



Open Kernel LabsTM

Be open. Be safe.

UNSW

Introduction

COMP9242

2008/S2 Week 1

Part 1

These slides are distributed under the Creative Commons Attribution 3.0 License

- You are free:
 - **to share** — to copy, distribute and transmit the work
 - **to remix** — to adapt the work
- Under the following conditions:
 - **Attribution.** You must attribute the work (but not in any way that suggests that the author endorses you or your use of the work) as follows:
 - “Courtesy of Gernot Heiser, [Institution]”, where [Institution] is one of
 - “UNSW”, “NICTA”, or “Open Kernel Labs”
- The complete license text can be found at <http://creativecommons.org/licenses/by/3.0/legalcode>

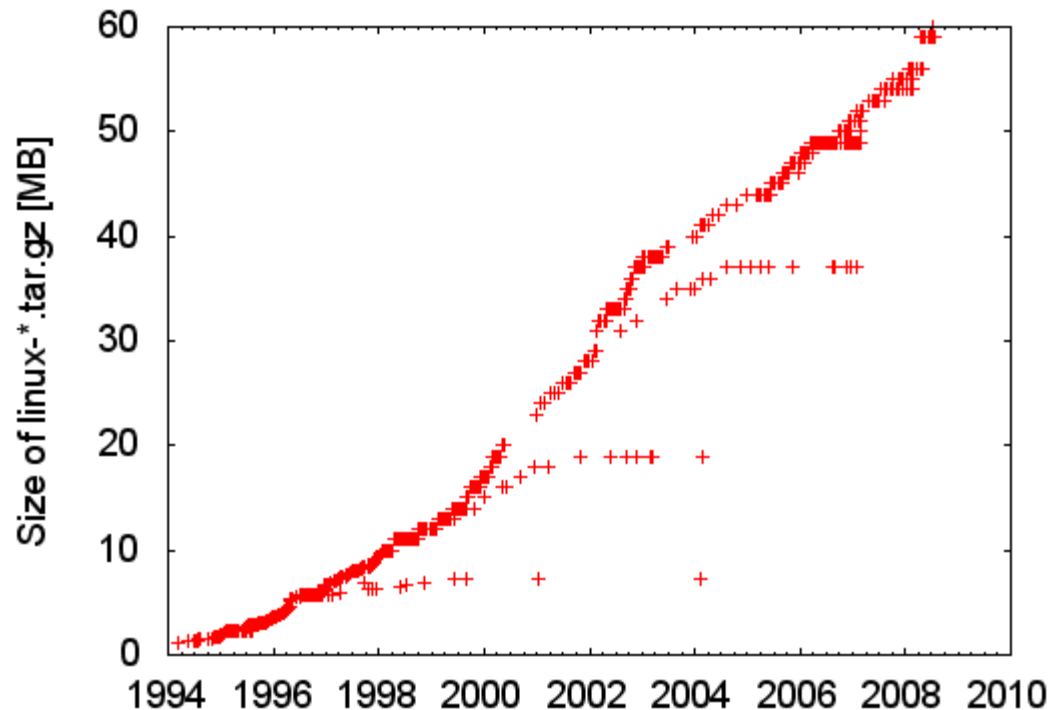
Outline

- *Introduction: What are microkernels?*
- Microkernel Performance
- L4 History and Future
- Basic L4 concepts

My Microkernels?

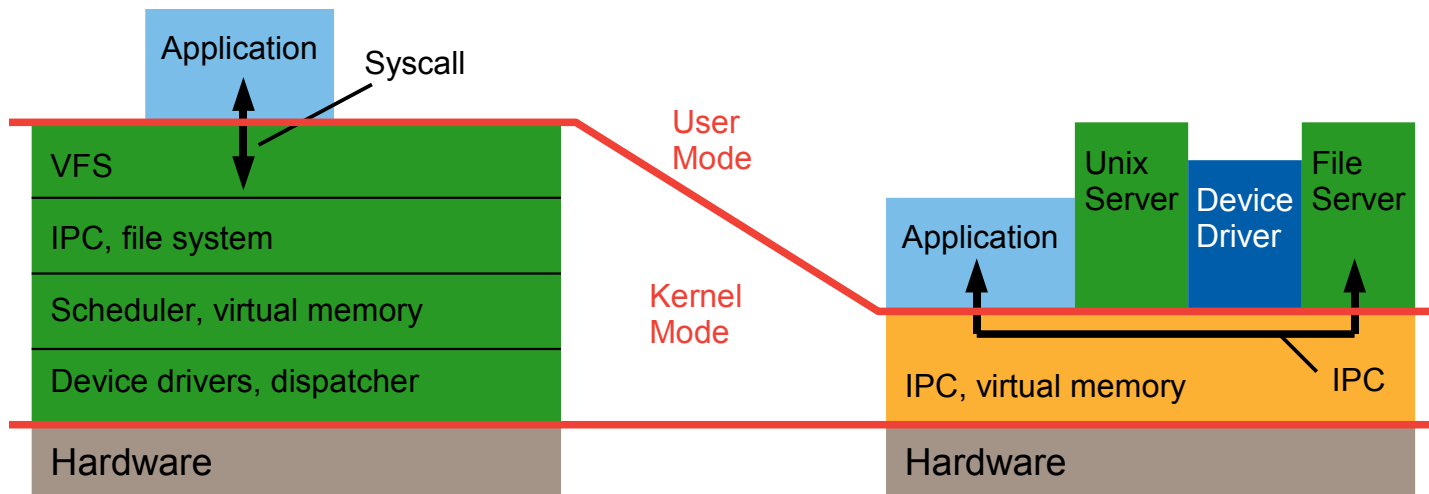
Monolithic Kernel

- Kernel has access to everything
 - all optimizations possible
 - all techniques/mechanisms/concepts implementable
- Can be extended by simply adding code
- Cost: complexity
 - growing size
 - limited maintainability



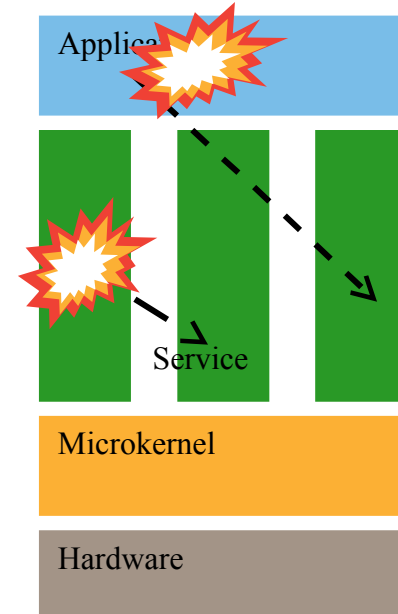
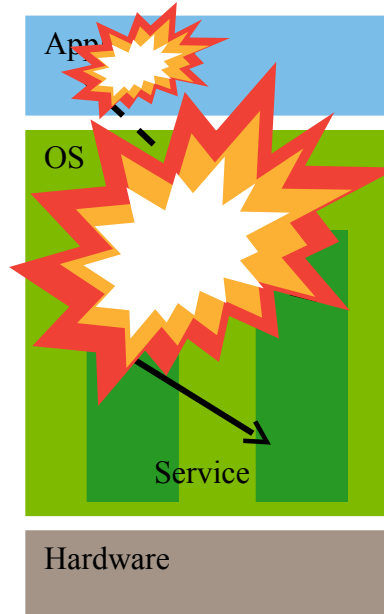
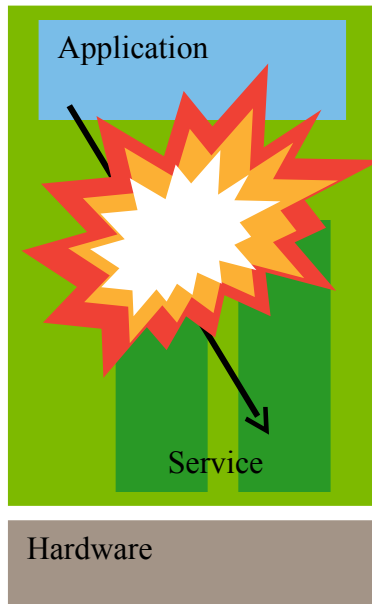
Microkernel: Idea

- Small kernel providing core functionality
 - Only code running in privileged mode
- Most OS services provided by user-level servers
- Applications communicate with servers via message-passing IPC



Trusted Computing Base (TCB)

Definition: The part of the system which can circumvent security



System: traditional embedded

Linux/
Windows

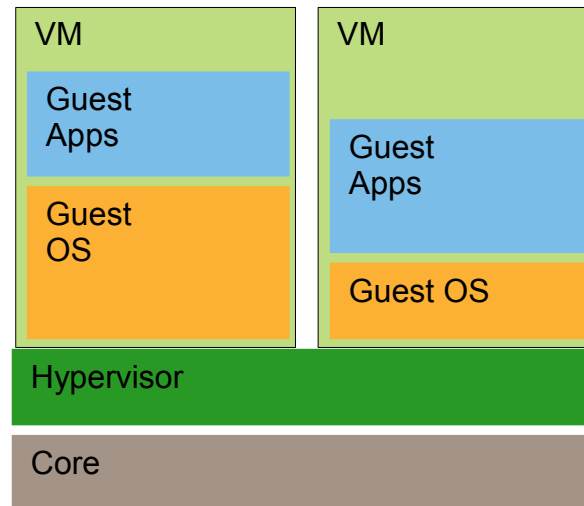
Microkernel-based

TCB: **all** code

100,000's LOC

10,000's LOC

- Partition system into several subsystems
 - Each partition runs its own operating system
 - Hypervisor controls resources
 - Hypervisor is kind-of microkernel



- Typical uses
 - Server consolidation: multiple logical machines on single physical
 - Embedded systems: high-level OS co-hosted with RTOS

- Combat kernel complexity, increase robustness, maintainability
 - dramatic reduction in amount of kernel code
 - modularity with hardware-specific services
 - normal resource management available to OS services
- Flexibility, adaptability
 - policies at user level, subject to change
 - additional services provided by adding servers
- Hardware interaction
 - hardware-dependent part of system is small, easy to optimise
- Security, safety
 - internal protection boundaries

REALITY CHECK!
slow, inflexible
100usec IPC

Outline

- Introduction: What are microkernels?
- *Microkernel Performance*
- L4 History and Future
- Basic L4 concepts

→ **First-generation microkernels**

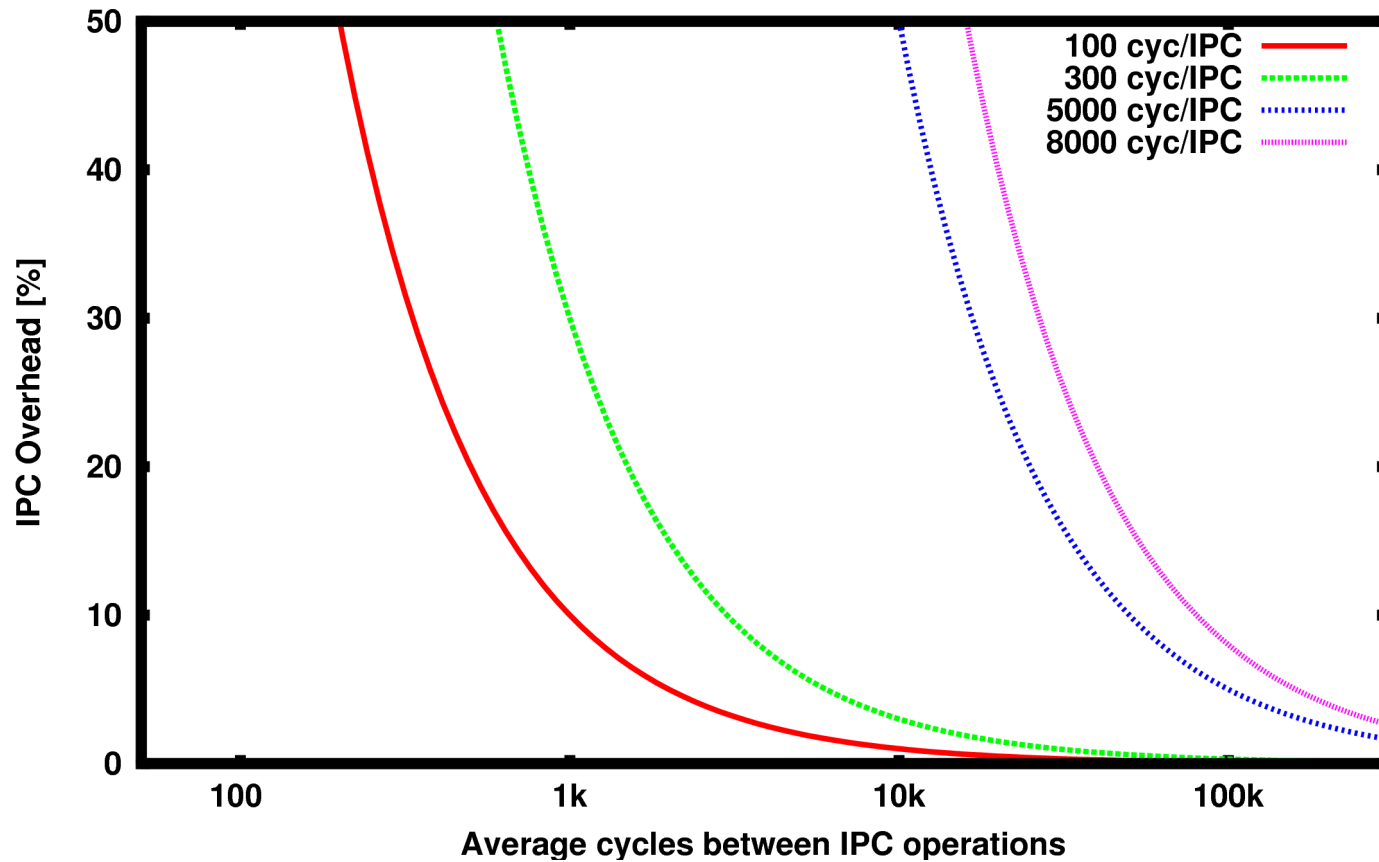
- Mach, Chorus, Amoeba, QNX
... were slow...
- 100 μ s IPC
- almost independent of clock speed!

→ L4 did better

- Close to hardware cost
- 20 times faster than Mach on identical hardware (i486)

Microkernel	IPC cost [Cycles]
Mach	5750
Amoeba	6000
Spin	6783
L4	250

IPC Cost Implications



L4 Performance: Cross Address-Space IPC

Architecture	Intra-core Cycles	Inter-core Cycles
ARM XScale PXA255 400MHz	155	
MIPS-64 100MHz dual core	109	690
Pentium 3	305	
AMD-64	230	
Itanium 2	36	

- IPC overhead generally within 20% of bare hardware cost
- Essentially as fast as it gets

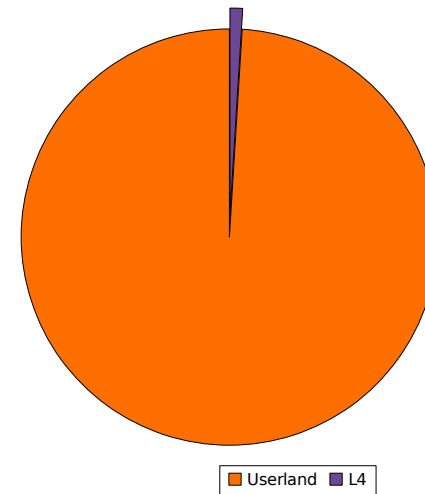
First-generation microkernels were slow

→ Reason: Poor design [Liedtke SOSP'95]

- complex API
- Too many features
- Poor design and implementation
- Large cache footprint \Rightarrow memory-bandwidth limited

→ L4 is fast due to small cache footprint

- 10–14 I-cache lines
- 8 D-cache lines
- Small cache footprint \Rightarrow CPU limited

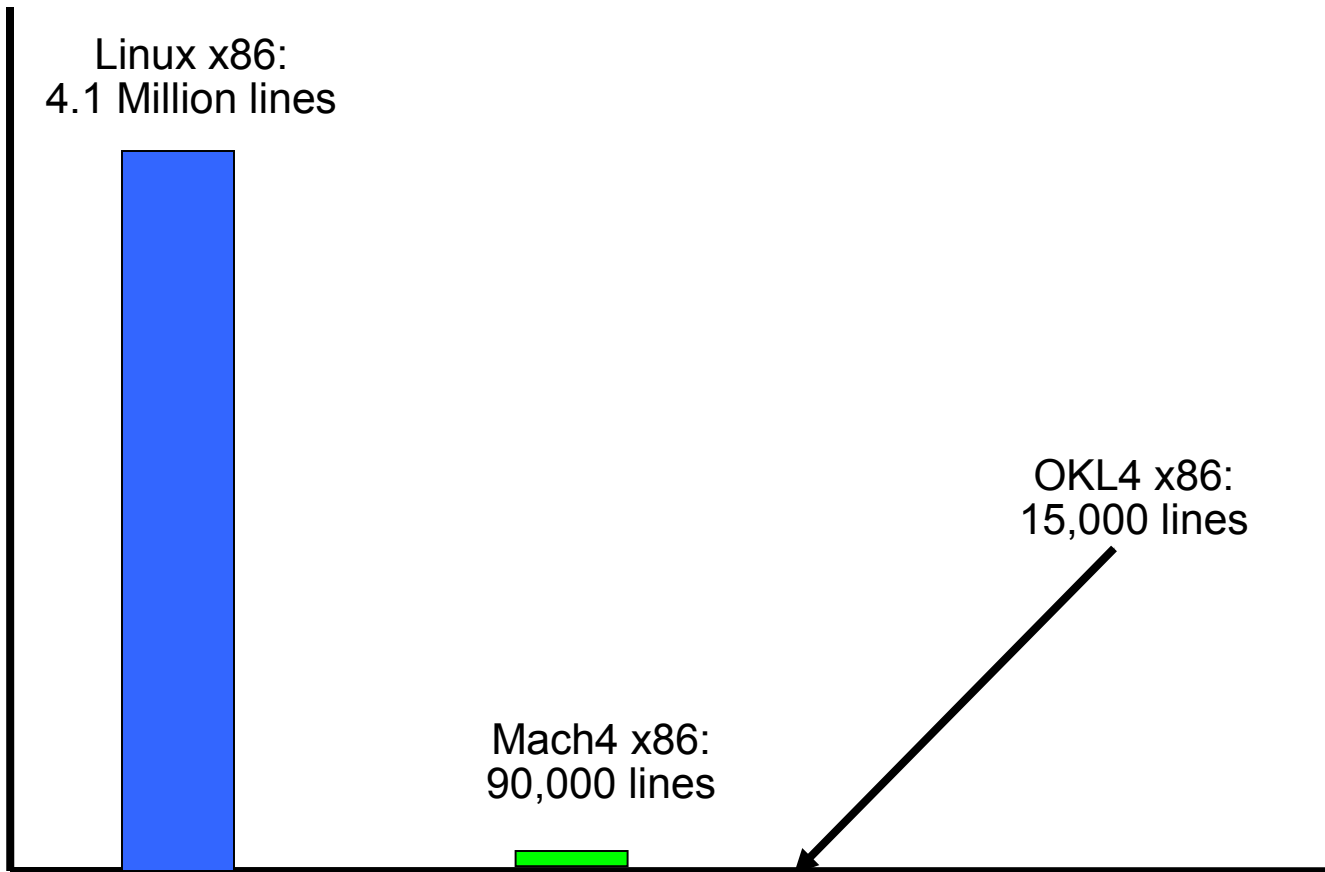


What makes Microkernel Fast?

- Small cache footprint — but how?
 - Minimality: no unnecessary features
 - Orthogonality: complementary features
 - Well-designed, and *well implemented* from scratch!
- Kernel provides *mechanisms*, not *services*
- Microkernel design principle (Minimality):

A feature is only allowed in the kernel if this is required for the implementation of a secure system.
- “*Small is beautiful!*”

Size Comparison



L4 Kernel Size

→ Source code (OKL4)

- \approx 9k LOC architecture-independent
- \approx 0.5–6k LOC architecture/platform-specific

→ **Memory** footprint kernel (not aggressively minimised):

- Using gcc (poor code density on RISC/EPIC architectures)

Architecture	Version	Text	Total
x86	L4Ka	52k	98k
Itanium	L4Ka	173k	417k
ARM	OKL4	48k	78k
PPC-32	L4Ka	41k	135k
PPC-64	L4Ka	60k	205k
MIPS-64	NICTA	61k	100k

→ Fast IPC path footprint (typical)

- 10-14 I-cache lines
- 8 D-cache lines

Outline

- Introduction: What are microkernels?
- Microkernel Performance
- *L4 History and Future*
- Basic L4 concepts