Chapter 2. Operating-System Structures

- System Components
- Operating System Services
- System Calls
- System Programs
- System Structure
- Virtual Machines

System Components

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

System Components - continued

- Process management
  - A ***process*** is a program in execution.
    - A process is a single executable and loadable module in execution, which can run concurrently with other executable modules. For example, in a multi-tasking environment that supports processes, like MS Windows, a word processor, an Internet browser, and a data base are separate executable modules and can run concurrently. Processes are separate executable and loadable modules in execution, as opposed to **threads** which are not loadable.
    - A process needs certain resources, including CPU time, memory, files, and I/O devices to accomplish its task.
  - The operating system is responsible for the following activities in connection with process management:
    - Process creation and deletion
    - Process suspension and resumption
    - Provision of mechanisms for:
      - Process synchronization
      - Process communication

System Components - continued

- Memory management
  - *Memory* is a large array of words or bytes, each with its own address.
    - It is a repository of quickly accessible data shared by the CPU and I/O devices.
  - *Main memory* is a volatile storage device. It loses its contents in the case of system failure.
  - The operating system is responsible for the following activities in connections with memory management:
    - 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.

System Components - continued

- I/O system management
  - The I/O system consists of:
    - A buffer-caching system
    - **Drivers** for specific hardware devices
    - A general device-driver interface

System Components - continued

- File management
  - A ***file:***
    - Is a collection of related information defined by its creator.
    - Commonly, files represent programs (both source and object forms) and data.
  - The operating system is responsible for the following activities in connections with file management:
    - File creation and deletion
    - Directory creation and deletion
    - Support of primitives for manipulating files and directories
    - Mapping files onto *secondary storage*
    - File backup on *stable (nonvolatile) storage* media

System Components - continued

- Secondary-storage management
  - Since main memory (*primary storage*) is volatile and too small to accommodate all data and programs permanently, the computer system must provide *secondary storage* to back up main memory.
  - Most modern computer systems use disks as the principle on-line storage medium, for both programs and data.
  - The operating system is responsible for the following activities in connection with disk management:
    - Free space management
    - Storage allocation
    - Disk scheduling

System Components - continued

- Networking (distributed systems)
  - A *distributed* system is a collection of processors that do not share memory or a clock over a network.
    - Each processor has its own local memory.
  - The processors in the system are connected through a communication network.
  - Communication takes place using a ***protocol.***
  - A distributed system provides user access to various system resources.
  - Access to a shared resource allows:
    - Computation speed-up
    - Increased data availability
    - Enhanced reliability

System Components - continued

- 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

System Components - continued

- Command-interpreter system
  - Many commands are given to the operating system by control statements which deal with:
    - Process creation and management
    - I/O handling
    - Secondary-storage management
    - Main-memory management
    - File-system access
    - Protection
    - Networking
  - The program that reads and interprets control statements is called variously:
    - *Command-line interpreter*
    - ***Shell*** *(in UNIX)*
    - Its function is to get and execute the next command statement.

System Components - continued

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

Operating System Services

- See Figure 2.1.
- 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
- 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 – cont.

Additional functions exist not for helping the user, but rather for ensuring efficient system operations:

- 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

System Calls

- ***System calls*** provide the interface between a running program and the operating system.
  - Generally available as assembly-language instructions
  - Languages defined to replace assembly language for systems programming allow system calls to be made directly (e.g., C, C++).
  - See Figures 2.4 – 2.6.

- 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 operating system

System Calls - continued

- Passing of parameters as a table
  - See Figure 2.7.

System Calls - continued

- Types of system calls
  - Process control
  - File management
  - Device management
  - Information maintenance
  - Communications
  - Protection

System Calls - continued

- Process control
  - End, abort
  - Load, execute
  - Create, terminate
  - Get process attributes, set process attributes
  - Wait for time
  - Wait event, signal event
  - Allocate and free memory

  - See Figure 2.10 MS-DOS execution.
  - See Figure 2.11 FreeBSD running multiple programs.

System Calls - continued

- File management
  - Create file, delete file
  - Open, close
  - Read, write, reposition
  - Get file attributes, set file attributes

System Calls - continued

- Device management
  - Request device, release device
  - Read, write, reposition
  - Get device attributes, set device attributes
  - Logically attach or detach devices

System Calls - continued

- 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 - continued

- Communications
  - Create, delete communication connection
  - Send, receive messages
  - Transfer status information
  - Attach or detach remote devices

  - Implementation
    - **Message passing** between two processes
    - **Shared memory** for two processes

System Programs

- ***System programs*** provide a convenient environment for program development and execution. They can be divided into:
  - File manipulation
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Application programs
- Most users' view of the operation system is defined by system programs, not the actual system calls.

System Structure

- 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.
  - See Figure 2.12.

System Structure - continued

- 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; and
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level.
  - See Figure 2.13.

System Structure - continued

- 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 layer uses the functions (operations) and services of only lower-level layers
  - See Figure 2.14.

System Structure - continued

- Microkernels
  - Moves as much from the kernel into "*user*" space.
  - Communication takes place between user modules using message passing
  - Benefits:
    - Easier to extend a microkernel
    - Easier to port the operating system to new architectures
    - More reliable (less code is running in kernel mode)
    - More secure
  - Detriments:
    - Performance overhead of user space to kernel space communication
  - Examples: Mach

System Structure - continued

- **Modules**
  - Most modern operating systems implement kernel *modules*.
    - Uses object-oriented approach.
    - Each core component is separate.
    - Each talks to the others over known *interfaces*.
    - Each is *loadable* as needed within the kernel.
  - Overall, similar to layers but with more flexible
  - See Figure 2.15.
- Hybrid structure
  - See Figure 2.16.

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.

- See Figure 2.17.

Virtual Machines - continued

- 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.

- ☺ *Anybody uses a virtual machine? VirtualBox?*
- See Figure 2.18, "VMware architecture".

Advantages/Disadvantages of Virtual Machines

- 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 to the underlying machine.

Java Virtual Machine

- Compiled Java programs are platform-neutral byte codes executed by a Java Virtual Machine (JVM)
- JVM consists of:
  - Class loader
  - Class verifier
  - Runtime interpreter
- Just-In-Time (JIT) compilers increase performance

- See Figure 2.20.