Advanced Operating Systems

COMP9242
Introduction

Why are you here?

• You’ve done comp3231
  – Did well (minimum credit)
  – You would like to delve deeper into OS issues
  – You’d like to get your hands really dirty
• Curious about where field is heading.
• Thinking about doing a thesis in OS
• Thinking about a job in embedded systems industry

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.

Staff

• Lecturer in Charge
  – Gernot Heiser
• Lecturer
  – Kevin Elphinstone
• Various Support Staff
  – TBA
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 OS Group / NICTA.

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
  - Detailed look at a real microkernel (L4Ka::Pistachio).
- Persistent and single-address-space systems
- OS Projects at UNSW/NICTA
  - Gelato@UNSW, NICTA Embedded OS, Mungi

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 Sulima
  - 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 on class web site
- Late milestones:
  - Max 1 week late: 25% of milestone mark lost
  - 1-2 weeks late: 50% of milestone mark lost
  - More than two weeks: all marks lost
- Bonus task
  - Much pain for little gain ;-)  
  - Don’t go overboard
  - Full mark doesn’t require bonus!

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 have some relevance to you or us.
  - Must convince us that you can actually do it.
  - Unable to get bonus marks

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
- Manuals provided in hardcopy and via class web site

L4 and Microkernels

Background
Monolithic Kernels - Advantages

- Kernel has access to everything, potentially:
  - All optimisations are possible
  - All techniques/mechanisms/concepts are implementable
- Can be extended by simply adding more code to the kernel

Linux Kernel Evolution

Approaches to tackling complexity

- Monolithic approaches
  - Layered Kernels
  - Modular Kernels
  - Object Oriented Kernels
- Alternatives
  - Extensible Kernels
  - Microkernels

History

- monolithic kernels
Brief History

- monolithic kernels

- 1st generation µ-kernels
  - Mach
  - Chorus
  - Amoeba
  - (L3) GMU

- 2nd generation µ-kernels
  - (Spin)
  - Exokernel
  - L4

Project

The Great Promise

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

The Big Disaster
Thesis:

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

When analyzing IPC performance,
Cycles are not the only the to consider!!

Processor-DRAM Gap (latency)

Today’s Situation: Microprocessor

- Microprocessor-DRAM performance gap
  - time of a full cache miss in instructions executed
    1st Alpha (7000): 340 ns/5.0 ns = 68 clks x 2 or 136
    2nd Alpha (6400): 266 ns/3.3 ns = 80 clks x 4 or 320
    3rd Alpha (t.b.d.): 180 ns/1.7 ns = 108 clks x 6 or 648
  - 1/2X latency x 3X clock rate x 3X Instr/clock ⇒ ≈5X

Cache Working Sets

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 µ-kernel does no real work. µ-Kernel services are only required to overcome µ-kernel constraints. Therefore, µ-kernels have to be infinitely fast!

Minimality is the key!

- Threads
- Address Spaces
  - IPC
  - Mapping

Drivers at User Level

INTR

Device

Driver

User

= ipc

- IO ports: part of the user address space
- interrupts: messages from hardware

Address Spaces

- map
Address Spaces

Page Fault Handling

Address Spaces

Page Fault Handling

Address Spaces

Address Spaces

Physical Memory
Address Spaces

Initial AS

Pager 1

Pager 2

Physical Memory

Address Spaces

Initial AS

Pager 1

Pager 2

Physical Memory

Address Spaces

Application

Application

Application

Page 1

Page 2

Initial AS

Physical Memory

Address Spaces

Application

Application

Application

Page 1

Page 2

Initial AS

Physical Memory

Mach Virtual Memory

In comparison

Application

Application

Application

Physical Memory

Paging Policy

Mach

Size Comparison

Linux (all platforms):

2.7 Million lines

L4/Ka::Pistachio/i32
10,000 lines

Mach 4 x86:
90,000 lines
Classic + HW

OS

Security

Thin

Native

Self-contained HW

Specialized

Highly-specialized component

Highly-specialized component

Native

HW