Introduction to Operating Systems

Chapter 1 – 1.3
Chapter 1.5 – 1.9

Learning Outcomes

• High-level understanding of 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

Operating System 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

Structure of a Computer System

OS interacts via load and store instructions to all memory, CPU and device registers, and interrupts
**Structure of a Computer System**

- Application
- System Libraries
- OS
- Device

Applications interact with themselves and via function calls to library procedures.

**Privilege-less OS**

- Some Embedded OSs have no privileged component
  - e.g. PalmOS, Mac OS 9, RTEMS
  - Can implement OS functionality, but cannot enforce it.
    - All software runs together
    - No isolation
    - One fault potentially brings down entire system

**Operating System Software**

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

**A Note on System Libraries**

System libraries are just that, libraries of support functions (procedures, subroutines)

- Only a subset of library functions are actually systems calls
  - `strcmp()`, `memcpy()`, are pure library functions
    - manipulate memory within the application, or perform computation
  - `open()`, `close()`, `read()`, `write()` are system calls
    - they cross the user-kernel boundary, e.g. to read from disk device
    - Implementation really focused on passing request to OS and returning result to application
- System call functions are in the library for convenience
  - try `man syscalls` on Linux

**Major OS Concepts (Overview)**

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

Process

- Minimally 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

Process state

- Consists of three components
  - An executable program
    - text
  - Associated data needed by the program
    - Data and stack
  - Execution context of the program
    - Registers, program counter, stack pointer
  - Information the operating system needs to manage the process
    - OS-internal bookkeeping, files open, etc...

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 others 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 to disk.
Virtual Memory Addressing

Memory management unit (hardware) translates program memory addresses to main memory addresses.

File System

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

Example File System

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 Internal Structure?
Classic 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

Unrealistic!

Operating System Structure

- In practice, layering is only a guide
  - Operating Systems have many interdependencies
    a) Scheduling on virtual memory
    b) Virtual memory (VM) on I/O to disk
    c) VM on files (page to file)
    d) Files on VM (memory mapped files)
    e) And many more...

The Monolithic Operating System Structure

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

The Monolithic Operating System Structure

- However, some reasonable structure usually prevails


Computer Hardware Review

Chapter 1.4
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
  – 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 more complex

A Simple Model of CPU Computation

• The fetch-execute cycle

![Diagram of CPU computation]

Figure 1.2 Basic Instruction Cycle
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

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.

Example Unsafe Instruction

- “cli” instruction on x86 architecture
  - Disables interrupts
- Example exploit
  cli /* disable interrupts */
  while (true)
  /* loop forever */;
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

Interrupt Terminology

- Program exceptions (sometimes called synchronous interrupts, traps)
  - Arithmetic overflow
  - Division by zero
  - Executing an illegal/privileged instruction
  - Reference outside user’s memory space.
- Asynchronous (external) interrupts (usually just called interrupts)
  - 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

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!!!!!

Caching as a general technique

- Given a two-levels data storage: small and fast, versus large and slow,
- Can speed access to slower data by using intermediate-speed memory as a cache.

A hardware approach to improving system performance?

CPU Registers
- Fast

Cache Memory (SRAM)
- Fast

Main Memory (DRAM)
- Slow

CPU Cache

- CPU 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 fetch data is higher than the probability of reusing ejected data.

Performance

- What is the effective access time of memory subsystem?
- Answer: It depends on the hit rate in the first level.

Effective Access Time

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

- \( T_1 \): access time of memory 1
- \( T_2 \): access time of memory 2
- \( H \): hit rate in memory 1
- \( T_{\text{eff}} \): effective access time of system
Example

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

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

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 OS approach to improving system performance?

A Strategy: Avoid Waiting for Disk Access

- Keep a subset of the disk’s data in main memory
  ⇒ OS uses main memory as a cache of disk contents

Application approach to improving system performance
A Strategy: Avoid Waiting for Internet Access

- Keep a subset of the Internet's data on disk
  ⇒ Application uses disk as a *cache* of the Internet