Advanced Operating Systems

COMP9242
Introduction
Staff

• Lecturer in Charge
  – Gernot Heiser
• Lecturer
  – Kevin Elphinstone
• Various Support Staff
  – TBA
Why are you here?

• You’ve done comp3231
  – Did well (minimum credit)
  – You would like delve deeper into issues of modern OS construction
  – You’d like more challenging projects to really get your hands dirty
• Curious about where research is heading in the field of operating systems.
• Thinking about doing a thesis project in operating systems
What can you expect?

• Challenging Project
• Lectures in general:
  – Background required for project
  – Exposure to local research projects
  – An in-depth look at OS issues
    • Building upon the background in COMP3231
  – Exposure to recent and seminal research papers
  – Guest lectures by active researchers (PhD students and local researchers)
Project Goal

Provide students with a deeper understanding of Operating Systems through practical experience.

• Approach: Participate in the design and implementation of a simple operating system (SOS).
Project Aims

• Provide experience in OS design and development, including:
  – Microkernel based systems (L4::Pistachio).
    • User-level OS servers.
    • User-level page fault handlers.
  – Device drivers
  – Performance evaluation
  – Implications of cache architectures
  – Exposure to alternative OS Designs

• Demonstrate the importance of design

• Provide experience of being a team member in a large software project.
Project Aims

• Expose students to a mostly realistic OS development environment.
  – Similar to professional OS and or embedded systems developer.
• Give an understanding of what’s involved in constructing an entire OS on bare hardware.
• Give an understanding of the interaction between low-level software and hardware.
• Encourage you to undertake a thesis, or do research within Distributed Systems Group.
Prerequisites

• Students are expected to be very competent C programmers.
• Students are expected to be familiar with
  – basic computer architecture concepts.
  – Assembly language (read-only)
  – Basic RISC processor characteristics (we’ll use a MIPS R4600 for the project)
Lectures
(subject to change)

• Introduction and Overview
• Introduction to the L4 Microkernel
  – L4 system calls and usage (to get you started on the project)
• A close look at selected OS issues
  – Protection, capabilities
  – Caching, and its implications for OS
  – Page tables for wide address spaces
  – SMP issues: locking, cache coherence, scheduling
  – File systems
Lectures
(subject to change)

• Microkernels and User-level Servers
  – History and motivation for microkernel systems, Hydra, Mach, discussion, experiences; second-generation microkernel systems, L4, Exokernel, Spin; design and implementation of microkernel-based systems, including user-level page fault handling and device drivers

• Microkernel Implementation
  – A detailed look at a real microkernel (L4Ka::Pistachio).

• Persistent systems and Single-address-space operating systems
  – Concepts and examples; UNSW Mungi project
Project/Lab Work

• Build a simple operating system (SOS) from the ground up
• Major component of the course
• Use L4Ka::Pistachio
  – ported to MIPS here by Carl van Schaik
• Develop and test on U4600 computers
  – R4600 based machines design and built by Kevin Elphinstone and Dave Johnson
  – “Clean” machine to get your hands dirty
• Can also use CPU simulator Simular
  – Developed locally
  – Demos must be on real hardware
Project

• Some warm-up experiments
• Students will work in groups of two
• End goal:
  – To produce a small efficient operating system
• Project will have a series of due milestones
  – Demo to pass the milestone and be awarded marks
  – Help you manage your time
  – Avoid major problems
Milestones

• Details released RSN (week 2)
• Late milestones
  – Less than one week late, 25% of the milestone mark is lost
  – More than one week late, but still less than two weeks late, 50% of the milestone mark lost
Alternative Projects

• Special arrangements might be made for particular student to do alternative projects
  – Must be at least as challenging as the original project
  – Must convince us that you can actually do it.
Assessment

• 65% for project work
• 35% for final exam
  – A minimum of 14 (40%) required in final exam to pass

• Final Exam
  – 24hr take-home exam
  – Read and analyse two recent research papers and submit a critical report
Textbook

• No particular textbook for course
  – See course web page for useful reference books
• Selected research papers referred to in the course
L4 and Microkernels

Background
Monolithic Kernels - Advantages

• Kernel has access to everything, potentially:
  – All optimizations are possible
  – All techniques/mechanisms/concepts are implementable

• Can be extended by simply adding more code to the kernel
Linux Kernel Evolution

For reference:

Linux 2.4.18 = 2.7 million lines of code
Approaches to tackling complexity

- Monolithic approaches
  - Layered Kernels
  - Modular Kernels
  - Object Oriented Kernels

- Alternatives
  - Extensible Kernels
  - Microkernels
History

- monolithic kernels
History

- monolithic kernels

- 1\textsuperscript{st}- generation μ-kernels
  - Mach \textit{CMU, OSF} \textbf{External Pager}
  - Chorus \textit{Inria, Chorus}
  - Amoeba \textit{Vrije Universiteit} \textbf{User-Level Driver}
  - (L3) \textit{GMD}
Brief History

• monolithic kernels

• 1\textsuperscript{st} generation μ-kernels
  – Mach \textit{CMU, OSF} \textcolor{Green}{External Pager}
  – Chorus \textit{Inria, Chorus}
  – Amoeba \textit{Vrije Universiteit}
  – (L3) \textit{GMD} \textcolor{Red}{User-Level Driver}

• 2\textsuperscript{nd} generation μ-kernels
  – (Spin) \textit{U Washington}
  – Exokernel \textit{MIT}
  – L4 \textit{GMD / IBM / UKa}
classic +

- classic OS
- Security
- RT MM
- L4
- HW

thin

- native Java
- embedded app
- L4
- HW

specialized

- highly-specialized component
- L4
- HW
Project

sosh
sosh
SOS
L4
HW
The Great Promise

- coexistence of different
  - APIs
  - file systems
  - OS personalities
- flexibility
- extensibility
- simplicity
- maintainability
- security
- safety
The Great Promise

The Big Disaster

- coexistence of different
  - APIs
  - file systems
  - OS personalities
- flexibility
- extensibility
- simplicity
- maintainability
- security
- safety

- SLOW
- UNFLEXIBLE
- LARGE
Macro $\mu$
The 100-\(\mu\)s Disaster
The 100-µs Disaster
IPC Costs (486, 50 MHz)
L4 IPC

- 0.04 µs (P III 500 MHz) (hopefully)
- 0.47 µs (P III 500 MHz)
- 0.36 µs (P III 500 MHz)
- 0.73 µs (Pentium 166 MHz)
- 0.91 µs (R4600 100 MHz)
- 0.10 µs (21164 433 MHz)
Thesis:

• A μ-Kernel does the Job
  • if Properly Designed
  • if Carefully Implemented

Minimality
Elegance
Architectural Integration

Efficiency
Flexibility
When analyzing IPC performance,

Cycles are not the only the to consider!!
Processor-DRAM Gap (latency)

"Moore’s Law"

Processor-Memory Performance Gap:
(grows 50% / year)

μProc
60%/yr.

DRAM
7%/yr.

Slide originally from Dave Patterson, Parcon 98
Today’s Situation: Microprocessor

- Microprocessor-DRAM performance gap
  - time of a full cache miss in instructions executed
    1st Alpha (7000): \(340 \text{ ns}/5.0 \text{ ns} = 68 \text{ clks} \times 2 \text{ or } 136\)
    2nd Alpha (8400): \(266 \text{ ns}/3.3 \text{ ns} = 80 \text{ clks} \times 4 \text{ or } 320\)
    3rd Alpha (t.b.d.): \(180 \text{ ns}/1.7 \text{ ns} = 108 \text{ clks} \times 6 \text{ or } 648\)
  - \(1/2X\) latency \(\times 3X\) clock rate \(\times 3X\) Instr/clock \(\Rightarrow \approx 5X\)
Cache Working Sets

L1 cache
- 1024 cache lines (16K + 16K)
- 12 lines used for IPC

L2 cache
- 8192 cache lines (256K)
- 12 lines used for IPC
Other Complications

• P4 trace cache
  – A cache of recently translated μ-ops
  – Flushed on every page-table switch

• Virtual Caches
  – Need to be flushed on address space switch
A $\mu$-kernel does no real work.
$\mu$-Kernel services are only required to overcome $\mu$-kernel constraints.

Therefore, $\mu$-kernels have to be infinitely fast!

Minimality is the key!

- Threads
- Address Spaces

$IPC$

$Mapping$
Threads, Communication

Address Spaces

Threads
Drivers at User Level

- IO ports: part of the user address space
- Interrupts: messages from hardware
Address Spaces
Address Spaces

- map
Address Spaces

- map
- unmap
Address Spaces

- map
- unmap
- grant
Page Fault Handling

Application → Pager

"PF" msg

Pager → Application

map msg
Page Fault Handling

Application

Pager

PF $\rightarrow$ IPC

res $\rightarrow$ IPC

"PF" msg

map msg
Address Spaces
Address Spaces
Address Spaces

Pager 1

Pager 2

Initial AS

Physical Memory
Address Spaces

Physical Memory

Initial AS

Pager 1

Pager 2

Pager 3

Pager 4
Address Spaces

Physical Memory

Initial AS

Pager 1

Pager 2

Pager 3

Pager 4

Application

Application

Application
Address Spaces

- Physical Memory
  - Initial AS
    - Pager 1
    - Pager 2
    - Pager 3
    - Pager 4
  - Application
    - Application
    - Application
    - Application
  - Driver
    - Driver
Mach Virtual Memory

In comparison

Inflexible

Application

Application

Mach

Physical Memory

Paging Policy
Size Comparison

Linux (all platforms):

- 2.7 Million lines

Mach 4 x86:

- 90,000 lines

L4Ka::Pistachio/ia32

- 10,000 lines
**classic +**

- classic OS
- Security
- RT MM

**thin**

- native Java
- embedded app

**specialized**

- highly-specialized component