I – M.Sc IT

MODERN OPERATING SYSTEMS

UNIT – I

Prepared By

Dr. T. Christopher

UNIT–I

- **Introduction – History of Operating systems –– Computer Hardware review – Operating** System Concepts – System calls – Processes – Model – Creation – Termination – Process Hierarchy – Process States – Implementation of Processes.

What Is An Operating System

A modern computer consists of:

- One or more processors
- Main memory
- Disks
- Printers
- Various input/output devices

Managing all these components requires a layer of software – the **operating system**

What is an Operating System?

- A modern computer is very complex.
  - Networking
  - Disks
  - Video/audio card
  - ….
- It is impossible for every application programmer to understand every detail
- A layer of computer software is introduced to provide a better, simpler, cleaner model of the resources and manage them

What Is An Operating System

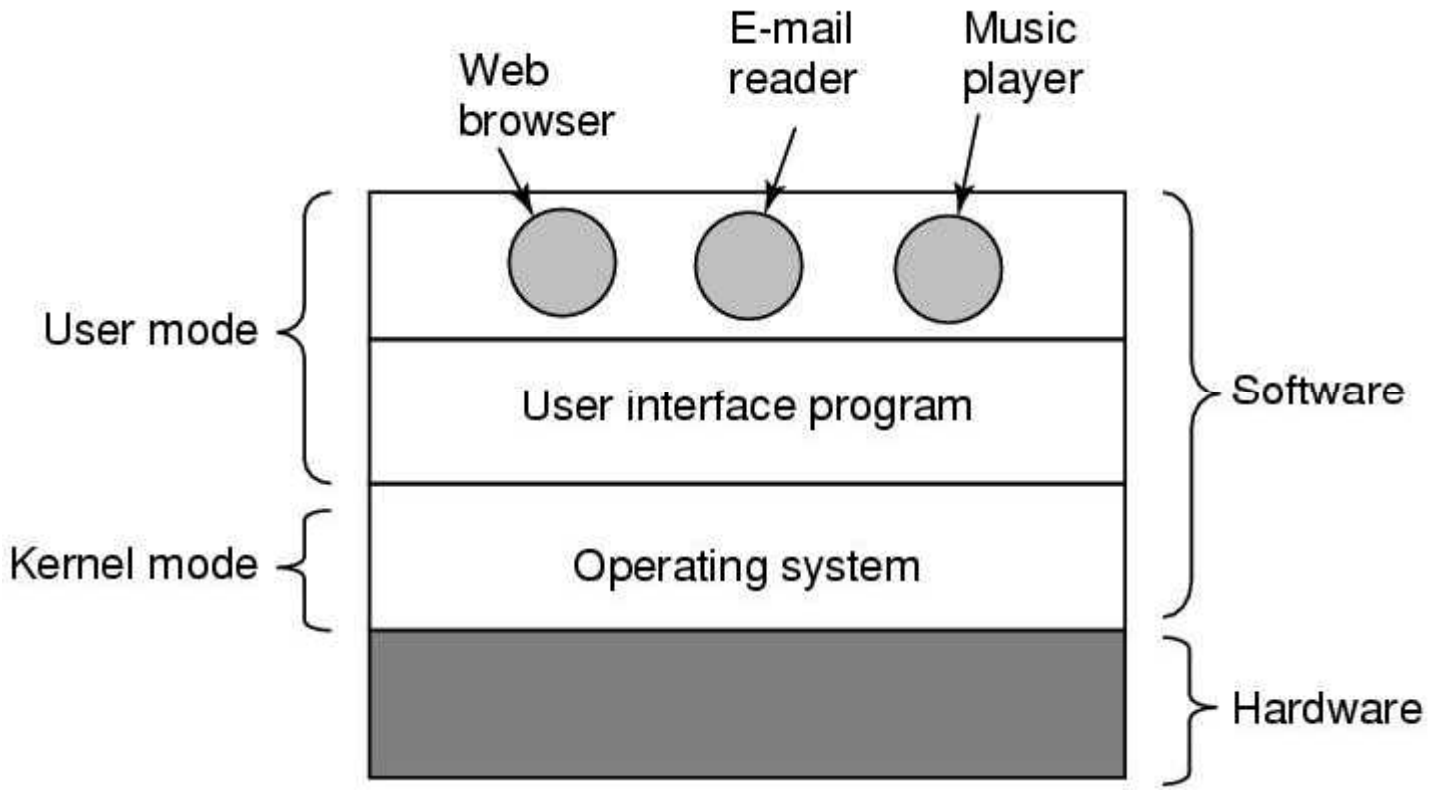


Figure 1-1. Where the operating system fits in.

What is an Operating System?

- Users use various OS
  - Windows, Linux, Mac OS etc.
- User interacts with shell or GUI
  - part of OS?
  - they use OS to get their work done
- Is device driver part of OS?

What is an Operating System?

- On top of hardware is OS
- Most computers have two modes of operation:
  - Kernel mode and user mode
  - OS runs in **kernel mode,** which has complete access to all hardware and can execute any instruction
  - Rest of software runs in user mode, which has limited capability
  - Shell or GUI is the lowest level of user mode software

What is an operating system?

- Two functions:
  - From top to down: provide application programmers a clean abstract set of resources instead of hardware ones
  - From down to top: Manage these hardware resources

The Operating System as an Extended Machine

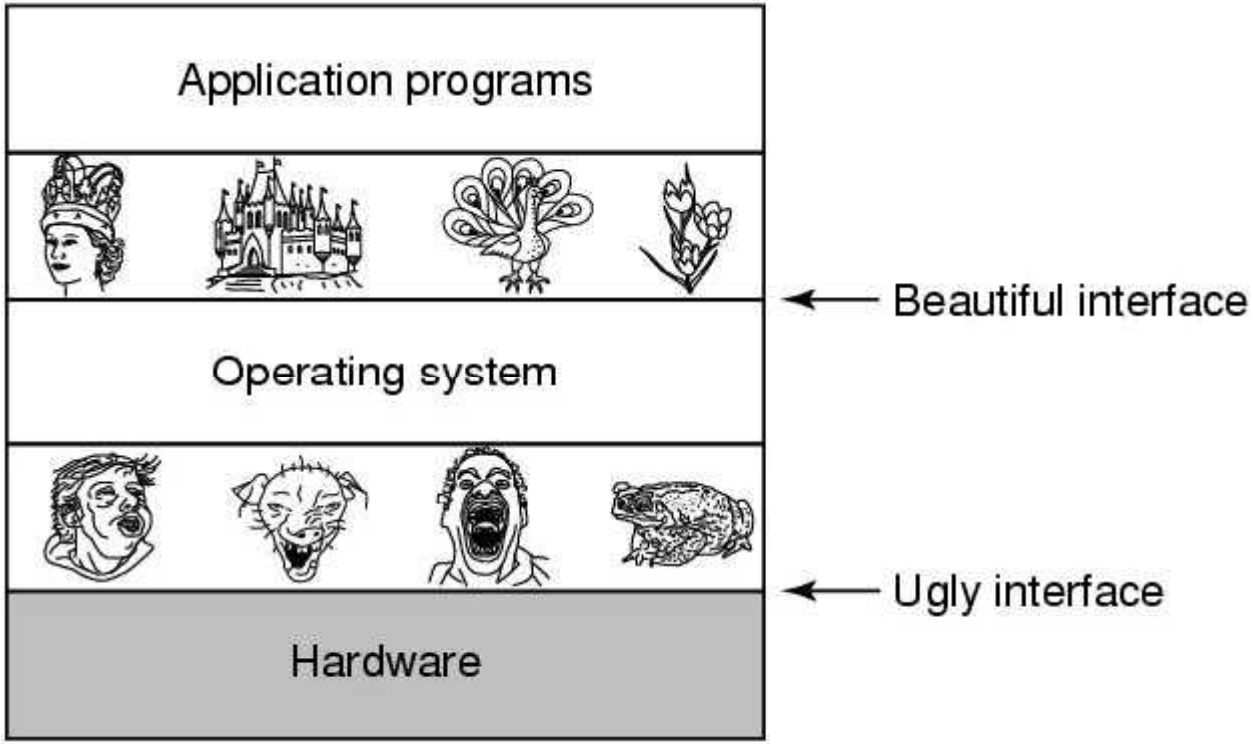


Figure 1-2. Operating systems turn ugly hardware into beautiful abstractions.

As an extended machine

- Abstraction:
  - CPU—process
  - Storage –- files
  - Memory– address space
- 4 types of people:
  - Industrial engineer: design hardware
  - Kernel designer
  - Application programmer: OS’s user
  - End users

The Operating System as a Resource Manager

- Allow multiple programs to run at the same time
- Manage and protect memory, I/O devices, and other resources
- Includes multiplexing (sharing) resources in two different ways:
  - In time
  - In space

As a resource manager

- Modern OS runs multiple programs of multiple users at the same time
  - Imagine what would happen if several programs want to print at the same time?
  - How to account the resource usage of each process?
  - Resources can be multiplexed:
    - How to ensure fairness and efficiency?

summary

- Operating system is a software
  - Is a complex software
  - Runs in kernel mode
  - Manages hardware
  - Provide a friendly interface for application programmer

History of Operating Systems

Generations:

- (1945–55) Vacuum Tubes
- (1955–65) Transistors and Batch Systems
- (1965–1980) ICs and Multiprogramming
- (1980–Present) Personal Computers

1st: vacuum tubes

- Large and slow
- Engineers design, build, operate and maintain the computer
- All programming is done with machine language, or by wiring circuits using cables
- insert plugboards into the computer and operate
- The work is mainly numerical calculations

2nd: transistors and batch systems

- Also called mainframes
- Computers are managed by professional operators
- Programmers use punch card to run programs; operators operate (load compiler, etc) and collect output to the user
- Complains soon come:
  - Human Operation between computer operation
  - Lead to batch system
  - Collect a batch of jobs in the input room, then read them into a magnetic tape; the same for output

2nd: Transistors and Batch Systems (1)

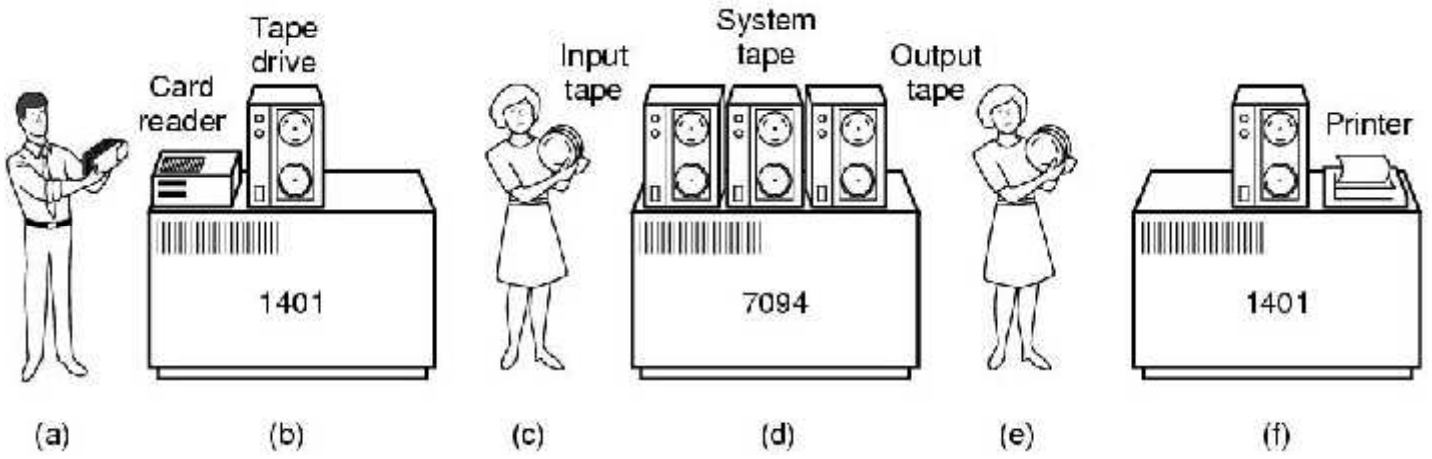


Figure 1-3. An early batch system.
(a) Programmers bring cards to 1401.
(b)1401 reads batch of jobs onto tape.

Transistors and Batch Systems (2)

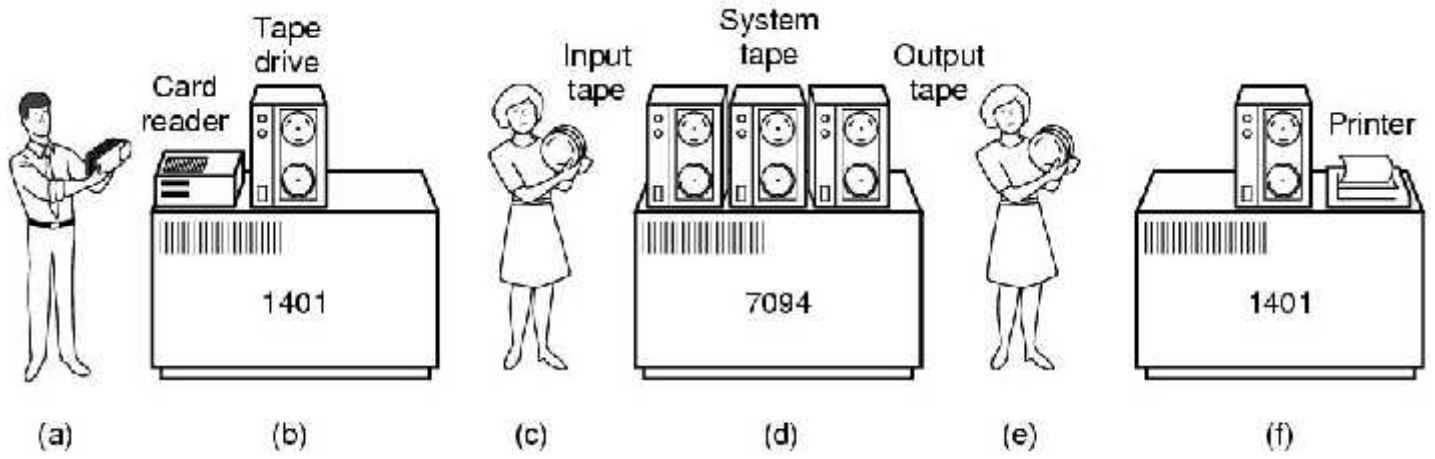


Figure 1-3. (c) Operator carries input tape to 7094. (d) 7094 does computing. (e) Operator carries output tape to 1401. (f) 1401 prints output.

Transistors and Batch Systems (4)

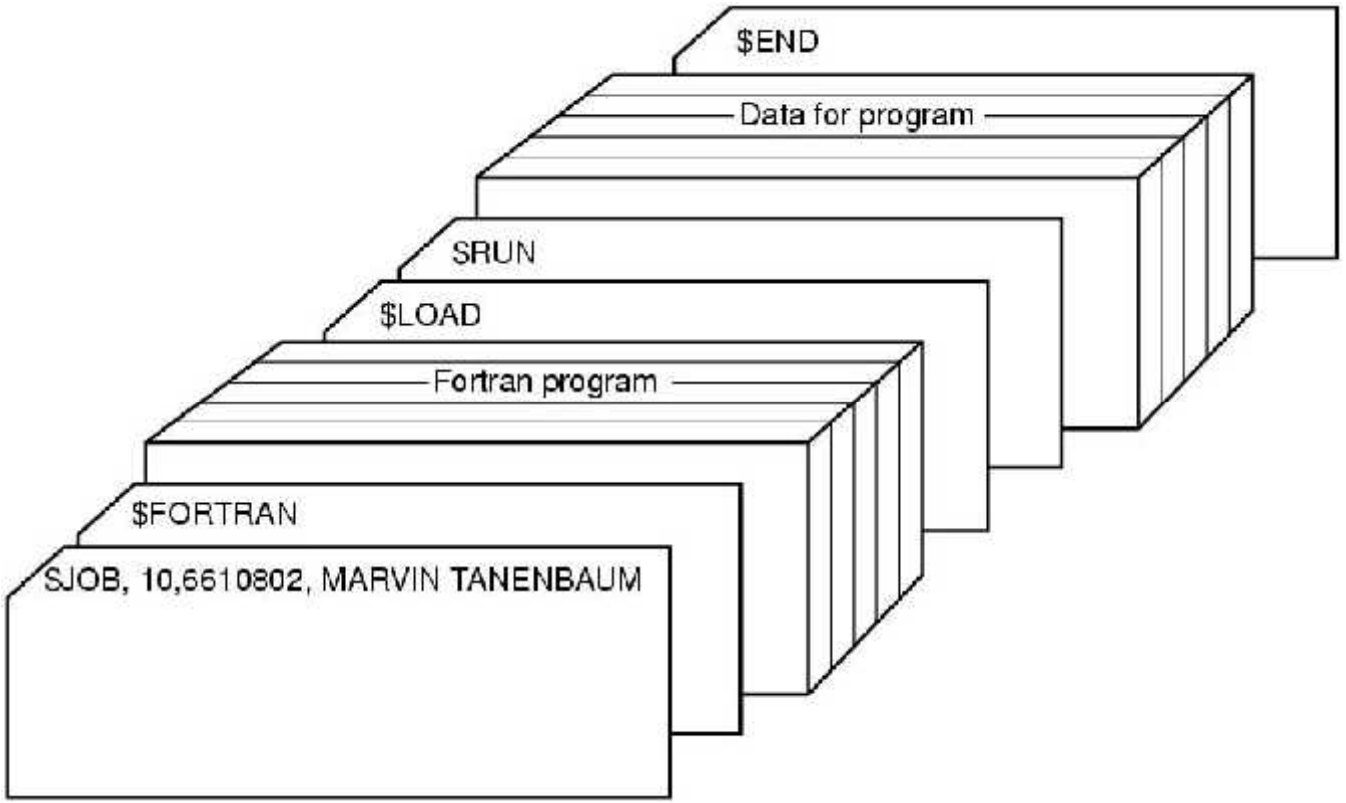


Figure 1-4. Structure of a typical FMS job.

3rd: IC and Multiprogramming

- OS/360:
  - introduces several key techniques
    - Multi-programming: solve the problem of CPU idling
    - Spooling: simultaneous peripheral operation on line
      - Whenever a job finishes, OS load a new job from disk to the empty-partition

3rd: ICs and Multiprogramming

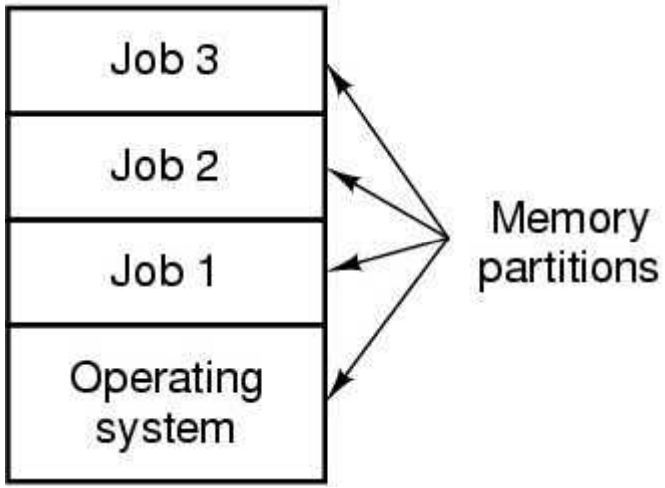


Figure 1-5. A multiprogramming system with three jobs in memory.

3rd: Ics and Multiprogramming

- Problems:
  - 3rd generation OS was well suited for big scientific calculations and massive data processing
  - But many programmers complain a lot… for not be able to debug quickly…. Why?
  - And the solution to this problem would be….?
  - Timesharing:
    - A variant of multiprogramming
    - Provide both fast interactive service but also fits big batch work

3rd: IC and Multiprogramming

- A system to be remembered: MULTICS
  - A machine that would support hundreds of simultaneous timesharing users– like the electricity system (like a web server nowadays)
  - Introduces many brilliant ideas but enjoys no commercial success
  - Its step-child is the well-known and time-honored UNIX
  - System V/ FreeBSD, MINIX, Linux

4th: personal computers

- Computers have performance similar to 3rd generation, but prices drastically different
- CP/M
  - First disk-based OS
- 1980, IBM PC, Basic Interpreter, DOS, MS-DOS
- GUI--Lisa—Apple: user friendly
- MS-DOS with GUI– Win95/98/me—winNT/xp…

Summary

- 4 generations OS
- Develops with hardware and user needs
- Multi-user, multi-programming, time-sharing

Computer Hardware Review

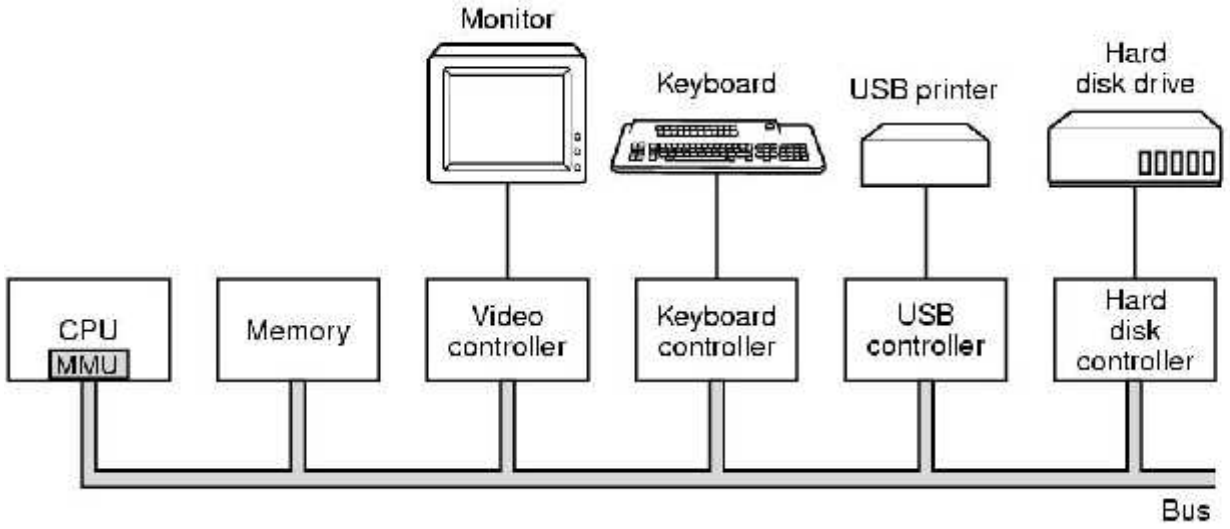


Figure 1-6. Some of the components of a simple personal computer.

Hardware: processor

- Brain of computer
  - Fetches instruction from memory and execute
  - Cycle of CPU:
    - fetch, decode, execute
  - CPU has registers to store variable and temporary result: load from memory to register; store from register to memory
  - Program counter: next instruction to fetch
  - Stack pointer: the top of the current stack
  - PSW: program status word, priority, mode…

A reference

CPU Pipelining

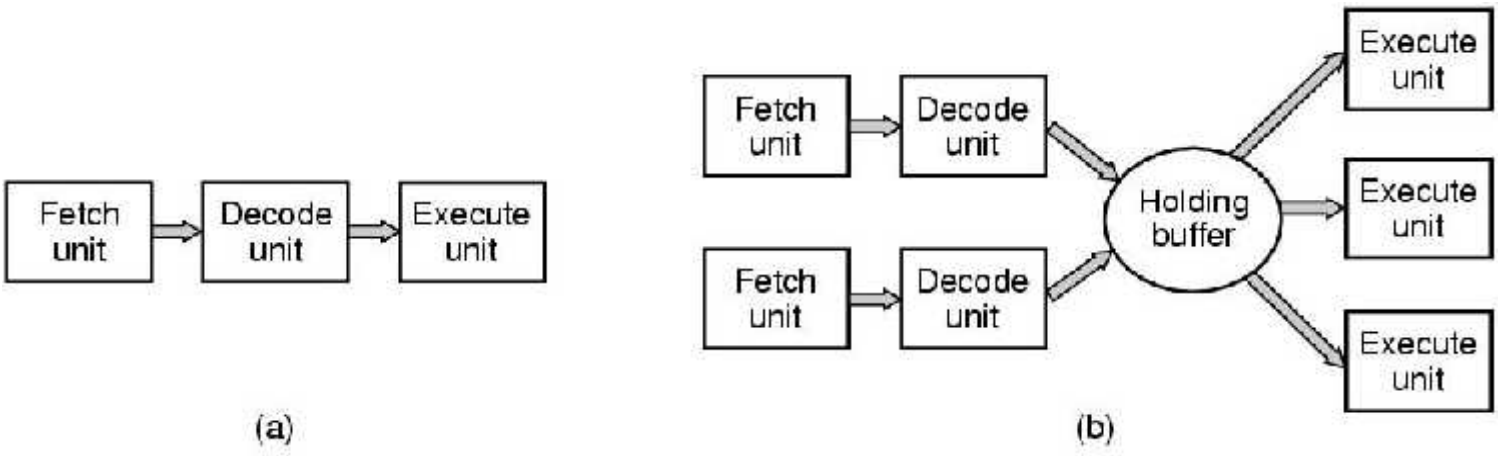


Figure 1-7. (a) A three-stage pipeline. (b) A superscalar CPU.

CPU Pipeline

- Accelerate the execution
- Cause headaches to OS/compiler writers

- For superscalar: instructions are often executed out of order

Multithreaded and Multicore Chips

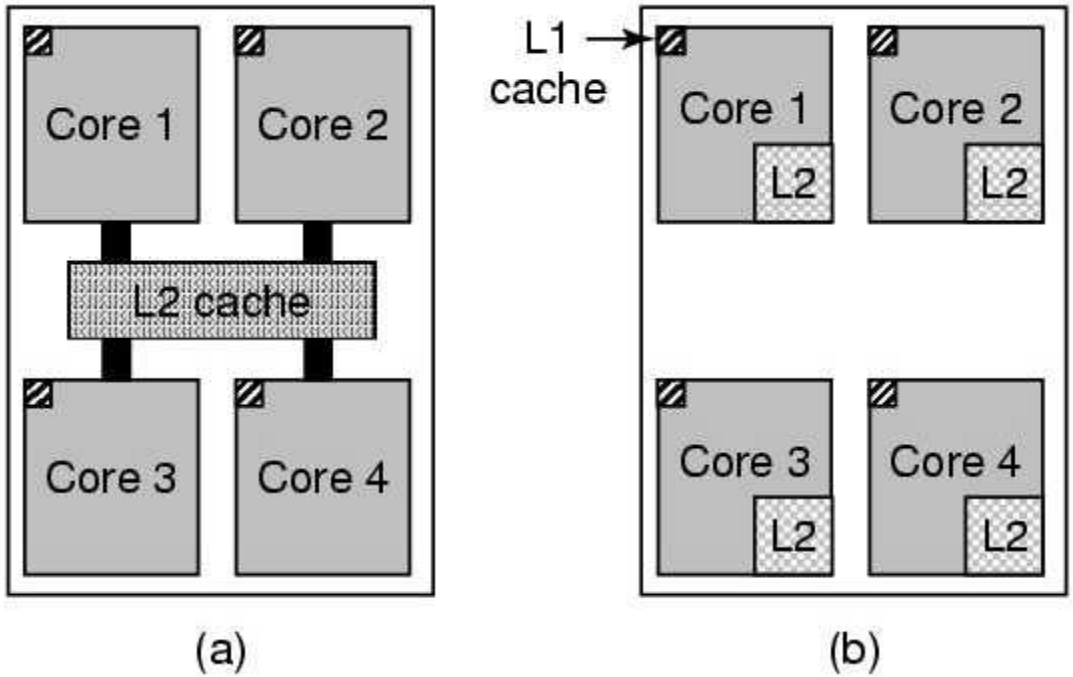


Figure 1-8. (a) A quad-core chip with a shared L2 cache. (b) A quad-core chip with separate L2 caches.

Memory (1)

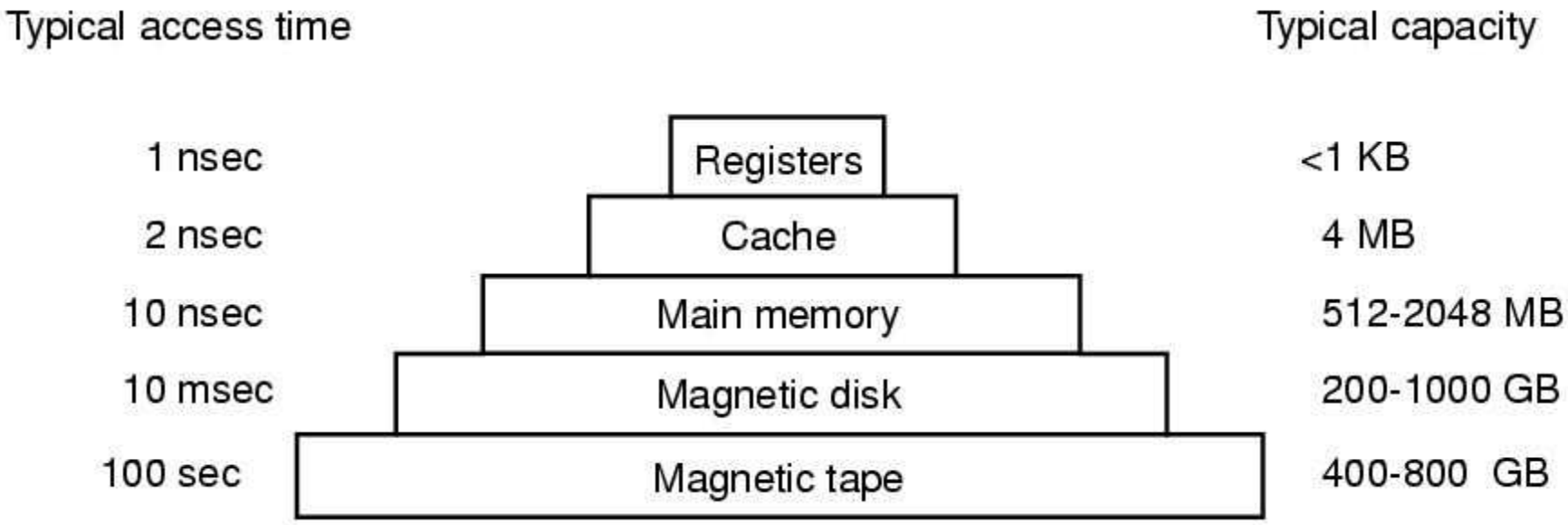


Figure 1-9. A typical memory hierarchy.
The numbers are very rough approximations.

memory

- Memory is where computer fetch and store data, ideally, it should be both chip/large
- The best people can do, is to construct a memory hierarchy
- Cache lines:
  - Memory divided into cache lines; the mostly used ones are stored in caches
  - Cache hit, cache miss
  - Cache: whenever there is disparity in usage or speed; used to improve performance

Memory (2)

Questions when dealing with cache:

- When to put a new item into the cache.
- Which cache line to put the new item in.
- Which item to remove from the cache when a slot is needed.
- Where to put a newly evicted item in the larger memory.

Disks

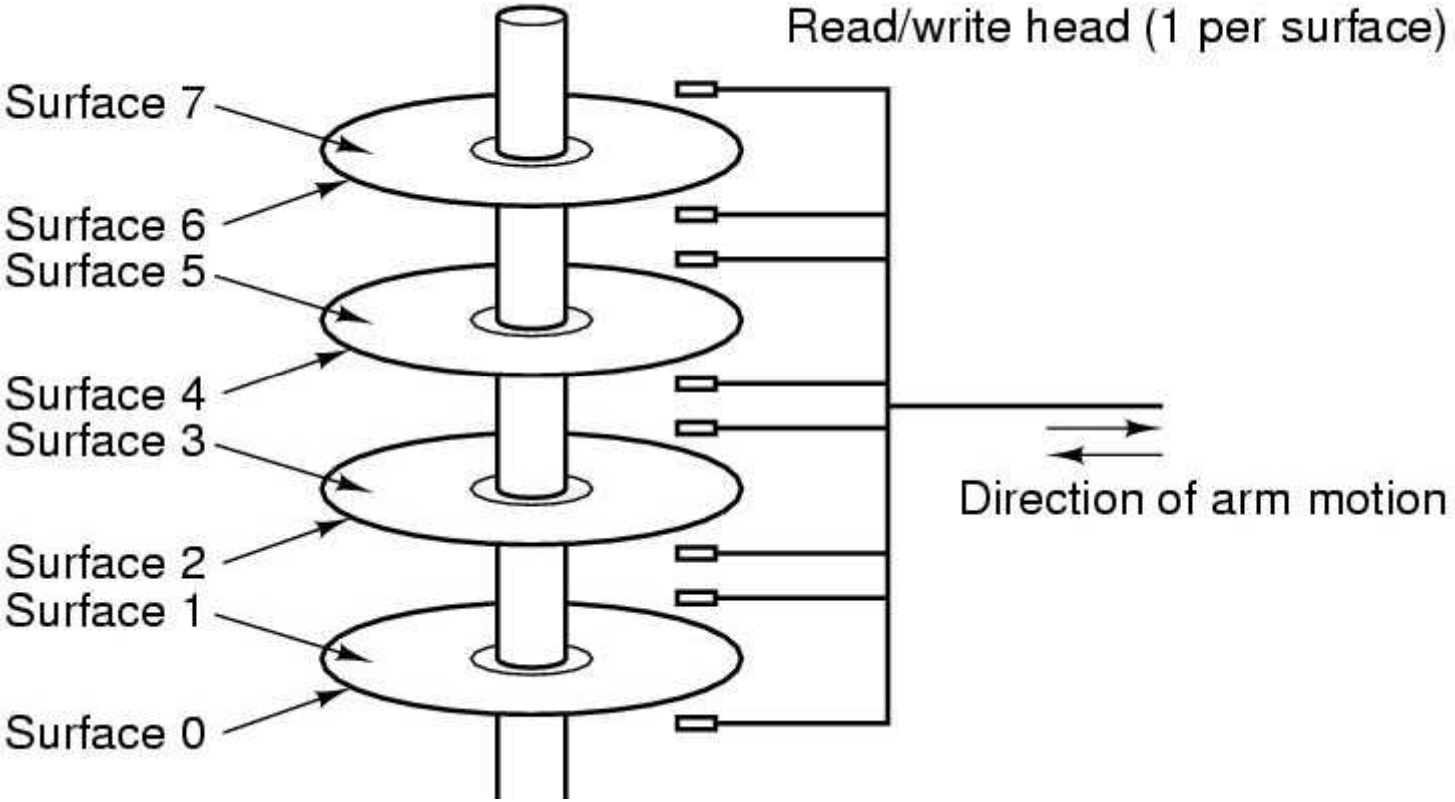


Figure 1-10. Structure of a disk drive.

Disks

- Cheap and large: two orders better than RAM
- Slow: three orders worse than RAM
- Mechanical movement to fetch data
- One or more platter—rotate– rpm
- Information is stored on concentric circles
- Arm, track, cylinder, sector
- Disk helps to implement Virtual Memory
  - When no enough memory is available, disks are used as the storage, and memory as cache

Booting the Computer

- BIOS—basic input/output system
  - On the parentboard, low-level I/O software
  - Checks RAM, keyboard and other basic devices
  - Determine the boot device: floppy, CD-ROM, disk
  - First sector of the boot-device is read into memory
  - The sector contains program to check which partition is active
  - Then a secondary boot-loader is read into memory and reads in operating system from the active partition

Operating System Concepts

- Processes
- Address spaces
- Files
- Input/Output
- Protection
- The shell
- Ontogeny recapitulates phylogeny
  - Large memories
  - Protection hardware
  - Disks
  - Virtual memory

processes

- Process
  - Address space: 0-4G; executable program, program's data, and its stack
  - Other resources like: registers, files, alarms, related processes, and other information
  - A process is fundamentally a container that holds information for a program to run

Processes

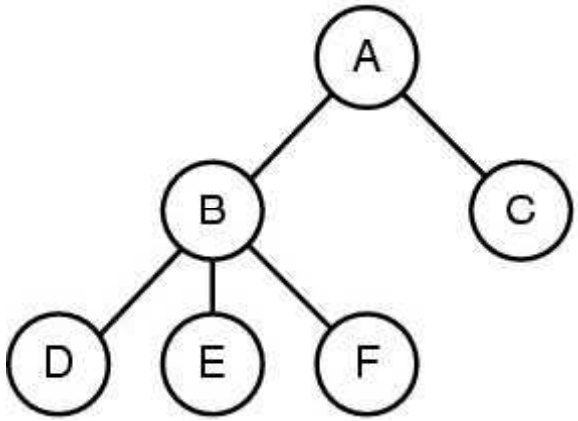


Figure 1-13. A process tree. Process A created two child processes, B and C. Process B created three child processes, D, E, and F.

Address Space

- The memory used by a process, in concept
- MOS allows multiple processes in memory simultaneously
- Some processes need more memory than physically available– virtual memory

Files (1)

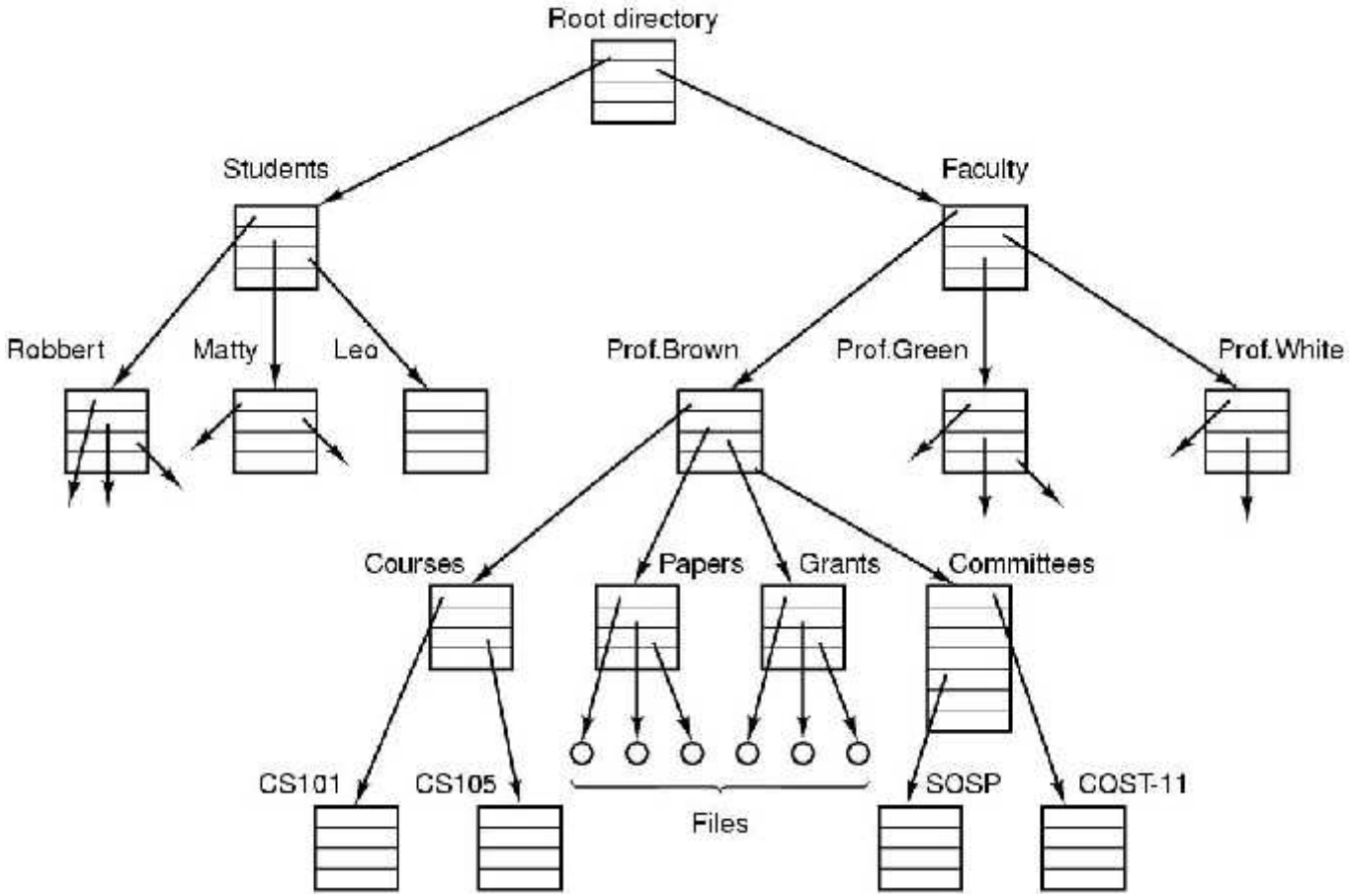


Figure 1-14. A file system for a university department.

Files (2)

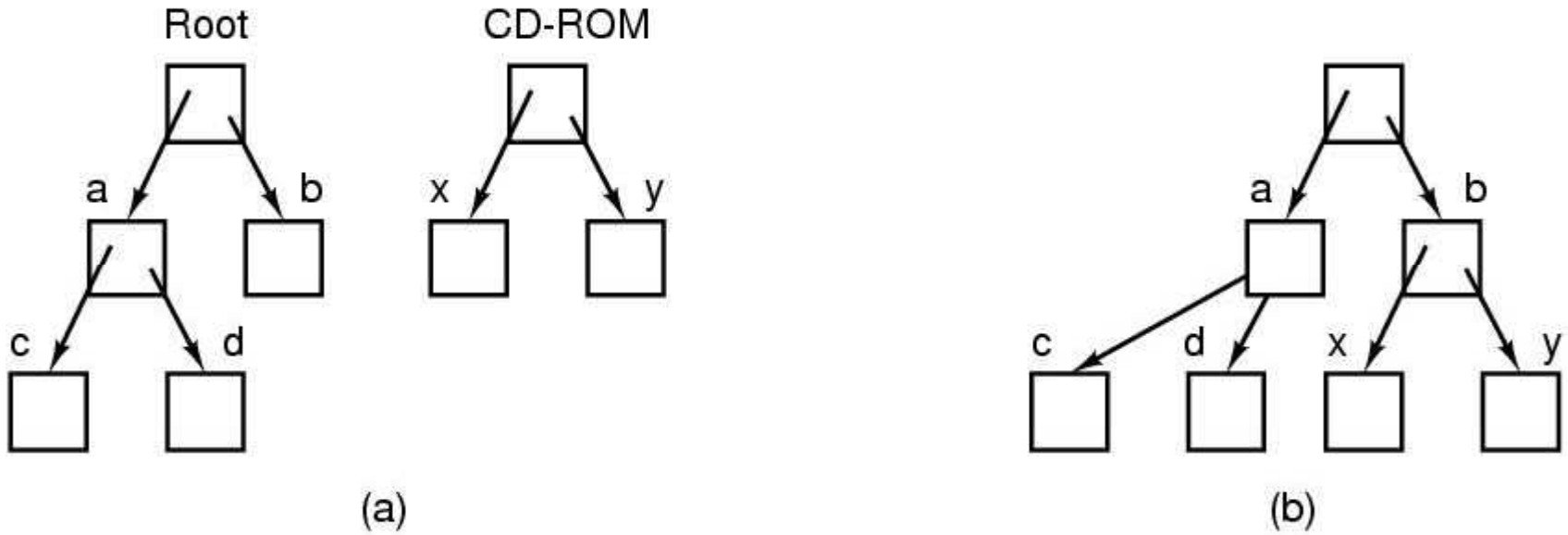


Figure 1-15. (a) Before mounting, the files on the CD-ROM are not accessible. (b) After mounting, they are part of the file hierarchy.

Files (3)

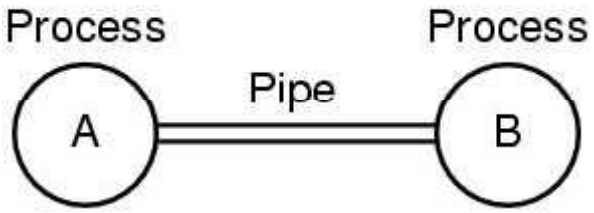


Figure 1-16. Two processes connected by a pipe.

System calls

- System calls is the interface users contact with OS and hardware
- System calls vary from system to system, but the underlying concepts are similar

System Calls

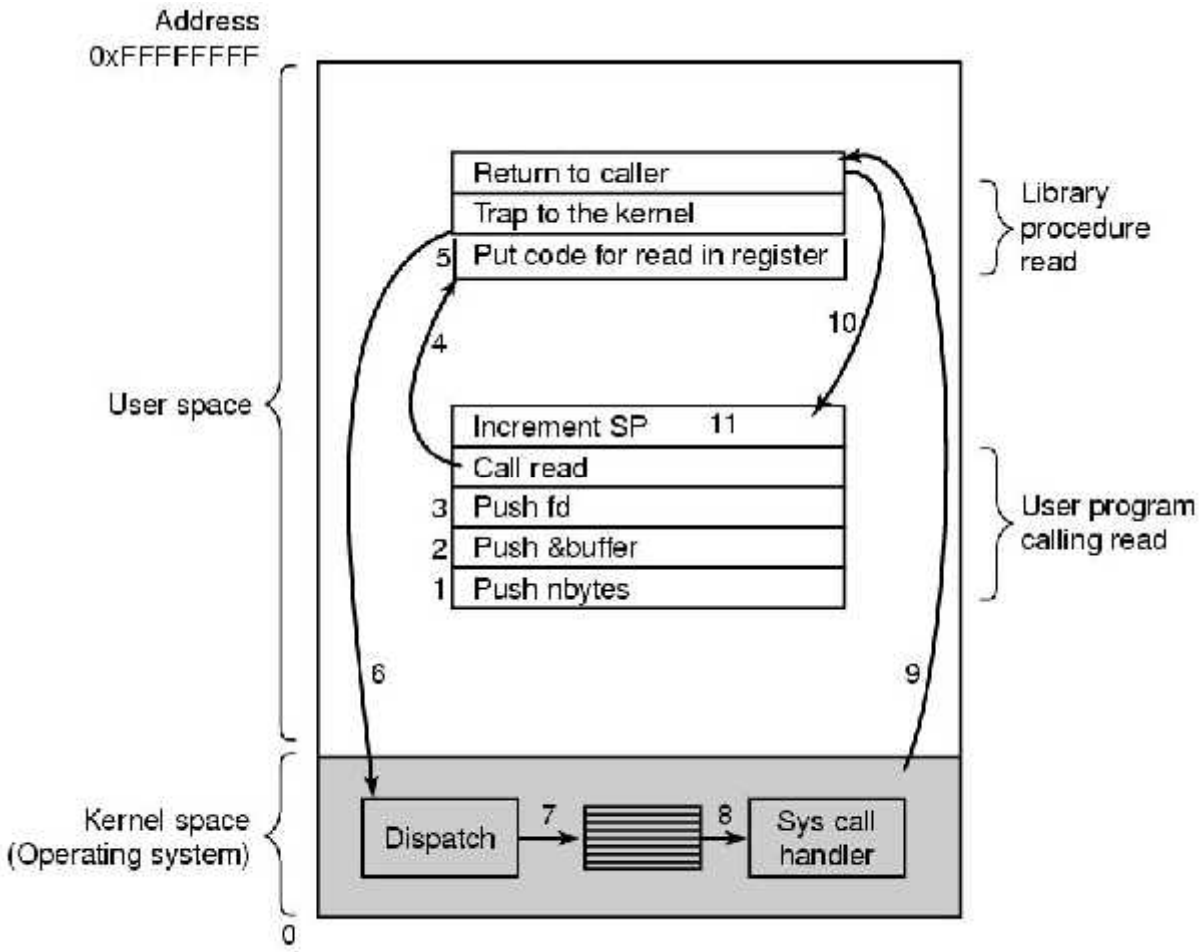


Figure 1-17. The 11 steps in making the system call read(fd, buffer, nbytes).

System Calls for Process Management

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

Figure 1-18. Some of the major POSIX system calls.

System Calls for File Management (1)

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Figure 1-18. Some of the major POSIX system calls.

System Calls for File Management (2)

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Figure 1-18. Some of the major POSIX system calls.

Miscellaneous System Calls

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

Figure 1-18. Some of the major POSIX system calls.

```
int pid;

pid = fork();
if(pid > 0){
  printf("parent: child=%d\n", pid);
  pid = wait();
  printf("child %d is done\n", pid);
} else if(pid == 0){
  printf("child: exiting\n");
  exit();
} else {
  printf("fork error\n");
}
```

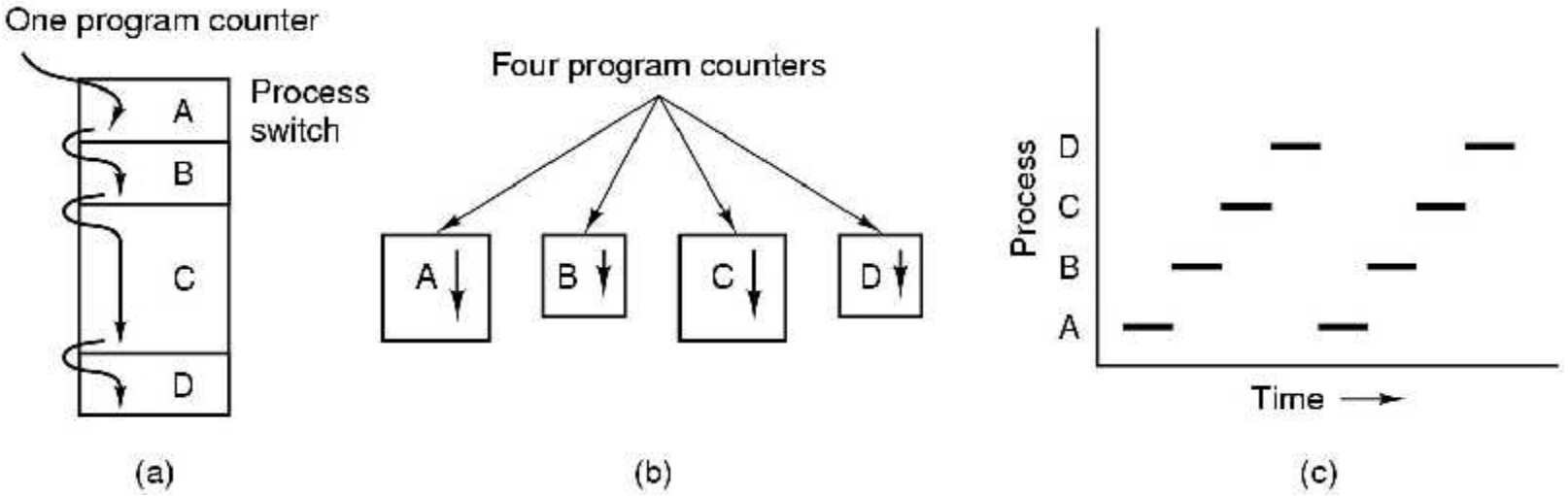
```
char *argv[3];

argv[0] = "echo";
argv[1] = "hello";
argv[2] = 0;
exec("/bin/echo", argv);
printf("exec error\n");
```

What is a process?

- An instance of a program, replete with registers, variables, and a program counter
- It has a program, input, output and a state
- Why is this idea necessary?
  - A computer manages many computations concurrently-need an abstraction to describe how it does it

Multiprogramming



(a) Multiprogramming of four programs. (b) Conceptual model of four independent, sequential processes. (c) Only one program is active at once.

Process Creation

Events which can cause process creation

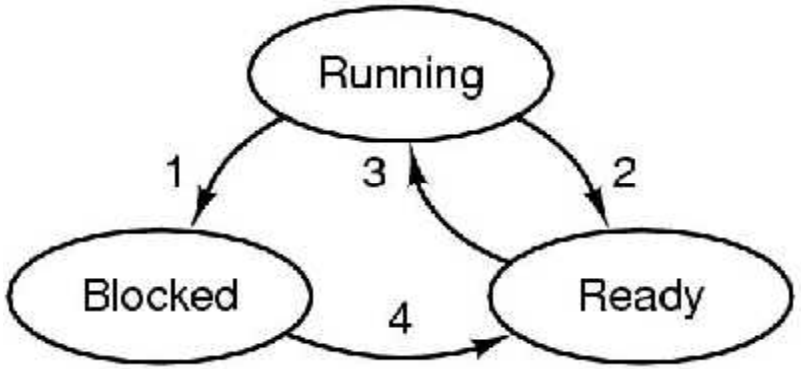
- System initialization.
- Execution of a process creation system call by a running process.
- A user request to create a new process.
- Initiation of a batch job.

Process Termination

Events which cause process termination:

- Normal exit (voluntary).
- Error exit (voluntary).
- Fatal error (involuntary).
- Killed by another process (involuntary).

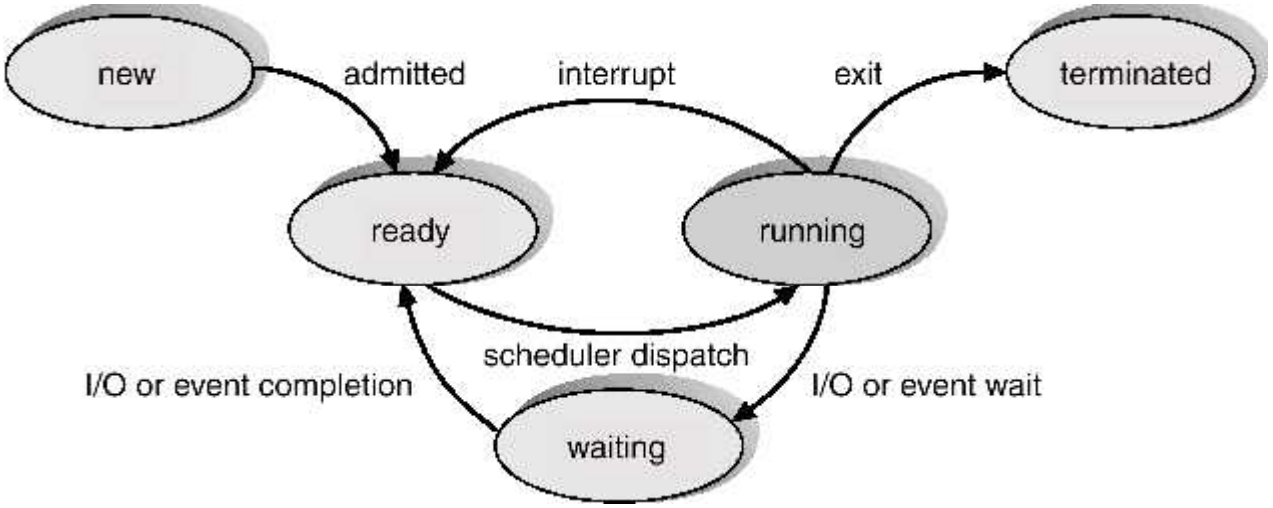
Process States



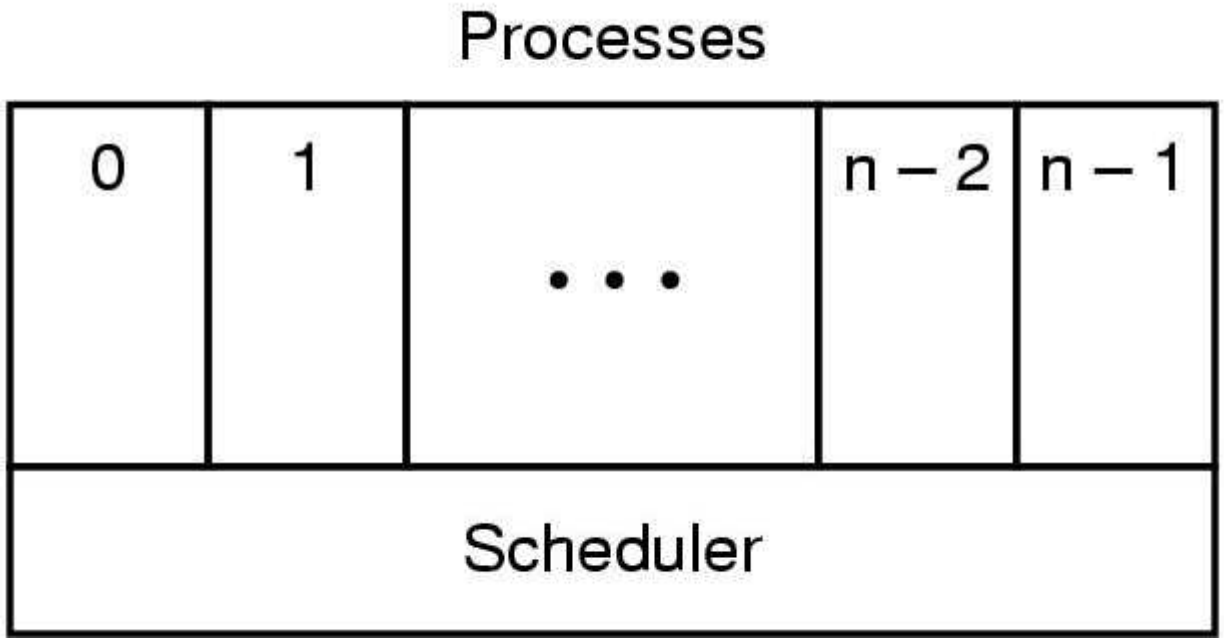
1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

A process can be in running, blocked, or ready state. Transitions between these states are as shown.

- **New** The process is just being put together.
- **Running** Instructions being executed. This running process holds the CPU.
- **Waiting** For an event (hardware, human, or another process.)
- **Ready** The process has all needed resources - waiting for CPU only.
- **Suspended** Another process has explicitly told this process to sleep. It will be awakened when a process explicitly awakens it.
- **Terminated** The process is being torn apart.


new
admitted
interrupt
exit
terminated
ready
running
scheduler dispatch
I/O or event completion
I/O or event wait
waiting

Implementation of Processes (1)



The lowest layer of a process-structured operating system handles interrupts and scheduling. Above that layer are sequential processes.

Implementation of Processes (2)

Process management	Memory management	File management
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Some of the fields of a typical process table entry.