Silberschatz, et al.

Topics based on Chapter 3
Operating System Structures

Different "actors" view OS differently

- Operating system **designers--***system's components and their interconnections*
- **Users--***services provided by the operating system*
- **Programmers--***interface provided (i.e., system calls), their organization, and other abstractions*

Common system components

- Process Management
- Main-Memory Management
- Secondary-Storage Management
- File Management
- I/O System Management
- Protection System
- Networking
- Command-Interpreter System

Process management

- A *process* is a unit of work in a system.
- A process is a program in execution. A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.
- A program is passive; a process is dynamic

Process management

- The operating system is responsible for the following process management activities
  - process creation and deletion.
  - process suspension and resumption.
  - provision of mechanisms for:
    - process synchronization
    - process communication
    - deadlock handling

Main memory management

- Main memory is
  - large array of words or bytes, each with its own address
  - repository of quickly accessible data shared by CPU and I/O devices
  - volatile

Main memory management

- The operating system must
  - Keep track of which parts of memory are currently being used and by whom.
  - Decide which processes to load when memory space becomes available.
  - Allocate and deallocate memory space as needed.

Secondary storage management

- Persistent storage; larger capacity than primary storage
- Generally disks in modern systems
- Operating system responsibilities:
  - Free-space management
  - Storage allocation
  - Disk scheduling

File management

- Logical storage unit: *file* (an abstract concept that is mapped onto a physical implementation)
- Operating system responsibilities:
  - creation and deletion of files and directories
  - primitives for manipulating files and directories
  - mapping files onto secondary storage
  - backup of files on stable storage media

I/O System management

- Hides details of hardware devices from user
- I/O subsystem consists of
  - memory management component: buffering, caching, and spooling
  - general device-driver interface
  - drivers for specific hardware devices

Protection system

- Protection refers to a mechanism for controlling access by programs, processes, or users to both system and user resources.
- The protection mechanism must:
  - distinguish between authorized and unauthorized usage.
  - specify the controls to be imposed.
  - provide a means of enforcement.

Networking

- Distributed system: collection of processors that do not share memory, peripheral devices, or a clock (they have local memory and clock)
- Communicate through a communications network (many different routing and connection strategies)
- Provides user access to (heterogeneous) system resources
- Allows computation speedup, increased data availability, and enhanced reliability

Command-interpreter system

- Interface between user and operating system
- Some systems put into kernel; others treat as a program (e.g., Unix and MS-DOS)
- Control-statement-driven systems also called
  - control-card interpreter
  - command-line interpreter
  - shell
- Function: Get next command and execute it

Command-interpreter system

- Control statements may deal with:
  - process creation and management
  - I/O handling
  - secondary-storage management
  - main-memory management
  - file-system access
  - protection
  - networking

Users' view
Operating system services

- Program execution – system capability to load a program into memory and to run it.
- I/O operations – since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O.
- File-system manipulation – program capability to read, write, create, and delete files.

Operating system services

- Communications – exchange of information between processes executing either on the same computer or on different systems tied together by a network. Implemented via shared memory or message passing.
- Error detection – ensure correct computing by detecting errors in the CPU and memory hardware, in I/O devices, or in user programs.

Operating system services

- Services that ensure the efficient operation of the system
  - Resource allocation: allocating resources to multiple users or multiple jobs running at the same time.
  - Accounting: keep track of and record which users use how much and what kinds of computer resources for account billing or for accumulating usage statistics.
  - Protection: ensuring that all access to system resources is controlled.

Programmer's view
System calls

- Interface between process and operating system
- Generally called by assembly-language programs but may be available to higher-level language programmers in some systems (e.g., C, Bliss, BCPL, etc.)

System calls

- Three general methods are used to pass parameters between a running program and the operating system:
  - Pass parameters in registers.
  - Store the parameters in a table in memory, and the table address is passed as a parameter in a register.
  - Push (store) the parameters onto the stack by the program, and pop off the stack by the operating system.

System calls

- Major categories of system calls:
  - Process control
  - File manipulation
  - Device manipulation
  - Information maintenance
  - Communications

System calls
Process control

end, abort

load, execute

create process, terminate process

get process attributes, set process attributes

wait for time

wait event, signal event

allocate and free memory

System calls
File manipulation

create file, delete file

open, close

read, write, reposition

get file attributes, set file attributes

System calls
Device manipulation

request device, release device

read, write, reposition

get device attributes, set device attributes

logically attach or detach devices

System calls
Information maintenance

get time or date, set time or date

get system data, set system data

get process, file, or device attributes

set process, file, or device attributes

System calls
Communications

create, delete communication connection

send, receive messages

transfer status information

attach or detach remote devices

System structure
Simple approach

- MS-DOS – written to provide the most functionality in the least space
  - not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated

System structure
Simple approach

- Unix – limited by hardware functionality, the original Unix operating system had limited structuring. The Unix OS consists of two separable parts:
  - Systems programs
  - The kernel
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level.

System structure
Layered approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers.

System structure
Layered approach

- A layered design was first used in the THE operating system (Dijkstra, 1968). Its six layers are as follows:
  - layer 5: user programs
  - layer 4: buffering for input and output devices
  - layer 3: operator-console device driver
  - layer 2: memory management
  - layer 1: CPU scheduling
  - layer 0: hardware

System structure
Virtual machines

- A virtual machine takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware.
  - A virtual machine provides an interface identical to the underlying bare hardware.
  - The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.

System structure
Virtual machines

- The resources of the physical computer are shared to create the virtual machines.
  - CPU scheduling can create the appearance that users have their own processor.
  - Spooling and a file system can provide virtual card readers and virtual line printers.
  - A normal user time-sharing terminal serves as the virtual machine operator's console.
- *(Example, IBM VM)*

Virtual machines advantages and disadvantages

- The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources.
- A virtual-machine system is a perfect vehicle for operating-systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
- The virtual machine concept is difficult to implement due to the effort required to provide an exact duplicate of the underlying machine.

System design goals

- User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast.
- System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient.

Mechanism and policy

- Mechanisms determine how to do something; policies decide what will be done.
- The separation of **policy** from **mechanism** is a very important principle; it allows maximum flexibility if policy decisions are to be changed later.

System implementation

- OS used to be written exclusively in assembly language
- Now some are written in higher-level languages (e.g., C); assembly-language routines (e.g., for identified bottlenecks) provide speed for key functions
- Advantage: faster development, easier to understand and debug, easier to port
- Disadvantage: reduced speed and increased storage requirements

System Generation (SYSGEN)

- Configure for a particular machine in a class and/or for peripheral configurations
  - CPU to be used
  - Amount of memory available
  - Available devices
  - Operating system options desired, e.g., what job mix is expected, etc.
- *Bootstrap program* (*bootstrap loader*): stored in ROM; locates kernel, loads it into main memory; starts execution
- Alternately fetches more complex *boot* program and transfers control to it (two step process)