of the model still need to be worked out
    - however, model is based on our experience with a similar model in Mungi

  - Basic idea: all resources have a price that must be paid by the user
    - Model provides great flexibility for defining charging details
    - Generalisation of Mungi bank accounts

**WOMBAT: A PORTABLE LINUX SERVER ON L4**

**Motivation**

- Provide Linux API
- Support architectures other than x86
- Be easily maintainable
- Integrate with Iguana

**Legacy APIs on L4**

- Microkernel, like L4, provides mechanisms, not services
  
  - services implemented by user-level server
  - microkernel is OS agnostic
  
  - can, in principle, provide any OS API (OS “personality”)
  - can provide multiple OS APIs

  - Multiple OS APIs are a classical motivation for microkernels

  - Simplest approach is to port an existing monolithic kernel
    
    - e.g., Unix, Linux
    - even several running concurrently

  - Past experience poor:
    - Mach Unix server
    - IBM Workplace OS (Mach-based)

  - Attributed to poor Mach IPC performance, large cache footprint
**Porting Monolithic Kernels: Pros**
- Code reuse: Existing services
  - can use existing code unchanged (file systems, network stacks, drivers, ...)
  - potential reduction in maintenance cost (but ...)
- Code reuse: Legacy support
  - can support huge number of existing apps (if sufficiently compatible)
  - dramatic reduction of cost of deploying new system
- Coder reuse
  - can provide a familiar target for programmers
  - reduction of cost of maintaining system and new app development
- Buzzword reuse
  - can jump on Linux bandwagon while really doing something more sensible...
  - good PR value ;-)  
  - can support huge number of existing apps (if sufficiently compatible)
  - reduction of cost of deploying new system
  - can provide a familiar target for programmers
  - can support huge number of existing apps (if sufficiently compatible)
  - reduction of cost of deploying new system
  - can provide a familiar target for programmers

**Porting Monolithic Kernels: Cons**
- Performance
  - Microkernel inherently adds overhead
  - microkernel-mediated rather than direct access to hardware
  - IPC to server (4 mode switches) rather than syscall (2 mode switches)
  - Legacy system on L4 will *always* be slower than native!
  - Question is: How much slower???
- Miss out on most advantages of microkernels
  - software-engineering advantages
    - microkernels can provide strongly encapsulated components with small interfaces
    - Linux approach is unmaintainable in the long run [Schach et al. Maintainability of the Linux Kernel, IEE Proceedings, 149:18–23 (2002)]
    - reduction of trusted computing base
    - trusting 3-3Mloc vs 10kloc kernel plus core services and drivers
    - potential for formal verification of TCB?
    - might be feasible if it’s small
    - definitely infeasible with something the size of Linux
- Main-stream systems have notoriously poor real-time support
  - typically unsuitable for hard real time

**Hybrid Systems: Best of Two Worlds?**
- Legacy server and native apps running side-by-side
  - code/coder/buzzword reuse
  - software-engineering advantages for critical/new parts
  - hard real-time apps not affected by legacy system
- How about performance?
  - L4 primitives much faster than Mach and others
  - fast enough?

**L4 on Linux: History**

**L’Linux**
- First port of Linux kernel to L4
- Done for ia64 at Dresden [SOSP 97]
  - modified architecture-specific part of Linux to use L4 syscalls
  - binary compatibility: syscall redirection to Linux server
  - syscall exception delivered to L4 via exception IPC
    - better performance using modified libc
    - syscall trap replaced by L4 IPC to Linux server
  - re-done for 2.2, 2.4, 2.6
  - L4/Linux implementation highly dependent on Linux internals
  - significant effort for forward porting (start from scratch each time)

**Wombat Linux Server**
- Done at NICTA 03–04
  - done from scratch, independent of Dresden work
- Portable between architectures
  - based on Iguana
  - presently runs on x86, ARM, MIPS64
  - MIPS64 supports 32-bit and 64-bit executables
- Much reduced dependence on Linux internals
- Binary compatibility via syscall redirection
- Proper Linux scheduling of Linux apps
- Potential to achieve good performance on ARM

**A System Running Wombat**

**Wombat Implementation**
- Dresden L’Linux approach:
  - applied modifications to architecture-dependent part of kernel
    - very x86-specific
- Wombat approach:
  - introduced new arch/14/ in Linux source
    - kept as architecture-neutral as possible
  - architecture-dependent code in arch/14/xsys-arm etc
    - otherwise only modified 3 Linux source files
    - two are one-line bug fixes (patches submitted)
    - third is printf (additional early debug output)
    - tracked head revision of Linux from 2.5.x to 2.6.4
    - now at 2.6.6, will do 2.9 within a week
PAGE FAULTS AND EXCEPTIONS

- Wombat has a single server thread per CPU, like L/Linux
  - is L4 pager and exception handler for Linux processes
  - Linux “syscalls” are L4 exceptions
    - use SWI instruction
    - private instruction
    - other exceptions are reflected back as Linux signals
- Have single runnable Linux user process per CPU, unlike L/Linux
  - differs from Dresden/Karlsruhe L/Linux approaches
  - helps to maintain Linux scheduling behaviour

LINUX PROCESSES

Wombat scheduling

- Wombat maintains a single server thread per CPU, unlike L/Linux
- Advantages of this approach:
  - no changes required to (architecture-independent) scheduler
  - no locking required other than what is already in Linux
- Disadvantages:
  - Linux scheduling only approximately maintained
    - actual scheduling happens next time process enters kernel after time slice expired
    - processes run normally slightly longer than Linux time slice
    - CPU-bound process runs up to twice as long
    - Should be able to get exact Linux scheduling (but at a cost)

WOMBAT PERFORMANCE (PRELIMINARY)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Server</th>
<th>Linux</th>
<th>trampoline L4 ratio</th>
<th>IPC L4 ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>getpid()</td>
<td>L/Linux</td>
<td>1.06</td>
<td>5.06</td>
<td>3.73</td>
</tr>
<tr>
<td></td>
<td>Wombat</td>
<td>0.41</td>
<td>1.94</td>
<td>4.7</td>
</tr>
<tr>
<td>2-proc ctsxw</td>
<td>L/Linux</td>
<td>7.00</td>
<td>18.2</td>
<td>2.63</td>
</tr>
<tr>
<td></td>
<td>Wombat</td>
<td>4.03</td>
<td>9.66</td>
<td>2.4</td>
</tr>
<tr>
<td>pipe</td>
<td>L/Linux</td>
<td>29.0</td>
<td>69.4</td>
<td>2.4</td>
</tr>
<tr>
<td></td>
<td>Wombat</td>
<td>6.3</td>
<td>16.5</td>
<td>2.6</td>
</tr>
</tbody>
</table>

- L/Linux 2.0 on 133MHz Pentium [SOSP 97], some improvements since
- Wombat 2.6 on Pentium-4 Xeon 2.66GHz, unoptimised
- overhead should be smaller on RISC processors

SYSTEM-CALL PROCESSING

- Access to application memory by Wombat:
  - provide copy to copy from functions
  - look up page table and find page(s) in server’s memory pool
  - on MIPS could instead run Wombat in supervisor mode
    - would give direct access to user's address space
- Context switching:
  - Wombat maintains pointer to state of current Linux process
  - On a process switch, switches state and updates pointer
    - on ARM, state is banked user registers
- Signal processing is architecture dependent
  - emulates Linux signal handling

WOMBAT SCHEDULING

- Wombat has second thread: timer
  - high-priority thread normally in timed wait
  - maintains Linux time slice
  - when woken:
    - checks whether same Linux process is still running
    - if so, sets reschedule flag
    - if flag already set then sets Linux process' total quantum to zero
    - forces preemption, and an IPC to the scheduler (the "kernel" thread)
- Scheduling happens when Linux process enters kernel
  - checks reschedule flag, if set:
    - invokes Linux scheduler
    - perform context switch, reset reschedule flag

LINUX IDLE TASK

- Linux's idle task executes cpu_idle()
  - architecture-specific function
  - in Wombat this IPCs to the "kernel" thread
  - that one just sleeps until the next time tick or interrupt
  - could just as well be made to go into a low-power mode

IGUANA DEVICE DRIVERS

- Drivers are Iguana services (data + server thread + methods)
  - At startup obtain access to hardware resources
    - can be done first-come, first-served
    - better: restrict allocation to privileged service
    - ... by restricting capabilities to the hardware object
    - privileged service can allocate device memory (using hardware->back_mem)
    - can then hand capability for device-backed memory section to driver
- Driver is then able to operate the hardware
  - provides methods as per Iguana driver protocol
**Sharing Iguana Device Drivers with Wombat**

- Wombat has stub driver module
  - invokes Iguana driver
  - sample exists for console

**Iguana Resource Manager**

- Proprietary Firmware
- Untrusted
- Sensitive App
- Sensitive App

**TARGET SYSTEM ARCHITECTURE**

- L4 Microkernel
- Hardware

**Similar to Dresden DROPS system**

**Status: Iguana, Wombat and Driver Framework**

- In use at (at least) two multinational companies
- Beta release Feb 05
- Proper release planned Nov 05
- Deployment in products planned for early next year
- Future work:
  - proper SMP implementation
  - support for heterogeneous multiprocessors (SoC)
  - design and implementation of proper resource management

**User-Level Device Drivers on Linux**

- BSD Sockets TX and RX
- Shared buffers
- Benchmarking setup

**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
  - Component-based 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

**SEL4: Secure Embedded L4**

- Goal
  - Satisfy high security requirements
  - Support certifiably secure systems
  - Requires proof of security properties
  - Present API is known not to support this

- Issues:
  - Information flow control
  - V2: clans and chiefs: clumsy, inflexible, inefficient
  - V4: redirectors: still clumsy, often inefficient
  - IPC addressing via thread IDs reveals global names
  - Violates need to know
  - Primitive kernel memory management
  - Allows denial-of-service attacks
  - Reveals internal state

- Collaboration with Dresden, Johns Hopkins (Jonathan Shapiro)
  - Kevin Elphinstone, 1 staff, students

**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

**Sel4: Secure Embedded L4**

- Goal
  - Satisfy high security requirements
  - Support certifiably secure systems
  - Requires proof of security properties
  - Present API is known not to support this

- Issues:
  - Information flow control
  - V2: clans & chiefs: clumsy, inflexible, inefficient
  - V4: redirectors: still clumsy, often inefficient
  - IPC addressing via thread IDs reveals global names
  - Violates need to know
  - Primitive kernel memory management
  - Allows denial-of-service attacks
  - Reveals internal state

- Collaboration with Dresden, Johns Hopkins (Jonathan Shapiro)
  - Kevin Elphinstone, 1 staff, students
**FORMAL VERIFICATION OF MICROKERNEL**

- First (and most significant) step to making the TCB trustworthy!
- Involves three aspects:
  1. develop a formal specification of the kernel API ➜ seL4
  2. prove safety properties about the API
  3. prove that implementation is correct ➜ never before been done for any protected kernel!
- Ultimate goal: Verify a real kernel (L4) that is used in production ➜ collaboration with NICTA Formal Methods Program
- Reduce risk: pilot project

**L4 VERIFICATION PILOT PROJECT**

- Purpose:
  - test tools and get experience with them
  - feasibility study
  - bridge cultural gap between theorists and kernel hackers
  - develop project plan for full-scale verification project
- Operates on a “slice” of the API (address-space management)
  - done formal model of (simplified) API slice
  - proved some properties (absence of loops in mappings)
  - refined specification down to source code ➜ page table operations and simplified mapping database

**L4 VERIFICATION PROJECT**

- Timeline:
  - commenced April 2005
  - 3 years
- Resources: ≈ 20 person years
  - presently:
    - 2 FTE researchers
    - 1 PhD student
    - 1 honours
  - definitely looking for students!
  - desired background:
    - good systems background and aptitude for maths, or
    - maths background and aptitude for systems

**PROJECT: L4 WCET ANALYSIS**

- Hard real-time systems need guarantees of worst-case latencies
  - Need worst-case execution time (WCET) of kernel operations
- Usually done:
  - by code inspection ➜ unsuitable for non-trivial kernel
  - experimentally ➜ unreliable: cannot guarantee 100% coverage!
  - by educated guess ➜ highly unreliable (RTLinux got it wrong) ➜ typically leads to vastly pessimistic latencies
  - Need systematic execution-time analysis of complete kernel ➜ never been done for non-trivial kernel!
  - project leader Stefan Petters done similar (non-kernel) work ➜ looking for students!

**COMPONENT-BASED SOFTWARE ARCHITECTURE**

- Goal: component architecture that is appropriate for microkernel ➜ based on Iguana
- To support:
  - effective software-engineering techniques
  - robustness by encapsulation without destroying performance
  - hot-swapping / hot upgrading of components
  - real-time (WCET) analysis of system
  - ultimately, formal verification of whole system
- Collaboration with NICTA Empirical Software Engineering Program ➜ 3 staff, several PhD and U/G students, visitors

**SECURE MOBILE CODE**

- Combines language and OS techniques for secure execution
- Code may be:
  - accompanied by a proof certificate establishing its trustworthiness
  - otherwise sandboxed execution
- Lead by Manuel Chakravarty ➜ presently working on theoretical foundations ➜ later to be integrated with Iguana ➜ requires configurable protection domains

**GOANNA: STATIC ANALYSIS OF SYSTEMS SOFTWARE**

- Model-checking-based tools looking for bugs in kernel & libs
- Collaboration with NICTA Formal Methods Program
- Pilot Project April 05 – March 06
  - examine state of the art
  - evaluate existing tools
  - analysis framework
  - get experience with model-checking of systems code
  - develop plan for follow-on project

**PROJECT: OS-FRIENDLINES OF ARCHITECTURES**

- Evaluate hardware-dictated OS overheads and trends
- Critical operations
  - kernel entry and exit
  - TLB refill
  - context switch
  - synchronization primitives
- Hardware characteristics
  - instruction-set architecture
  - clock speed
  - pipeline depth
  - issue width
- Aim: paper in architecture conference
**PROJECT: KERNEL IMPLEMENTATION TRADE OFFS**

- Idea: revisit some of Jochen’s tricks
  - virtually- vs physically-addressed TCBs
  - per-thread vs per-CPU kernel stack
  - exception IPC vs exception callback
- Question: how much influence of architecture?
- Aim: Conference paper

**PROJECT: L4DARWIN**

- Aim: Mac OS done right
  - remove Mach
  - substitute L4
  - de-privilege Darwin
  - de-privilege IOkit
  - kick-ass performance
- Aim: exercise for the reader

**GELATO: LINUX PERFORMANCE AND SCALABILITY**

- VM system performance and scalability
  - superpage support (particularly Itanium)
  - NUMA support
  - guarded page tables for large address spaces
- TCP/IP and NFS throughput and scalability
- NUMA-aware scheduling
- Aims: support
  - 1000’s of CPUs in a NUMA system
  - 1000’s of nodes in a cluster
  - terabytes of RAM
  - support commercial workloads
- Funded by HP and ARC, supported by SGI

**OTHER PROJECTS**

- Secure GUI
  - window system that ensures isolation of clients
- JVM on L4
- API emulation
  - support Symbian OS, QNX, ...
- Kernel ports
  - Blackfin, Hitachi, SPARC, Motorola
- Simulators
  - tools for OS development and architecture research
  - ARM, PowerPC, ...

**SUMMARY: OS RESEARCH AT UNSW/NICTA**

- Improve reliability and trustworthiness of embedded systems
  - make a **real** impact on the practice of embedded systems R&D
- Comprehensive approach used on using microkernel technology
- Combination of formal and practical approaches
- Leverage relevant competencies available within NICTA
- Ample opportunities to participate in cutting-edge research
  - summer projects
  - honours theses
  - PhD theses
  - employment