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?
Role 1: The Operating System is an Abstract Machine

- Extends the basic hardware with added functionality
- Provides high-level abstractions
  - More programmer friendly
  - Common core for all applications
    - E.g. Filesystem instead of just registers on a disk controller
- It hides the details of the hardware
  - Makes application code portable
Role 2: The Operating System is 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
Structural (Implementation) View: the Operating System is the Privileged Component
Operating System Kernel

• Portion of the operating system that is running in *privileged mode*
• Usually resident (stays) 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
Delving Deeper: The Structure of a Computer System

- Application
- System Libraries
- Application
- System Libraries
- Device
- Device
- Operating System
- Memory

User Mode
Kernel Mode
The Structure of a Computer System

OS interacts via load and store instructions to all memory, CPU and device registers, and interrupts.
Applications interact with themselves and via function calls to library procedures.
The Structure of a Computer System

Interaction via
System Calls

Application
System Libraries

User Mode
Kernel Mode
OS

Device
Device

Memory
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
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 mainly 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
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)
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/function/method
    - Local variables

Note:
- data can dynamically grow up
  - E.g., malloc()-ing
- The stack can dynamically grow down
  - E.g., increasing function call depth or recursion
Process state

• Consists of three components
  – An executable program code
    • 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
    - Users want to deal with data structures
    - Users don’t want to deal with physical memory directly
  - Protection and access control
    - Still want controlled sharing

- 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.

Figure 2.10 Virtual Memory Addressing
File System

- Implements long-term store
- Information stored in named objects called files
Example File System
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
    • Scheduling on virtual memory
    • Virtual memory (VM) 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,
The Monolithic Operating System Structure

- However, some reasonable structure usually prevails

The End