Preliminary

Handouts:

- Course Syllabus
- Homework Grading Policy
- Project 1
- Course Logistics

Introduction

What Is an Operating System?

An operating system (OS) is a set of procedures that provides two basic functions:

1. allocate *resources* to *processes* (resource manager), and
2. hide the details of the physical machine and provide a more pleasant *virtual machine* (abstract the resources of the physical machine).

We call these the *resource* and *beautification* principles respectively.

Resource is a commodity needed to get work done.

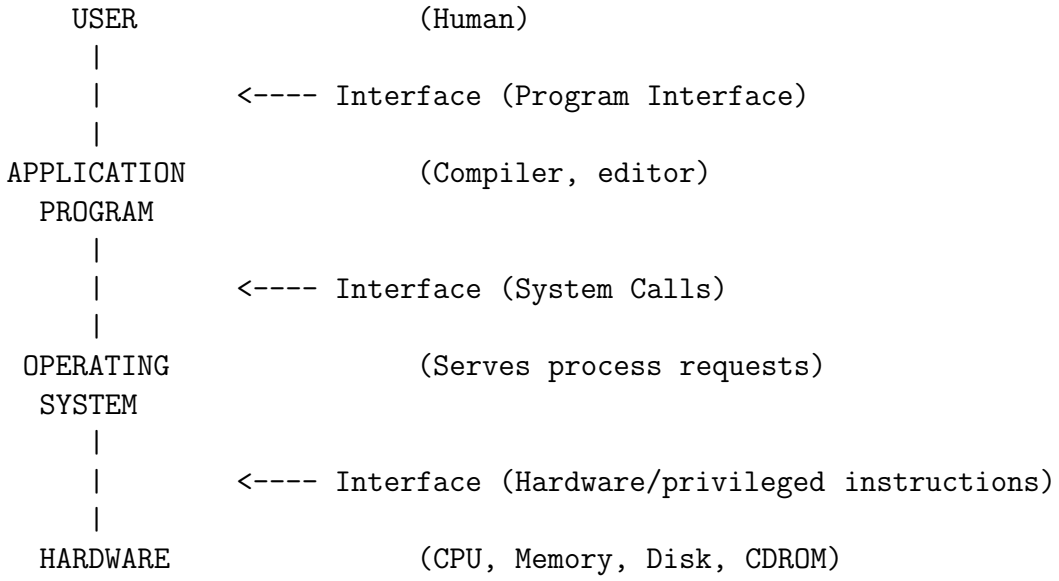
Actual Operating Systems

What are operating systems we know:

- Unix—many flavors
- Linux—Unix for the masses (FreeBSD, ...)
- Windows NT/2000/XP
- others, Mac OS, Palm OS, MS-DOS, BeOS, Multics, Digital VMS, Minix, Xinu, OS/2, Mach, ...

Layers of a System

Also look at picture from Tanenbaum's book.



Processes

Processes are hard to define precisely and exact definitions are usually operating system specific.

A *program* is a set of *data* and *code* that manipulates it.

A *process* is a program in *execution*. When a CPU executes a program, the operating system allocates space for data, space for a stack, and space for local variables. Processes need resources—CPU, disk, transput (input/output).

Resources

Operating System allocates and manages resources needed by processes.

What resources does the OS manage?

- time (e.g., CPU time, device time)
- space (e.g., memory)
- other physical devices (e.g., network interface)
- new resources defined by the operating system (e.g., files, swap space, processes)

For example, if a printer is not managed then get garbled output.

Services Provided by an Operating System

- context switching and scheduling (allocating CPU time to processes) (process creation tree)
- memory management (allocating space for instructions, data, etc.)
- interprocess communication (IPC) (facilities that allow concurrent processes to communicate with each other)
- access to a file system (picture)
- high level input/output (not to a device), is device independent

What an Operating System is not?

- hardware
- programming language
- command interpreter
- browsers
- applications such as editors, mailers, compilers, and loaders. Applications use the facilities provided by the operating system.

The Design Process

- *philosophies* define an overall strategy (fairness). The operating system should
  - *make efficient use of resources*. Resources should be used as much as possible and the operating system should finish as many processes as possible. However, the operating system should provide:
  - *make fair use of resources* and let each process have the resources it needs.

  It is a fundamental point that the two goals of an operating system are often contradictory.
- *policies* chose how activities are to proceed (FCFS scheduling, highest priority for super user jobs, etc.)
- *mechanisms* enforce policies. The nuts and bolts of the OS.

We'll study policies and mechanisms. Whenever possible, the operating system designer should build mechanism, rather than policy. Once a policy is placed in the implementation, it cannot easily be changed. Leaving policy in the hands of the user or system administrator provides the greatest flexibility. It is important to study both policy and mechanism, because there is a cost associated with carrying out specific policies.

The operating system designer wants to implement mechanism and allow policy to be flexible.

History

How did operating systems evolve?

(more details in Tanenbaum)

1. *Single program execution.* (no Operating System) In early days, machines didn't have operating systems; programs took over the entire machine. Programs, written in assembly language (or worse), had to know such low-level details such as how to perform input/output. Punched on paper tape and then computer cards. *However, early programs spent most of their time performing the input/output needed to read and print the job!*

2. *Batch (Spooling).* To make better use of resources, spooling systems were developed that copied jobs from card readers to tape, so that once the job could be run, the CPU could fetch it from tape rather than from the (relatively) slow card reader. Later spooled onto disk.

   Interrupts. Introduced the notion of interrupts so devices could alert the CPU when they were finished. However, running just one job by itself was still inefficient.

3. *Multiprogramming.* The next step in operating systems development, *multiprogramming*, gave the illusion of executing several programs simultaneously. Multiprogramming systems execute each job for several thousands of a second providing the appearance of running several jobs simultaneously. Thus, notion of a *process*. Still compute bound jobs could monopolize the CPU.

   Time-slicing. In *time-sharing* systems, all processes are scheduled to run in round-robin fashion.

   One of the systems was MULTICS at M.I.T. Was a predecessor of the UNIX operating system done at Bell Labs. 1970ish.

4. *Personal Computers.* MS-DOS did not allow multiprogramming. Windows Operating Systems now use it.

5. **nix for the Masses.* Linux, FreeBSD, ...

6. *Distributed and Network Operating Systems.* Spans multiple machines. Distributed is transparent of machines, N.O.S. is not.

7. *Specialized Operating Systems.*

   - server operating systems (Web), use of Unix/Linux
   - real-time operating systems, performance constraints—hard and soft.
   - embedded operating systems, Personal Data Assistants (PDAs). Examples are PalmOS and Windows CE (Consumer Electronics)
   - smart card operating systems, credit-card size with a CPU chip

Hardware Review

Look at Figure 1-5.

Will look at relevant details of devices.

Hierarchy of storage size/speed.

Look at textbook for more details.

Operating System Structure

What does the operating system look like? How is it organized?

- *Simple Systems*. Operating system not separated from applications. No protection. Example is MS-DOS. Embedded systems.
- *Monolithic Systems*. The big mess. Unix (Linux). See pictures. Talk about the kernel as the privileged portion of the O.S.
- *Layered Systems*. XINU as an example. Also MINIX (motivated Linux).
- *Microkernel*. Clients obtain service by sending messages to server processes. (also called client-server). Simple approach where simple kernel mechanism allows policies to be implemented in server processes.
- *Virtual Machines*. Emulated machine environment for application. IBM VM/370 environment. Java Virtual machine. Vmware, Xen, Linux Vservers, Solaris Zones, Virtuozzo.

A trend towards this latter type of organization. More amenable for moving towards a distributed system.

Build minimum *mechanism* into the kernel and leave *policy* decisions to the servers.

Issues of flexibility vs. performance. Windows NT uses partial microkernel approach.

Operating Systems for Study

- Unix— Developed late 70s. traditional, monolithic multitasking O.S.
- Linux— "free" on a wide range of platforms. Open source software.
  - 1991. Developed by Linus Torvalds for 80386 processor. evolved from Minix
  - v1.0 in 1994 supporting networking.
- Windows NT/2000/XP— real O.S. from Microsoft supporting multitasking.