COSC4740-01 Operating Systems Design, Fall 2001, Byunggu Yu

Chapter 3 Operating System Structures

1. OS Components

- Process management
- I/O management
- Main Memory management
- File & Storage Management
- Protection
- Networking
- Protection
- Command Interpreter

1.1 Process Management

- Process (or job): A program or a fraction of a program that is loaded in main memory.

- Motivation: We do not need a whole program code at once. To process an instruction, CPU fetches and executes one instruction of a process after another (i.e., the execution of a process progresses in a sequential fashion) in main memory.

- Tasks of Process Management of an OS:
    - Create, load, execute, suspend, resume, and terminate processes
    - Switch system among multiple processes in main memory (process scheduling)
    - Provides communication mechanisms so that processes can send (or receive) data to (or from) each other (process communication).
    - Control concurrent* access to shared data to keep shared data consistent (process synchronization).
    - Allocate/de-allocate resources properly to prevent or avoid deadlock situation**

  *: In time-shared systems, even if the system is equipped with a single CPU, we usually say that the processes in main memory are concurrently running.

  **: Deadlock situation: e.g., There are three running processes A, B, and C. A is waiting for the resource of B. B is waiting for the resource of C. C is waiting for the resource of A. At the same time they want to keep using their current resources. As a result, all processes will wait forever.
  Note, Inter-Process Communication will be discussed in Chapter 4!

1.2 I/O Management

- Motivations:
  - Provide an abstract level of H/W devices and keep the details from applications to ensure proper use of devices, to prevent errors, and to provide users with convenient and efficient programming environment.

- Tasks of I/O Management of OS:
  - Hide the details of H/W devices
  - Manage main memory for the devices using cache, buffer, and spooling
  - Maintain and provide device driver interfaces

1.3 Main Memory management

Processes must be loaded into main memory to be executed.

- Motivations:
  - Increase system performance by increasing "hit" ratio (e.g., optimum: when CPU read data or instruction, it is in the main memory always)
  - Maximize memory utilization

- Tasks of Main Memory Management of OS:
  - Keep track of which memory area is used by whom.
  - Allocate/de-allocated memory as need

1.4 File & Storage Management

- Motivation:
  - Almost everything is stored in the secondary storage. Therefore, secondary storage accesses must be efficient (i.e., performance) and convenient (i.e., easy to program I/O function in application level)
  - Important data are duplicated and/or stored in tertiary storage.

- Tasks of File Management
  - Create, manipulate, delete files and directories

- Tasks of Storage Management
  - Allocate, de-allocate, and defrag blocks[1]
  - Bad block marking
  - Scheduling for multiple I/O request to optimize the performance

1.5 Networking

[1] "Block" is the unit of data transfer of the storage device. If the block size is 1Kbyte, each access to the device will read or write 1Kbyte. Thus, for example, to read a whole file, whose size is 1024Kbyte, we should access the hard drive 1024 times. Each access requires seek-time to move I/O head to the destination cylinder, rotation-time to find the sector (block), and transfer-time to fill the device buffer with the data in the block.

Allow communications between computers (more important for Client/Server OS and Distributed OS).

1.6 Protection

Protect hardware resources, Kernel code, processes, files, and data from erroneous programs and malicious programs.

1.7 Command Interpreter

Command Interpreter is one of the most important system programs[2]. Because almost every OS provide system programs, some people argue that command interpreter is a part of OS.

- Motivation:
  - Allow human users to interact with OS
  - Provide convenient programming environment to users

- Tasks:
  - Execute a user command by calling one or more number of underlying system programs or system calls

- Examples:
  - Windows DOS command window
  - Bash of Unix/Linux
  - CSHELL of Unix/Linux
  - …

2. OS Architectures

- Monolithic (The big mass)
  - OS is simply a collection of functions.
  - One very big OS including everything (system calls, system programs, every managers, device drivers, etc)
  - Entire OS resides in main memory
  - Pros:
    - High performance
    - Easy to implement
    - Difficult to debug, modify, and upgrade
    - Poor security, protection, and stability of OS

[2] An abstracted level of underlying system calls (e.g., ls, mv, ps, cp, date, rm, …). For example, rm command that you type at the shell prompt is a system program that contains the system call *unlink()*.

- Layered: OS is still large single program but internally broken up into layers providing different functionalities. A layer can only use the services provided by layer in a level below.

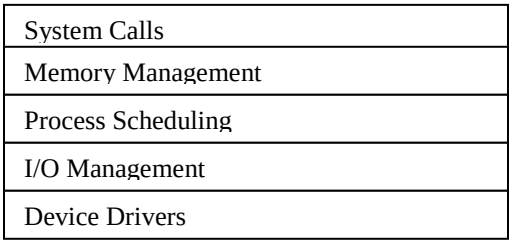

Figure 3.1 Example of Layered System

  - Information hiding between layers → Increased security and protection
  - Easy to debug, test, and modify OS
  - If one layer stops working, entire system will stop
  - Mapping overhead between layers
  - Difficult to categorize into layers

- Modular/Kernel based: OS is made up of several separated/independent modules (kernel, FS, MM, etc), and only kernel resides in main memory. Kernel calls other modules as needed to perform certain tasks.

- Virtual Machine: Allow multiple different OSs to run on a single machine. VM gives an illusion that the machine is at entire disposal of OS by providing each OS an abstract machine.
  - Pros:
    - Solve machine-OS-software compatibility issues.
    - Stable and Good for OS development and testing (If one OS fails, the other OS can continue to run)
  - Cons:
    - Complex to develop/implement
    - Poor performance (abstract layer, mapping overhead between layers, …)

- Client-Server: OS consists of servers, clients, and kernel. Kernel is very tiny and handles only the communication between the clients and servers. Both clients and servers run in user mode.
  - Pros:
    - Stable (even if a server fails, the OS can still operate)
    - Small kernel
  - Cons:
    - Complex to develop/implement (servers run in user mode and can access resources through the kernel)

        - Security problem: servers reside in user space.

3. Design Goal of OS

- Efficiency and Convenience
- Keep kernel adequately small
    - Too much functionality in kernel → low flexibility at higher level
    - Too little functionality in kernel → low functional support at higher level
- Maximize the utilization, performance, and functionality of the underlying H/W resources
    - OS design is affected by H/W properties
    - OS design is affected by the type of system to design (e.g., Multiprogramming, Multi-tasking, Real-time, Distributed, .etc)

4. System Generation (Installing OS)

- We should configure OS for the underlying hardware when we install it (i.e., installing OS)

- Approaches:
    - Modify the source code of OS and recompile it on the target machine
        - Pros: small OS size, We can tune the details for the H/W
        - Cons: Very difficult, less flexible for H/W upgrade
    - Provide many pre-compiled modules (e.g., five different memory manager object files). When we install OS, select certain object modules and link them together accordingly.
        - Pros: easy to configure, flexible for H/W upgrade
        - Cons: big OS size, We cannot tune the very details for the H/W