Lecture 6
From L3 to seL4: What Have We Learnt in 20 Years of L4 Microkernels?

Kevin Elphinstone and Gernot Heiser

Operating Systems Practical

12 November, 2014
Introduction and design principles

Brief history of microkernels

L4: Basic abstractions

L4: Design and implementation choices

Keywords

Questions
Outline

Introduction and design principles

Brief history of microkernels

L4: Basic abstractions

L4: Design and implementation choices

Keywords

Questions
Introduction to Operating Systems

- Operating system
- Kernel
- Monolithic kernel
- Microkernel
Operating system

- abbrv. OS
- Software (collection) to interface hardware with user
- Components:
  - Kernel: Linux, FreeBSD, Windows NT, XNU, L4, ...
  - Services/daemons: sysvinit, CUPS print server, udev, ...
  - Utilities: ls, Windows Commander, top
  - Other applications
Components directly interfacing with hardware
  Examples?
“Core” of OS
  No general definition of “core”
Monolithic vs. Micro-kernel

- VFS
- IPC, file system
- Scheduler, virtual memory
- Device drivers, dispatcher

Hardware

User Mode

Kernel Mode

Application

Syscall

Source: http://www.cse.unsw.edu.au/
Monolithic vs. Micro-kernel

**Monolithic kernel**
- IPC, scheduling, memory management
- File systems
- Drivers
- Higher-level API

**Microkernel**
- IPC, scheduling, memory management
- API closer to the hardware
Microkernel principles: minimality

- If it’s not critical, leave it out of the kernel
- Pros:
  - Small code base
  - Easy to debug
  - Trusted Computing Base, feasible for formal verification
- Cons:
  - Harder to find the “right” API design
  - Harder to optimize for high-performance
Drivers, file systems, etc. as user space services

Pros:
- Isolation $\Rightarrow$ limited attack surface
- High availability, fault tolerance
- Componentization, reusability

Cons:
- Performance: IPC is a bottleneck
Microkernel principles: policy freedom

- Kernel provides mechanisms, **not** policies
- Policy definition is left up to the user space application
- **Pros:**
  - Flexibility
- **Cons:**
  - Hard to achieve, e.g. for scheduling
  - May lead to application bloat
- **Example:** kernel provides user with memory, allocation algorithm depends on app
- **Example:** cache maintenance is explicitly exposed to user space, to improve performance
Introduction and design principles

Brief history of microkernels

L4: Basic abstractions

L4: Design and implementation choices

Keywords

Questions
0th Generation (1970s)

- Nucleus [Brinch Hansen ’70]
- Hydra [Wulf et al ‘74]
- Issues
  - Lack of hardware support
  - Bad performance
1st Generation (1980s)

- Mach
- Chorus
- Issues
  - Stripped-down monolithic kernels
  - Big
  - Bad performance: 100\(\mu s\) IPC
2nd Generation (1990s, early 2000s)

- Minix
- L3, L4 [Lietdke '95]
- Performance-oriented
  - From scratch design
  - Architecture-dependent optimizations, e.g. reduced cache footprint
  - L3 was fully implemented in assembly
- Issues
  - Security
L4 family tree

Source: http://www.cse.unsw.edu.au/
L4 family tree

Source: http://www.cse.unsw.edu.au/
3rd Generation (2007+)

- OKL4 Microvisor [Heiser and Leslie ’10]
- Microkernel **and** hypervisor
- Replaces some of the mechanisms with hypervisor mechanisms
- Deployed in older Motorola phones
3rd Generation (2007+)

- seL4 [Elphinstone et al '07, Klein et al '09]
- Security-oriented
  - Capability-based access control
  - Strong isolation
- Memory management policy fully exported to user space
  - Kernel objects are first class citizens
  - **All** memory is explicitly allocated
- Formally verified [Klein et al '09]
Outline

Introduction and design principles

Brief history of microkernels

L4: Basic abstractions

L4: Design and implementation choices

Keywords

Questions
What mechanisms to abstract?

- **Bare minimum:**
  - Processor
  - Memory
  - Interrupts/exceptions

- **Must replace memory isolation with communication protocols**
  - Communication (IPC)
  - Synchronization
## Hypervisor vs. Microkernel

<table>
<thead>
<tr>
<th>Resource</th>
<th>Hypervisor</th>
<th>Microkernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>Virtual MMU (vMMU)</td>
<td>Address space</td>
</tr>
<tr>
<td>CPU</td>
<td>Virtual CPU (vCPU)</td>
<td>Thread or scheduler activation</td>
</tr>
<tr>
<td>Interrupt</td>
<td>Virtual IRQ (vIRQ)</td>
<td>IPC message or signal</td>
</tr>
<tr>
<td>Communication</td>
<td>Virtual NIC</td>
<td>Message-passing IPC</td>
</tr>
<tr>
<td>Synchronization</td>
<td>Virtual IRQ</td>
<td>IPC message</td>
</tr>
</tbody>
</table>

Source: [http://www.cse.unsw.edu.au/](http://www.cse.unsw.edu.au/)
Abstracting memory

- Address space, fundamentally:
  - A collection of virtual → physical mappings
- Ways to expose this to user:
  - Array of (physical) frames or (virtual) pages to be mapped
  - Cache for mappings which might vanish (Virtual TLB)
Abstracting execution

- Threads, vCPUs
- What defines a thread?
- Migrating threads
  - Thread might be moved to different address space
Abstracting execution

- Scheduling: map threads to CPUs
- What is the scheduling policy?
- Simple round-robin
- Policy-free scheduling?
Communication abstraction

- Inter-Process Communication (IPC)
- Synchronous, asynchronous ≠ blocking, non-blocking
- Traditional L4 IPC is fully synchronous
- Asynchronous notification
  - Sender asynchronous, receiver blocking and synchronous
  - Similar to Unix’s `select`
Interrupt abstraction

- Hardware faults are abstracted through IPC
- Synchronous exceptions, page faults, etc.
- Interrupts are asynchronous notifications
  - Thread must register as a pagefault/exception/interrupt handler
Access control

How do we specify objects?

- IDs in a global list
  - Provably insecure
  - Can DDoS, create covert channels, etc.
- IDs in per-address space lists
- Capabilities
Access control: capabilities

- Developed in KeyKOS, Coyotos, Amoeba, L4 Pistachio, OKL4, seL4, ...

- A **token**
  - owned by the *subject* (e.g. a thread)
  - as *proof* that it has access rights to an *object* (e.g. a kernel object)

- All inter-domain accesses are mediated by capabilities
Introduction and design principles

Brief history of microkernels

L4: Basic abstractions

L4: Design and implementation choices

Keywords

Questions
Implementation language

- Initial L3 and L4: 100% x86 assembly
- Pistachio, OKL4 microkernel: C, C++, assembly
- OKL4 Microvisor, seL4: C
- seL4: Haskell prototype for correctness proof
seL4, OKL4: “Endpoints” as IPC targets
  ▶ Decouple target from actual service
  ▶ Fully signal-like asynchronous IPC (OKL4 Microvisor)
Access control and resource management

- seL4: access control based on delegable capabilities
- Take-grant model
- Provable security
  - Information leaks are impossible
  - ... if the policy is correct
  - ... and the implementation is correct
  - ... and the compiler is correct
  - ... and the hardware isn’t faulty
Access control and resource management

- seL4: resources are exposed as capabilities to physical memory
- May be:
  - Mapped
  - Delegated to children domains
  - Delegated to kernel: “retyped” into kernel objects
Preemption in the kernel

- Interrupts are disabled when running in kernel
- Microkernel is in general non-preemptable
- Preemption points for long-running operations
Scheduling contexts (Fiasco.OC)
- Separate scheduling parameters from threads
- Allow implementing hierarchical scheduling [Lackorzyński et al ’12]

Policy-free scheduling still unresolved
Multi-processors

- Initial L4 design is uniprocessor
- seL4: same, due to formal verification constraints
- Possible approach: multikernels [M Von Tessin ’12]
Introduction and design principles

Brief history of microkernels

L4: Basic abstractions

L4: Design and implementation choices

Keywords

Questions
Keywords

- microkernel
- l4
- thread
- address space
- inter-process communication
- access control
- capability
- preemption
▶ http://dl.acm.org/citation.cfm?id=224075
▶ http://www.cse.unsw.edu.au/~cs9242/13/lectures/
▶ http://os.inf.tu-dresden.de/L4/
▶ http://os.inf.tu-dresden.de/fiasco/
▶ http://www.ok-labs.com/products/okl4-microvisor
Introduction and design principles

Brief history of microkernels

L4: Basic abstractions

L4: Design and implementation choices

Keywords

Questions
Questions