Introduction

Comp3231
Operating Systems
Kevin Elphinstone

Course Outline

• Prerequisites
  – COMP2011 Data Organisation
  – COMP2021 Digital Systems Structure
  – or the postgraduate equivalent

  – You are expected to be competent programmers!!!!
    • We will be using the C programming language
      – The dominant language for OS implementation.

Lectures

• Wednesday, 2-4pm
• Thursday, 5-6pm
  – All lectures are here (EE LG1)
  – The lecture notes will be available on the course web site
    • Available prior to lectures, when possible.
  – The lecture notes and textbook are NOT a substitute for attending lectures.

Tutorials

• Start in week 2
• A tutorial participation mark will contribute to your final assessment.
  – Participation means participation, NOT attendance.
  – Comp9201 students excluded
• You will only get participation marks in your enrolled tutorial.

Announcement

• We have added an extra tutorial on Thursday at 10am.
• Consider moving if you can.

Assignments

• Assignments form a substantial component of your assessment.
• They are challenging!!!!
  – Because operating systems are challenging
• We will be using OS/161,
  – an educational operating system
  – developed by the Systems Group At Harvard
  – It contains roughly 20,000 lines of code and comments
Assignments

• Don’t under estimate the time needed to do the assignments.
• If you start a couple days before they are due, you will be late.
• To encourage you to start early,
  – Bonus 10% of max mark of the assignment for finishing a week early
  – To iron out any potential problems with the spec, 5% bonus for finishing within 48 hours of assignment release.
  – See course handout for exact details
    • Read the fine print!!!!

Assignments

• We usually offer advanced versions of the assignments
  – Available bonus marks are small compared to amount of effort required.
  – Student should do it for the challenge, not the marks.
  – Attempting the advanced component is not a valid excuse for failure to complete the normal component of the assignment

Assignments

• Four assignments
  – due roughly week 3, 6, 9,13
• The first one is trivial
  – It’s a warm up to have you familiarize yourself with the environment and easy marks.
  – Do not use it as a gauge for judging the difficulty of the following assignments.

Assignments

• Late penalty
  – 4% of total assignment value per day
    • Assignment is worth 20%
    • You get 18, and are 2 days late
    • Final mark = 18 – (20*0.04*2) = 16 (16.4)
  – Assignments are only accepted up to one week late. 8+ days = 0

Assignments

• To help you with the assignments
  – We dedicate a tutorial per-assignment to discuss issues related to the assignment
  – We provide labs for the first few weeks to ease the learning curve.
    • Details soon
    • Wed 4-7pm is the most likely candidate.

Assignments

• We take cheating seriously!!!
• Penalties include
  – Copying of code: 0 FL
  – Help with coding: negative half the assignment’s max marks
  – Originator of a plagiarised solution: 0 for the particular assignment
  – Team work: 0 for the particular assignments
**Cheating Statistics**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>enrolment</td>
<td>178</td>
<td>410</td>
<td>320</td>
<td>300</td>
<td>107</td>
<td>298</td>
<td>156</td>
<td>333</td>
<td>133</td>
</tr>
<tr>
<td>suspected</td>
<td>10(6%)</td>
<td>26(6%)</td>
<td>22(7%)</td>
<td>26(9%)</td>
<td>20(19%)</td>
<td>15(5%)</td>
<td>???(??%)</td>
<td>???(??%)</td>
<td></td>
</tr>
<tr>
<td>full penalties</td>
<td>2</td>
<td>6*</td>
<td>9*</td>
<td>14*</td>
<td>10*</td>
<td>95*</td>
<td>21*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reduced penalties</td>
<td>7</td>
<td>15</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>cheaters failed</td>
<td>4</td>
<td>10</td>
<td>16</td>
<td>16</td>
<td>12</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>cheaters suspended</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

*Note: Full penalty 0 FL not applied prior to 2001/S1

---

**Exams**

- There is NO mid-session
- The final written exam is 2 hours
- Supplementary exams are oral.
  - Supplementaries are available according to school policy, not as a second chance.

---

**Assessment**

- Exam Mark Component
  - Max mark of 100
- Based solely on the final exam
- Class Mark Component
  - Max mark of 100
- Based solely on the final exam
- 10% tutorial participation
- 90% Assignments

**Assessment**

- The final assessment is the harmonic mean of the exam and class component.
- If C >= 40, and E >= 40

\[
M = \frac{2EC}{E + C}
\]

---

**Assessment**

- If C < 40 or E < 40

\[
M = \min\left(44, \frac{2EC}{E + C}\right)
\]

---

**Harmonic Mean**
Harmonic Mean

Harmonic Mean (Class Mark = 100 - Exam Mark)

Assessment

• You need to perform reasonably consistently in both exam and class components.
• Harmonic mean only has significant effect with significant variation.

Textbook


References

• A. Tannenbaum, A. Woodhull, Operating Systems–Design and Implementation, Prentice Hall, ???
• John O’Gorman, Operating Systems, MacMillan, 2000
• Uresh Vahalla, UNIX Internals: The New Frontiers, Prentice Hall, 1996
• McKusick et al., The Design and Implementation of the 4.4 BSD Operating System, Addison Wesley, 1996

Consultations/Questions

• Email your questions to cs3231@cse.unsw.edu.au
  – Expect a 24 hr turn-around time.
  • We reserve the right to ignore email sent directly to us.
• Consultation Times
  – Kevin Elphinstone: Tuesday 10-11am, venue TBA
  • Please phone me if I’m not there.
  – Cameron Stone: TBA, venue TBA

Introduction to Operating Systems

Chapter 1 – 1.3
What is an Operating System?

Viewing the Operating System as an Abstract Machine
- Extends the basic hardware with added functionality
  - Provides high-level abstractions
    - More programmer friendly
    - Common core for all applications
  - It hides the details of the hardware
    - Makes application code portable

Viewing the Operating System as a Resource Manager
- Responsible for allocating resources to users and processes
- Must ensure
  - No Starvation
  - Progress
  - Allocation is according to some desired policy
    - First-come, first-served; Fair share; Weighted fair share; limits (quotas), etc…
  - Overall, that the system is efficiently used

Viewing the Operating System as the Privileged Component
The Operating System is Privileged

- Applications should not be able to interfere or bypass the operating system
  - OS can enforce the “extend machine”
  - OS can enforce resource allocation policies
  - Prevent applications from interfering with each other

- Note: Some Embedded OSs have no privileged component, e.g. PalmOS
  - Can implement OS functionality, but cannot enforce it.

Why Study Operating Systems?

- There are many interesting problems in operating systems.
- For a complete, top-to-bottom view of a system.
- Understand performance implications of application behaviour.
- Understanding and programming large, complex, software systems is a good skill to acquire.

(A brief) Operating System History

- Largely parallels hardware development
- First Generation machines
  - Vacuum tubes
  - Plug boards
    - Programming via wiring
    - Users were simultaneously designers, engineers, and programmers
    - “single user”
    - Difficult to debug (hardware)
  - No Operating System

Second Generation Machines

- IBM 7094
  - 0.35 MIPS, 32K x 36-bit memory
  - 3.5 million dollars
- Batching used to more efficiently use the hardware
  - Share machine amongst many users
  - One at a time
  - Debugging a pain
  - Drink coffee until jobs finished

Batch System Operating Systems

- Sometimes called “resident job monitor”
- Managed the Hardware
- Simple Job Control Language (JCL)
  - Load compiler
  - Compile job
  - Run job
  - End job
- No resource allocation issues
  - “one user”

Problem: Keeping Batch Systems Busy

- Reading tapes or punch cards was time consuming
- Expensive CPU was idle waiting for input
Third Generation Systems - Multiprogramming

- Divided memory among several loaded jobs
- While one job is loading, CPU works on another
- With enough jobs, CPU 100% busy
- Needs special hardware to isolate memory partitions from each other
  - This hardware was notably absent on early 2nd gen. batch systems

Job turn-around time was still an issue.

- Batch systems were well suited to
  - Scientific calcs
  - Data processing
- For programmers, debugging was much easier on older first gen. machines as the programmer had the machine to himself.
- Word processing on a batch system?

An then…

- Further developments (hardware and software) resulted in improved techniques, concepts, and operating systems….

Time sharing

- Each user had his/her own terminal connected to the machine
- All user’s jobs were multiprogrammed
  - Regularly switch between each job
  - Do it fast
- Gives the illusion that the programmer has the machine to himself
- Early examples: Compatible Time Sharing System (CTSS), MULTICS

The Advent of the PC

- Large Scale Integration (LSI) made small, fast(-ish), cheap computers possible
- OSs followed a similar path as with the mainframes
  - Simple “single-user” systems (DOS)
  - Multiprogramming without protection, (80286 era, Window 3.1, 95, 98, ME, etc…)
  - “Real” operating systems (UNIX, WinNT, etc..)
Operating Systems

- Exploit the hardware available
- Provide a set of high-level services that represent or are implemented by the hardware.
- Manages the hardware reliably and efficiently
- Understanding operating systems requires a basic understanding of the underlying hardware

Basic Computer Elements

- **CPU**
  - Performs computations
  - Load data to/from memory via system bus
- **Device controllers**
  - Control operation of their particular device
  - Operate in parallel with CPU
  - Can also load/store to memory (Direct Memory Access, DMA)
  - Control register appear as memory locations to CPU
  - Or I/O ports
  - Signal the CPU with "interrupts"
- **Memory Controller**
  - Responsible for refreshing dynamic RAM
  - Arbitrating access between different devices and CPU

The real world is logically similar, but a little more complex
A Simple Model of CPU Computation

• The fetch-execute cycle

- Load memory contents from address in program counter (PC)
- The instruction
- Execute the instruction
- Increment PC
- Repeat

CPU Registers

- PC: 0x0300
- SP: 0xcbf3
- Status
- R1
- Rn

A Simple Model of CPU Computation

- Stack Pointer
- Status Register
  - Condition codes
    - Positive result
    - Zero result
    - Negative result
- General Purpose Registers
  - Holds operands of most instructions
  - Enables programmers to minimise memory references.

Privileged-mode Operation

- To protect operating system execution, two or more CPU modes of operation exist
  - Privileged mode (system-, kernel-mode)
    - All instructions and registers are available
  - User-mode
    - Uses ‘safe’ subset of the instruction set
    - E.g. no disable interrupts instruction
    - Only ‘safe’ registers are accessible

‘Safe’ registers and instructions

- Registers and instructions are safe if
  - Only affect the state of the application itself
  - They cannot be used to uncontrollably interfere with
    - The operating system
    - Other applications
  - They cannot be used to violate a correctly implemented operating system policy.

Privileged-mode Operation

- The accessibility of addresses with an address space changes depending on operating mode
  - To protect kernel code and data

Address Space

- Accessible only to Kernel-mode
- Accessible to User- and Kernel-mode

0x00000000
I/O and Interrupts

- I/O events (keyboard, mouse, incoming network packets) happen at unpredictable times
- Direct memory access (DMA)
  - I/O exchanges occur directly with memory
  - Processor directs I/O controller to read/write to memory
  - Relieves the processor of the responsibility for data transfer
  - Processor free to do other things
- How does the CPU know when to service an I/O event?

Interrupts

- An interruption of the normal sequence of execution
- A suspension of processing caused by an event external to that processing, and performed in such a way that the processing can be resumed.
- Improves processing efficiency
  - Allows the processor to execute other instructions while an I/O operation is in progress
  - Avoids unnecessary completion checking (polling)

Interrupt Cycle

- Processor checks for interrupts
  - If no interrupts, fetch the next instruction
  - If an interrupt is pending, divert to the interrupt handler

Classes of Interrupts

- Program exceptions (also called synchronous interrupts)
  - Arithmetic overflow
  - Division by zero
  - Executing an illegal/privileged instruction
  - Reference outside user’s memory space.
- Asynchronous (external) events
  - Timer
  - I/O
  - Hardware or power failure

Interrupt Handler

- A program that determines the nature of the interrupt and performs whatever actions are needed.
- Control is transferred to the handler by hardware.
- The handler is generally part of the operating system.

Simple Interrupt Processing

- Device controller issues an interrupt
- Processor finishes current instruction and changes to kernel mode
- Processor makes interrupts and transfers flow of control (new PC)
  - Kernel SP-loaded
  - User SP & PC pushed on kernel stack
  - Remainder of user state is saved
  - Interrupt is processed including acknowledging the interrupt at the device level
  - User SP & PC restored
  - Hardware
  - Software
  - Hardware
- Processor unmarks interrupts and returns to user mode.
Multiple Interrupts

- Sequential interrupts
  - Processor ignores any new interrupt signals
  - Interrupts remain pending until current interrupt completes
  - Upon completion, processor checks for additional interrupts

Multiple Interrupts

- Prioritised (nested) interrupts
  - Processor ignores any new lower-priority interrupt signals
  - New higher-priority interrupts interrupt the current interrupt handler
  - Example: when input arrives from a communication line, it needs to be absorbed quickly to make room for more input

Programmed I/O

- Also called polling, or busy waiting
- I/O module (controller) performs the action, not the processor
- Sets appropriate bits in the I/O status register
- No interrupts occur
- Processor checks status until operation is complete
  - Wastes CPU cycles

Interrupt-Driven I/O

- Processor is interrupted when I/O module (controller) ready to exchange data
- Processor is free to do other work
- No needless waiting
- Consumes a lot of processor time because every word read or written passes through the processor

Direct Memory Access

- Transfers a block of data directly to or from memory
- An interrupt is sent when the task is complete
- The processor is only involved at the beginning and end of the transfer

Multiprogramming (Multitasking)

- Processor has more than one program to execute.
  - Some tasks waiting for I/O to complete
  - Some tasks ready to run, but not running
- Interrupt handler can switch to other tasks when they become runnable
- Regular timer interrupts can be used for timesharing
Memory Hierarchy

- Going down the hierarchy
  - Decreasing cost per bit
  - Increasing capacity
  - Increasing access time
  - Decreasing frequency of access to the memory by the processor
  - Hopefully
  - Principle of locality!!!!!!

Cache

- Cache is fast memory placed between the CPU and main memory
  - 1 to a few cycles access time compared to RAM access time of tens – hundreds of cycles
  - Holds recently used data or instructions to save memory accesses.
  - Matches slow RAM access time to CPU speed if high hit rate
  - (90%)
  - Is hardware maintained and (mostly) transparent to software
  - Sizes range from few kB to several MB.
  - Usually a hierarchy of caches (2–5 levels), on- and off-chip.
  - Block transfers can achieve higher transfer bandwidth than single words.

Processor-DRAM Gap (latency)

- Moore’s Law: Performance grows 50% / year
- Processor-Memory Performance Gap: (grows 50% / year)

Moving-Head Disk Mechanism
Example Disk Access Times

- Disk can read/write data relatively fast
  - 15,000 rpm drive - 80 MB/sec
  - 1 KB block is read in 12 microseconds
- Access time dominated by time to locate the head over data
  - Rotational latency
    - Half one rotation is 2 milliseconds
  - Seek time
    - Full inside to outside is 8 milliseconds
    - Track to track .5 milliseconds
- 2 milliseconds is 164KB in "lost bandwidth"

A Strategy: Avoid Waiting for Disk Access

- Keep a subset of the disk’s data in memory
  ⇒ Main memory acts as a cache of disk contents