Random Stuff

• No tutorials or comp3891/comp9283 lecture this week
• Release the warm-up exercise tomorrow
• New weeks tutorial questions – probably tomorrow also

Introduction to Operating Systems

Chapter 1 – 1.3
Chapter 1.5 – 1.9

Learning Outcomes

• High-level understand what is an operating system and the role it plays
• A high-level understanding of the structure of operating systems, applications, and the relationship between them.
• Some knowledge of the services provided by operating systems.
• Exposure to some details of major OS concepts.

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

### Kernel

- Portion of the operating system that is running in *privileged mode*
- Usually resident in main memory
- Contains fundamental functionality
  - Whatever is required to implement other services
  - Whatever is required to provide security
- Contains most-frequently used functions
- Also called the nucleus or supervisor

### The Operating System is Privileged

- Applications should not be able to interfere or bypass the operating system
  - OS can enforce the “extended machine”
  - OS can enforce its 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.
- Note: Some operating systems implement significant OS functionality in user-mode, e.g. User-mode Linux

### Structure of a Computer System

- **User Mode**
  - Application
  - System Libraries

- **Kernel Mode**
  - Operating System
  - Hardware
A note on System Libraries

- System libraries are just that, libraries of support functions (procedures, subroutines)
  - Only a subset of library functions are actually system calls
  - `strcmp()`, `memcp()` are pure library functions
  - `open()`, `close()`, `read()`, `write()` are system calls
  - System call functions are in the library for convenience

Operating System Objectives

- Convenience
  - Make the computer more convenient to use
- Abstraction
  - Hardware-independent programming model
- Efficiency
  - Allows the computer system to be used in an efficient manner
- Ability to evolve
  - Permit effective development, testing, and introduction of new system functions without interfering with existing services
- Protection

Services Provided by the Operating System

- Program development
  - Editors, compilers, debuggers
  - Not so much these days
- Program execution
  - Load a program and its data
- Access to I/O devices
- Controlled access to files
  - Access protection
- System access
  - User authentication
Services Provided by the Operating System

- Error detection and response
  - internal and external hardware errors
    - memory error
    - device failure
  - software errors
    - arithmetic overflow
    - access forbidden memory locations
    - operating system cannot grant request of application

Operating System Software

- Fundamentally, OS functions the same way as ordinary computer software
  - It is a program that is executed (just like apps)
  - It has more privileges
- Operating system relinquishes control of the processor to execute other programs
  - Reestablishes control after
    - System calls
    - Interrupts (especially timer interrupts)

Major OS Concepts

- Processes
- Concurrency and deadlocks
- Memory management
- Files
- Information Security and Protection
- Scheduling and resource management

Processes

- A program in execution
- An instance of a program running on a computer
- The entity that can be assigned to and executed on a processor
- A unit of resource ownership
  - A unit of activity characterized by a single sequential thread of execution, a current state, and an associated set of system resources
    - Nowadays the execution abstraction is separated out:
      - Thread
      - Single process can contain many threads
Process

- Consist of three segments
  - Text
    - contains the code (instructions)
  - Data
  - Global variables
  - Stack
    - Activation records of procedure
    - Local variables
- Note:
  - data can dynamically grow up
  - The stack can dynamically grow down

Memory

• Consists of three components
  – An executable program
    - text
  – Associated data needed by the program
    - Data and stack
  – Execution context of the program
    - All information the operating system needs to manage the process
      - Registers, program counter, stack pointer, etc...
    - A multithread program has a stack and execution context for each thread

Multiple processes creates concurrency issues

(a) A potential deadlock. (b) an actual deadlock.

Memory Management

- The view from thirty thousand feet
  - Process isolation
    - Prevent processes from accessing each other's data
  - Automatic allocation and management
    - Don't want users to deal with physical memory directly
  - Protection and access control
    - Still want controlled sharing
  - Long-term storage
  - OS services
    - Virtual memory
    - File system

Virtual Memory

- Allows programmers to address memory from a logical point of view
  - Gives apps the illusion of having RAM to themselves
  - Logical addresses are independent of other processes
  - Provides isolation of processes from each other
- Can overlap execution of one process while swapping in/out others.

Virtual Memory Addressing

Figure 2.10 Virtual Memory Addressing
File System

- Implements long-term store
- Information stored in named objects called files

Information Protection and Security

- Access control
  - regulate user access to the system
  - Involves authentication
- Information flow control
  - regulate flow of data within the system and its delivery to users

Scheduling and Resource Management

- Fairness
  - give equal and fair access to all processes
- Differential responsiveness
  - discriminate between different classes of jobs
- Efficiency
  - maximize throughput, minimize response time, and accommodate as many uses as possible

Operating System Structure

- The layered approach
  a) Processor allocation and multiprogramming
  b) Memory Management
  c) Devices
  d) File system
  e) Users
  - Each layer depends on the inner layers

In practice, layering is only a guide

- Operating Systems have many interdependencies
  - Scheduling on virtual memory
  - Virtual memory on I/O to disk
  - VM on files (page to file)
  - Files on VM (memory mapped files)
  - And many more…
The Monolithic Operating System Structure

- Also called the "spaghetti nest" approach
  - Everything is tangled up with everything else.
- Linux, Windows,

Learning Outcomes

- Understand the basic components of computer hardware
  - CPU, buses, memory, devices controllers, DMA, Interrupts, hard disks
- Understand the concepts of memory hierarchy and caching, and how they affect performance.

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
- Disk controller
- Printer controller
- Tape drive controller
- Memory controller
- Memory
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

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

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

The real world is logically similar, but a little more complex
‘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 within an address space changes depending on operating mode
  - To protect kernel code and data

I/O and Interrupts

- I/O events (keyboard, mouse, incoming network packets) happen at unpredictable times
- 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 software routine 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

User Mode

Application

Kernel Mode

Interrupt Handler

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!!!!

Memory Hierarchy

• Rough approximation of memory hierarchy

<table>
<thead>
<tr>
<th>Level</th>
<th>Access Time</th>
<th>Capacity Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>&lt;1 ns</td>
<td>1 GB</td>
</tr>
<tr>
<td>Main memory</td>
<td>100-1000 ns</td>
<td>1 GB-1 TB</td>
</tr>
<tr>
<td>Magnetic disk</td>
<td>2-10 sec</td>
<td>1 TB-10 TB</td>
</tr>
<tr>
<td>Magnetic tape</td>
<td>20-100 sec</td>
<td>20-100 TB</td>
</tr>
</tbody>
</table>

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
  – 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.
  – Also assumes probability of using newly fetched data is higher than the probability of reuse ejected data.

Processor-DRAM Gap (latency)

- ‘Moore’s Law’
- Processor-Memory Performance Gap: (grows 50% / year)
- DRAM 7%/yr.

Slide originally from Dave Patterson, Parcon 98
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

Two-level Memories and Hit Rates
- Given a two-level memory,
  - cache memory and main memory (RAM)
  - main memory and disk
  what is the effective access time?
- Answer: It depends on the hit rate in the first level.

Effective Access Time
\[
T_{eff} = H \times T_1 + (1 - H) \times (T_1 + T_2)
\]

Example
- Cache memory access time 1ns
- Main memory access time 10ns
- Hit rate of 95%

\[
T_{eff} = 0.95 \times 1 \times 10^{-9} + 0.05 \times (1 \times 10^{-9} + 10 \times 10^{-9})
= 1.5 \times 10^{-9}
\]