MODERN OPERATING SYSTEMS
Third Edition
ANDREW S. TANENBAUM

Chapter 1
Introduction

What Is An Operating System (1)

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 (2)

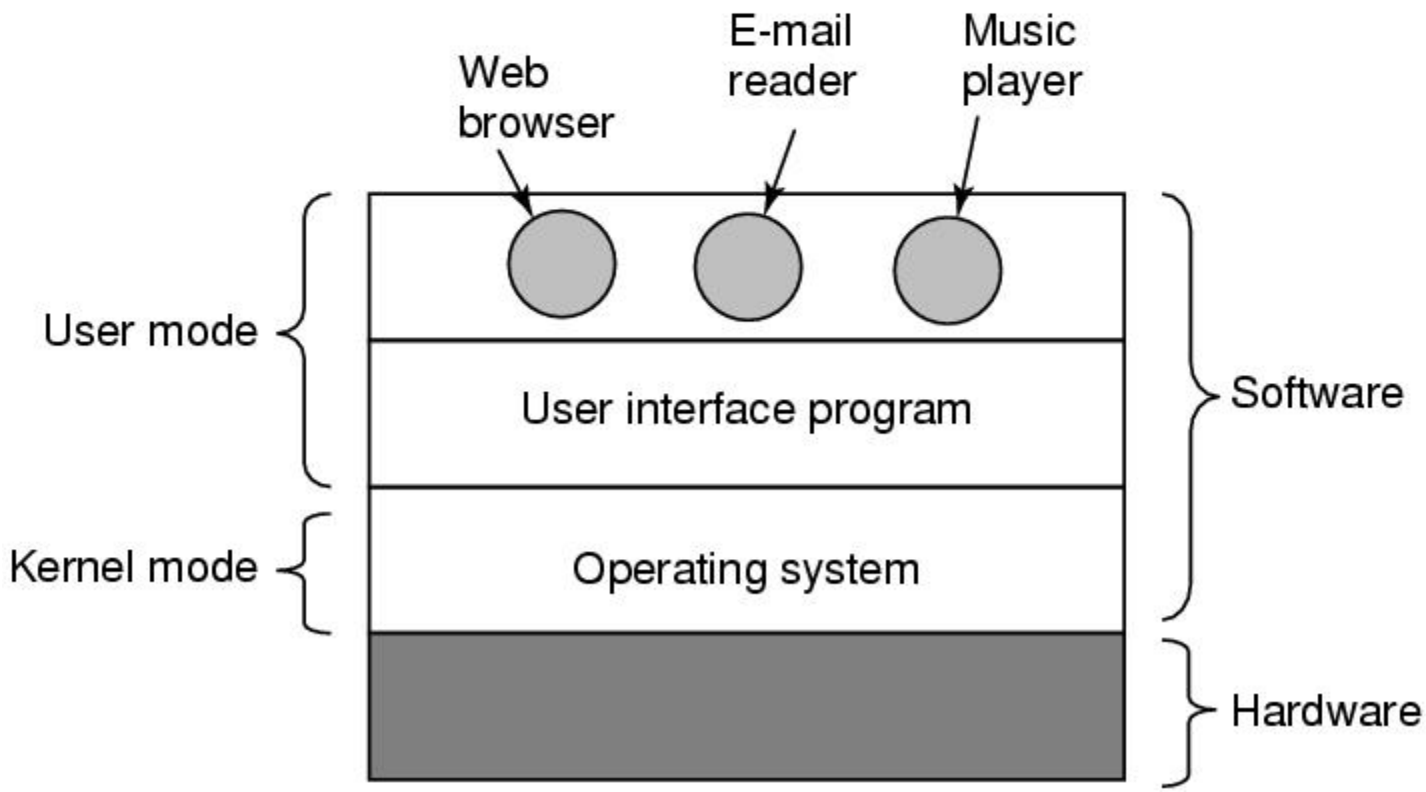

Figure 1-1. Where the operating system fits in.

The Operating System as an Extended Machine

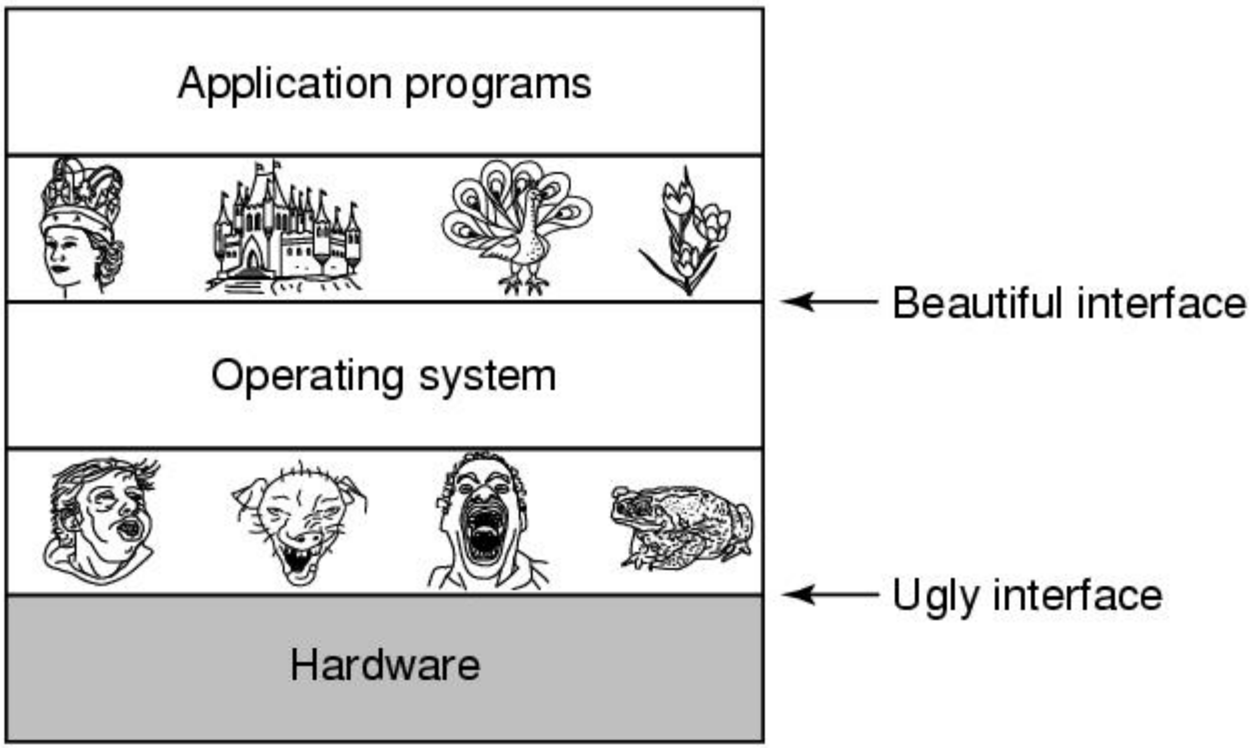

Figure 1-2. Operating systems turn ugly hardware into beautiful abstractions.

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

History of Operating Systems

Generations:

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

Transistors and Batch Systems (1)

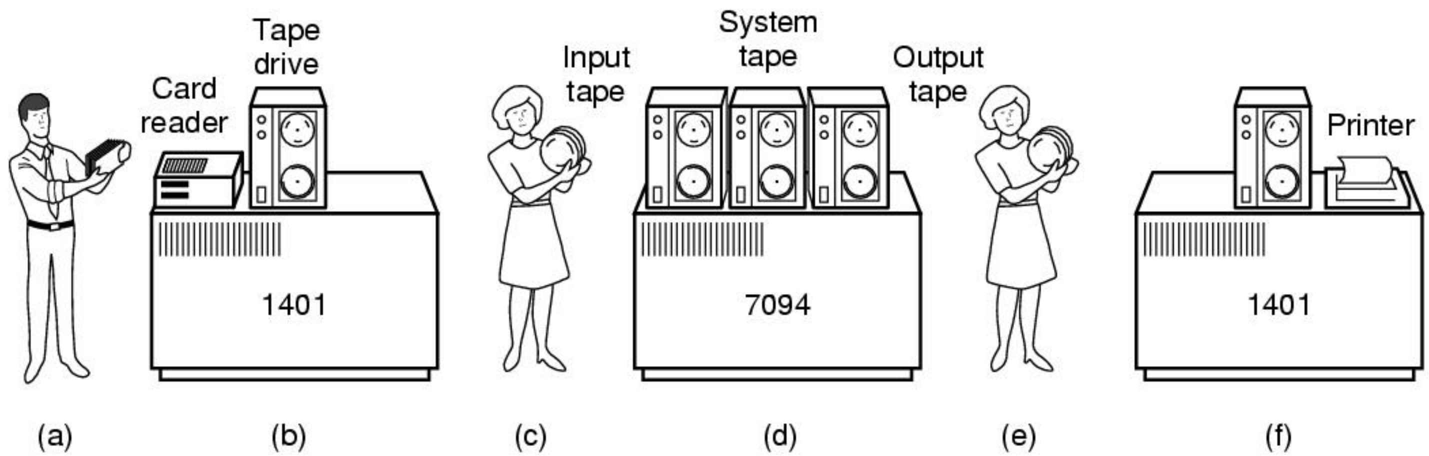

Figure 1-3. An early batch system.
(a) Programmers bring cards to 1401.
(b)1401 reads batch of jobs onto tape.

Transistors and Batch Systems (2)

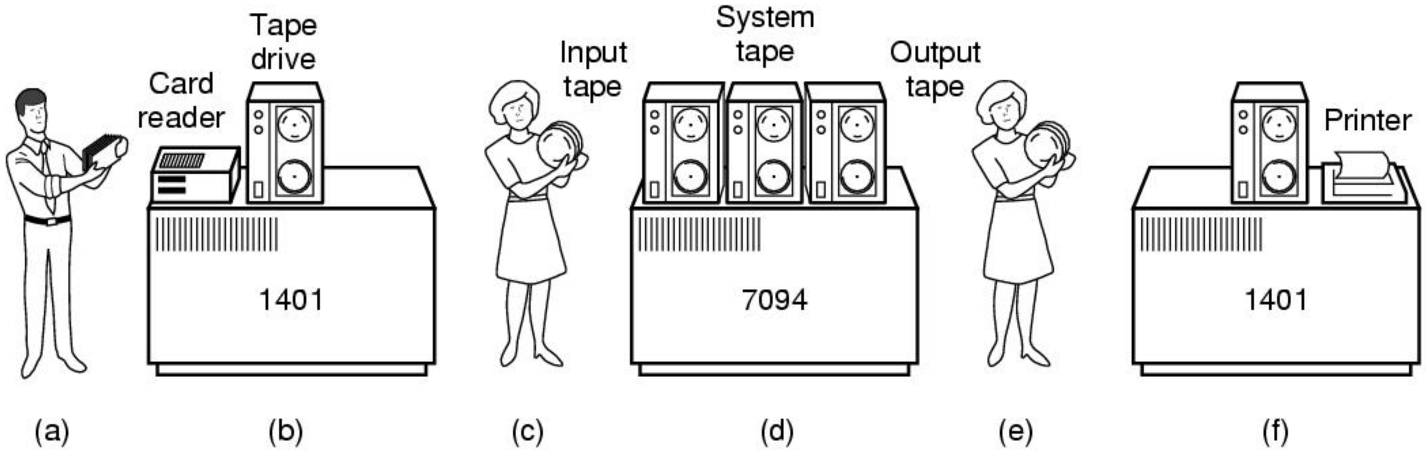

Figure 1-3. (c) Operator carries input tape to 7094. (d) 7094 does computing. (e) Operator carries output tape to 1401. (f) 1401 prints output.

Transistors and Batch Systems (4)

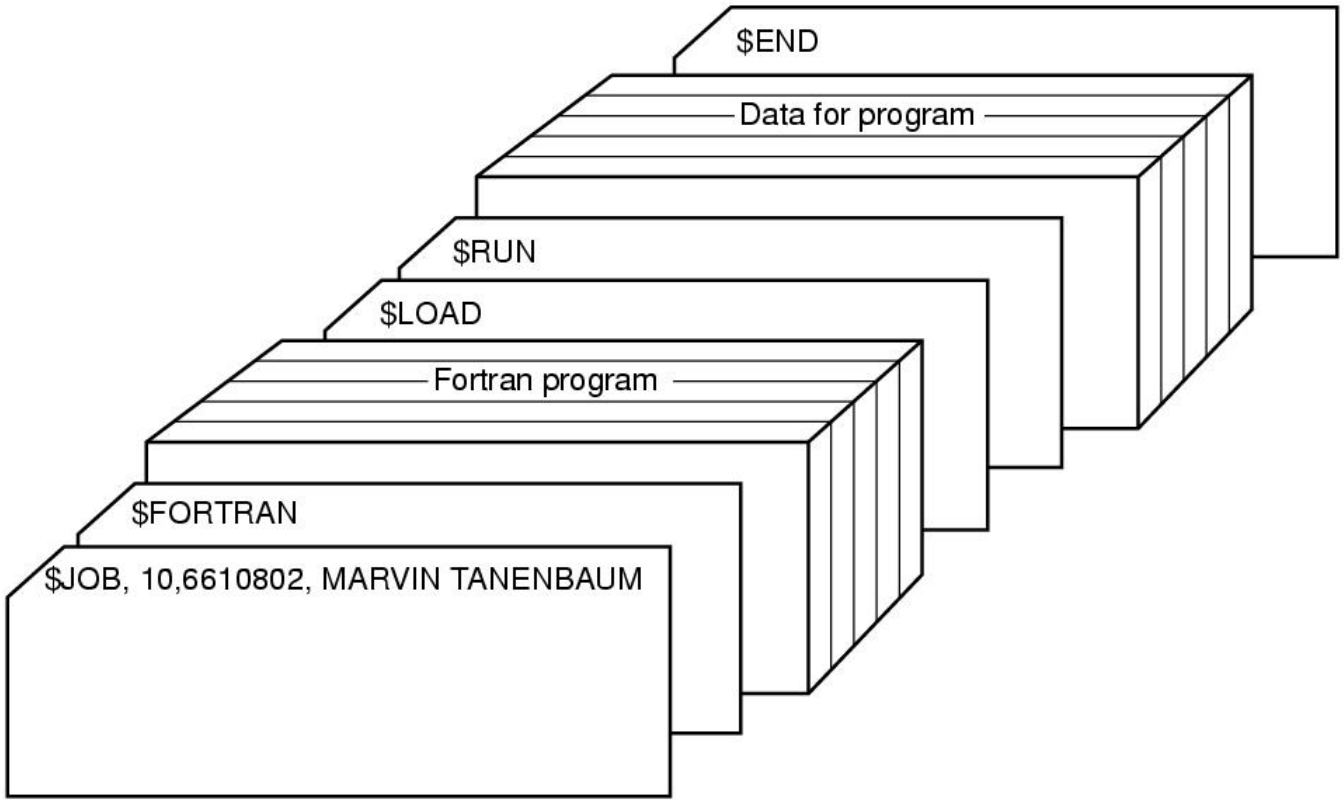

Figure 1-4. Structure of a typical FMS job.

ICs and Multiprogramming

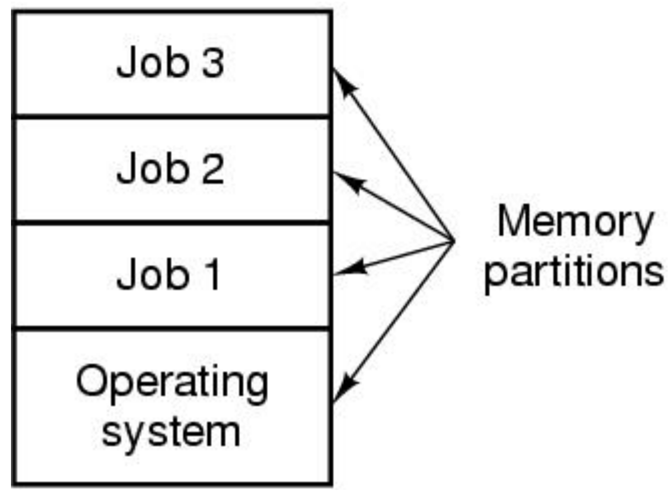

Figure 1-5. A multiprogramming system with three jobs in memory.

Computer Hardware Review

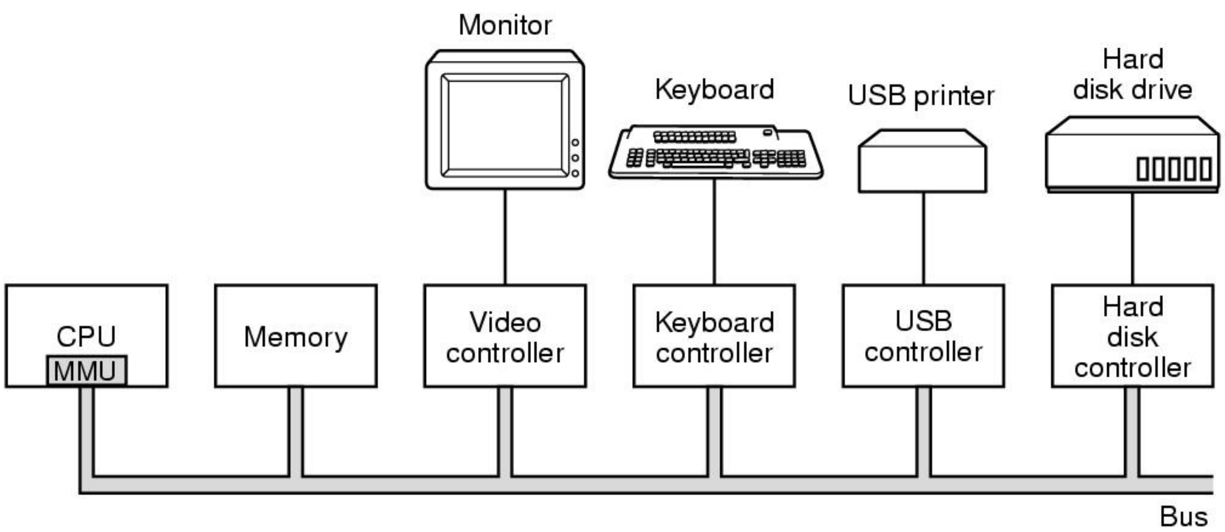

Figure 1-6. Some of the components of a simple personal computer.

CPU Pipelining

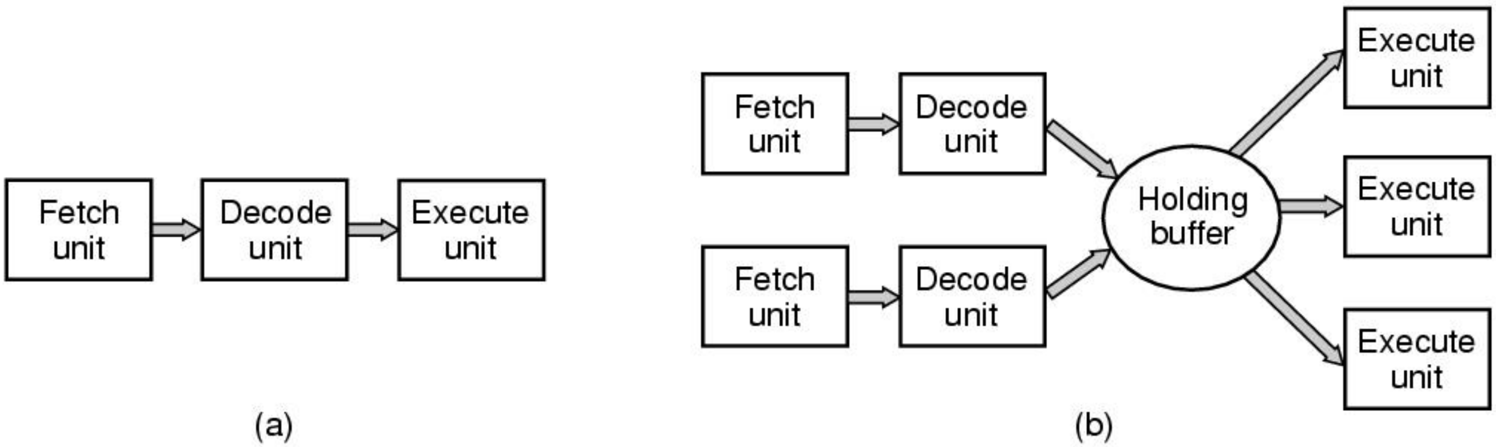

Figure 1-7. (a) A three-stage pipeline. (b) A superscalar CPU.

Multithreaded and Multicore Chips

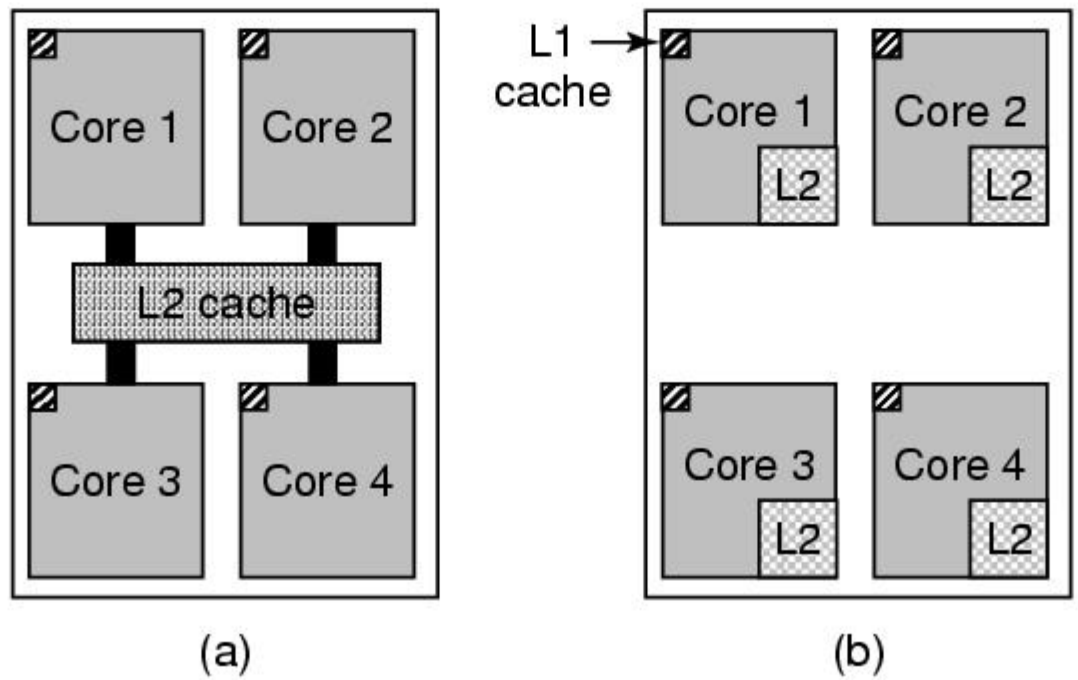

Figure 1-8. (a) A quad-core chip with a shared L2 cache.
(b) A quad-core chip with separate L2 caches.

Memory (1)

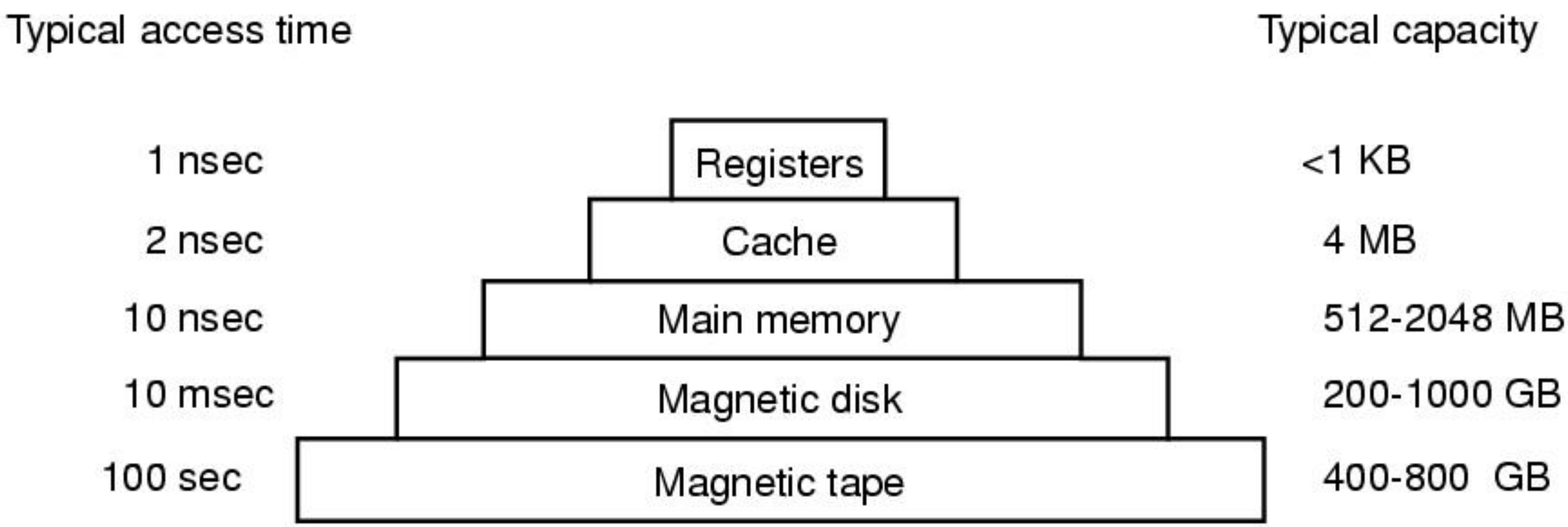

Figure 1-9. A typical memory hierarchy.
The numbers are very rough approximations.

Memory (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

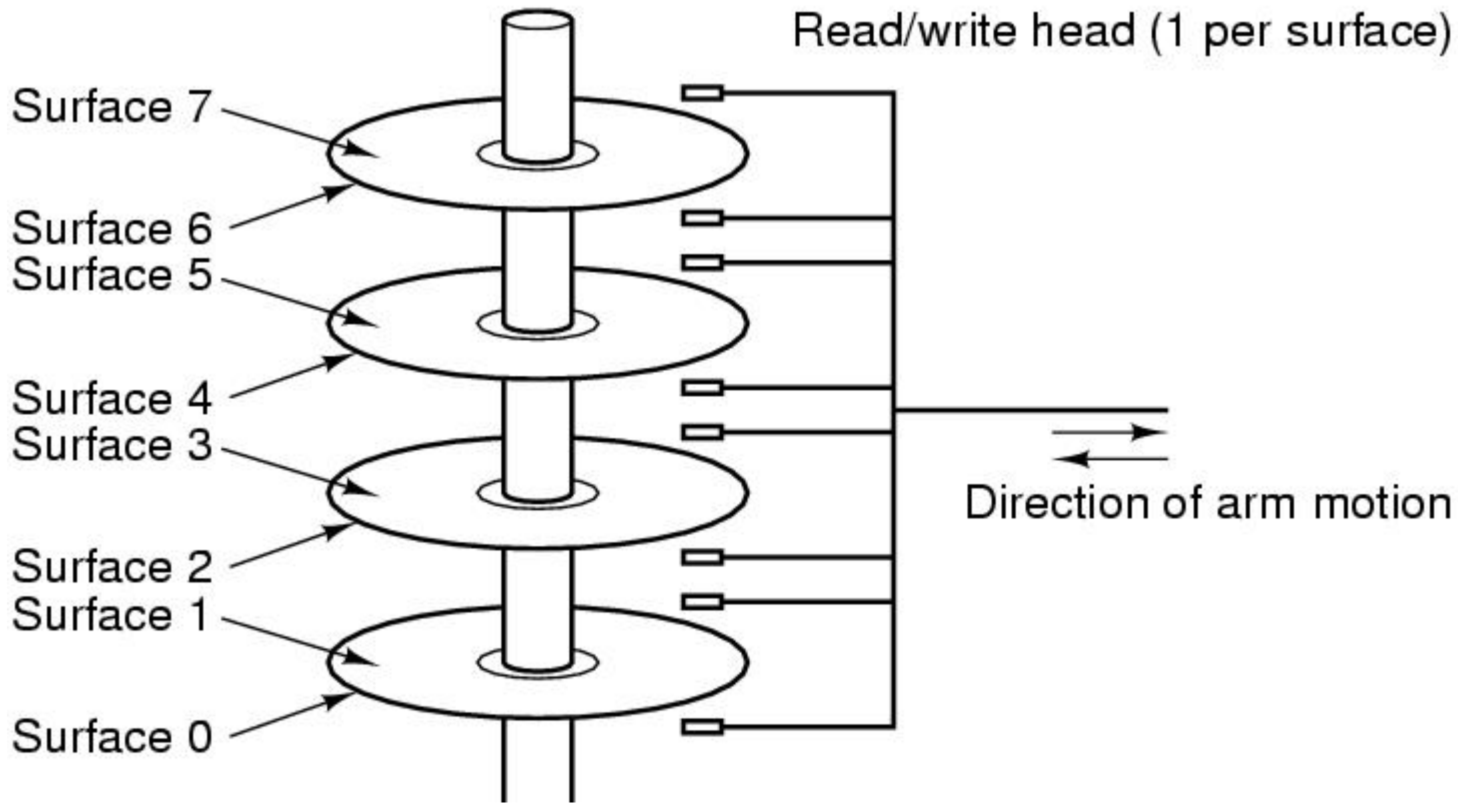

Figure 1-10. Structure of a disk drive.

I/O Devices

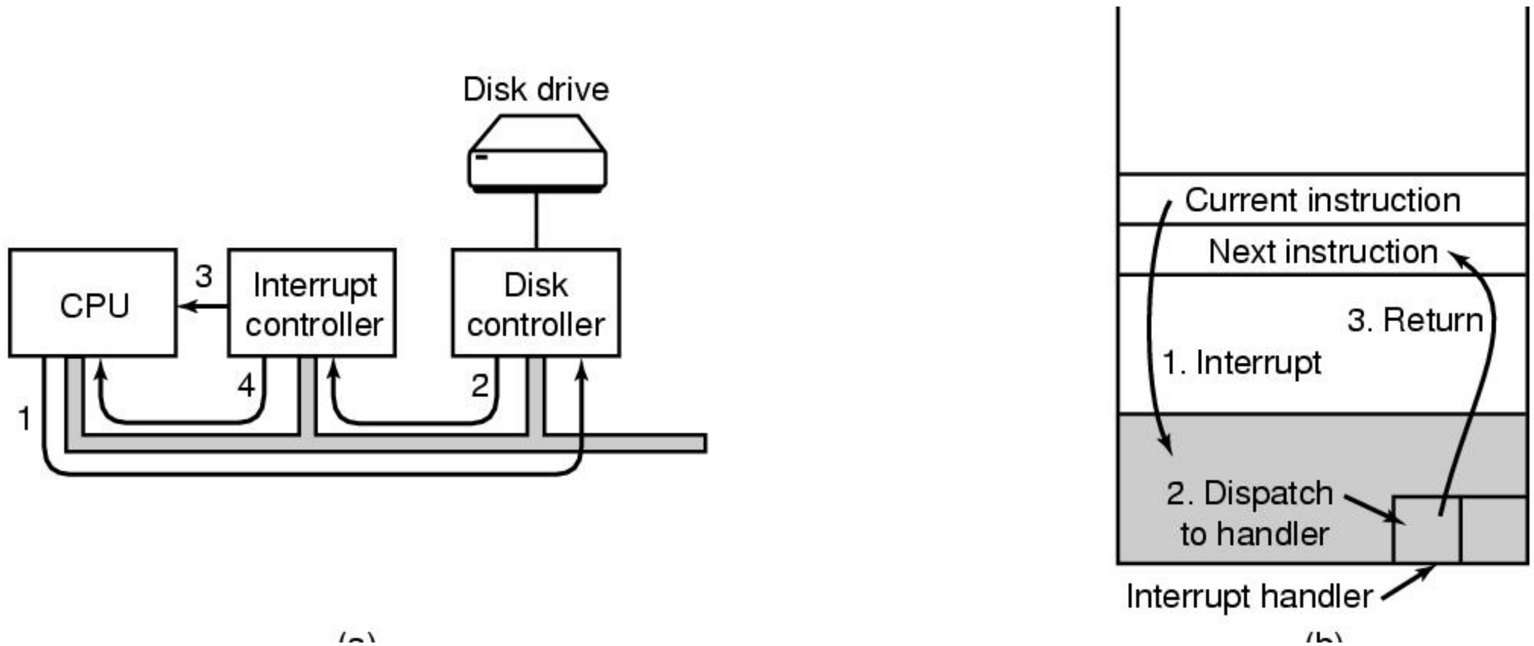

Figure 1-11. (a) The steps in starting an I/O device and getting an interrupt.

Buses

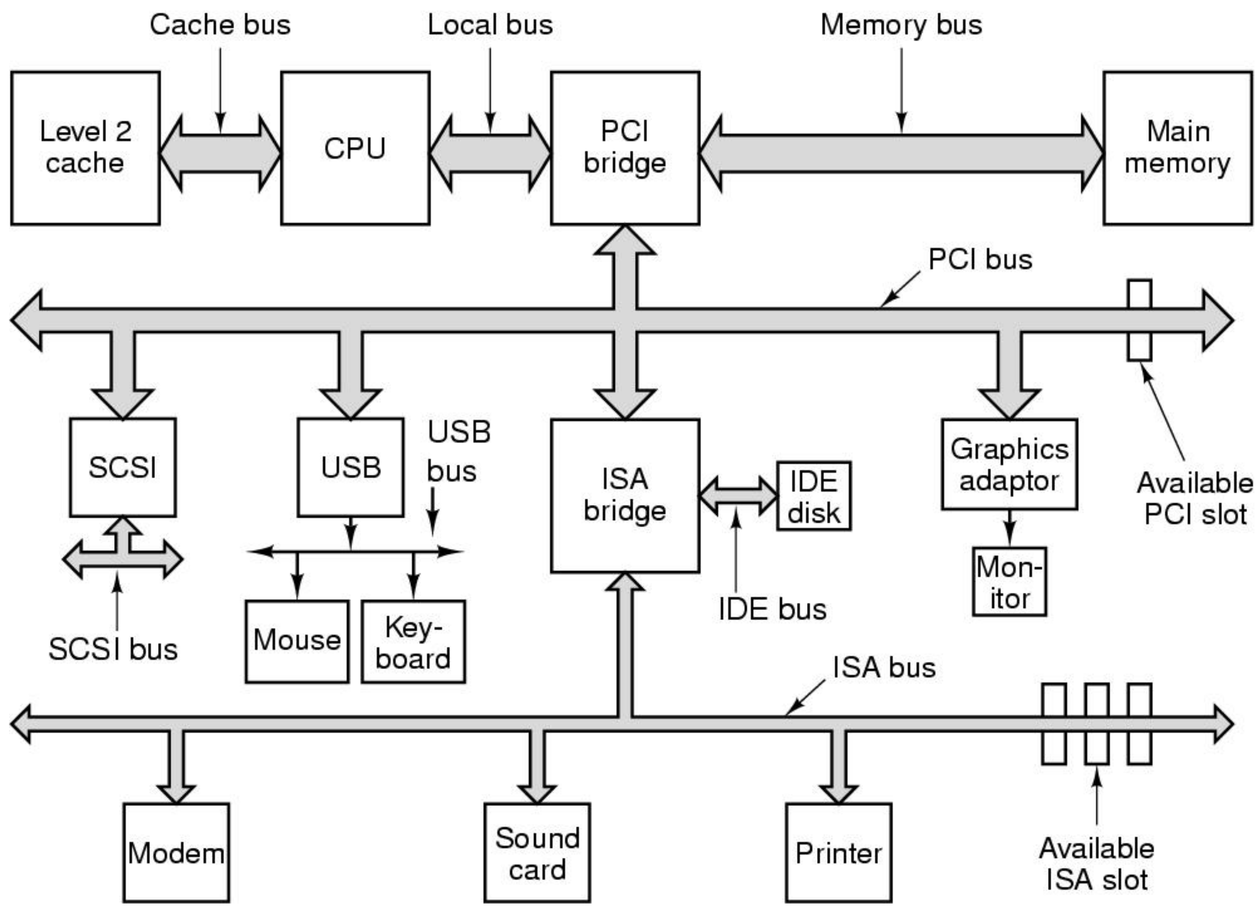

Figure 1-12. The structure of a large Pentium system

The Operating System Zoo

- Mainframe operating systems
- Server operating systems
- Multiprocessor operating systems
- Personal computer operating systems
- Handheld operating systems
- Embedded operating systems
- Sensor node operating systems
- Real-time operating systems
- Smart card operating systems

Operating System Concepts

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

Processes

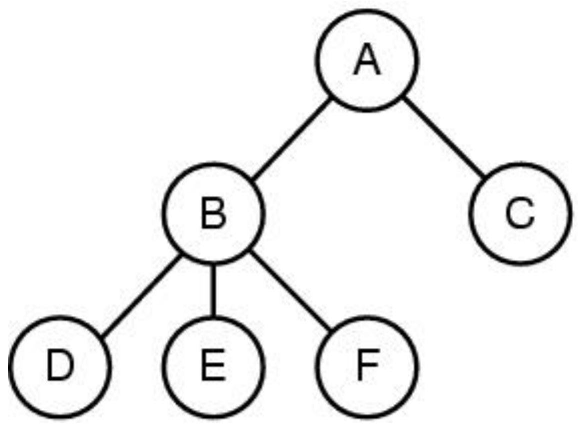

Figure 1-13. A process tree. Process A created two child processes, B and C. Process B created three child processes, D, E, and F.

Files (1)

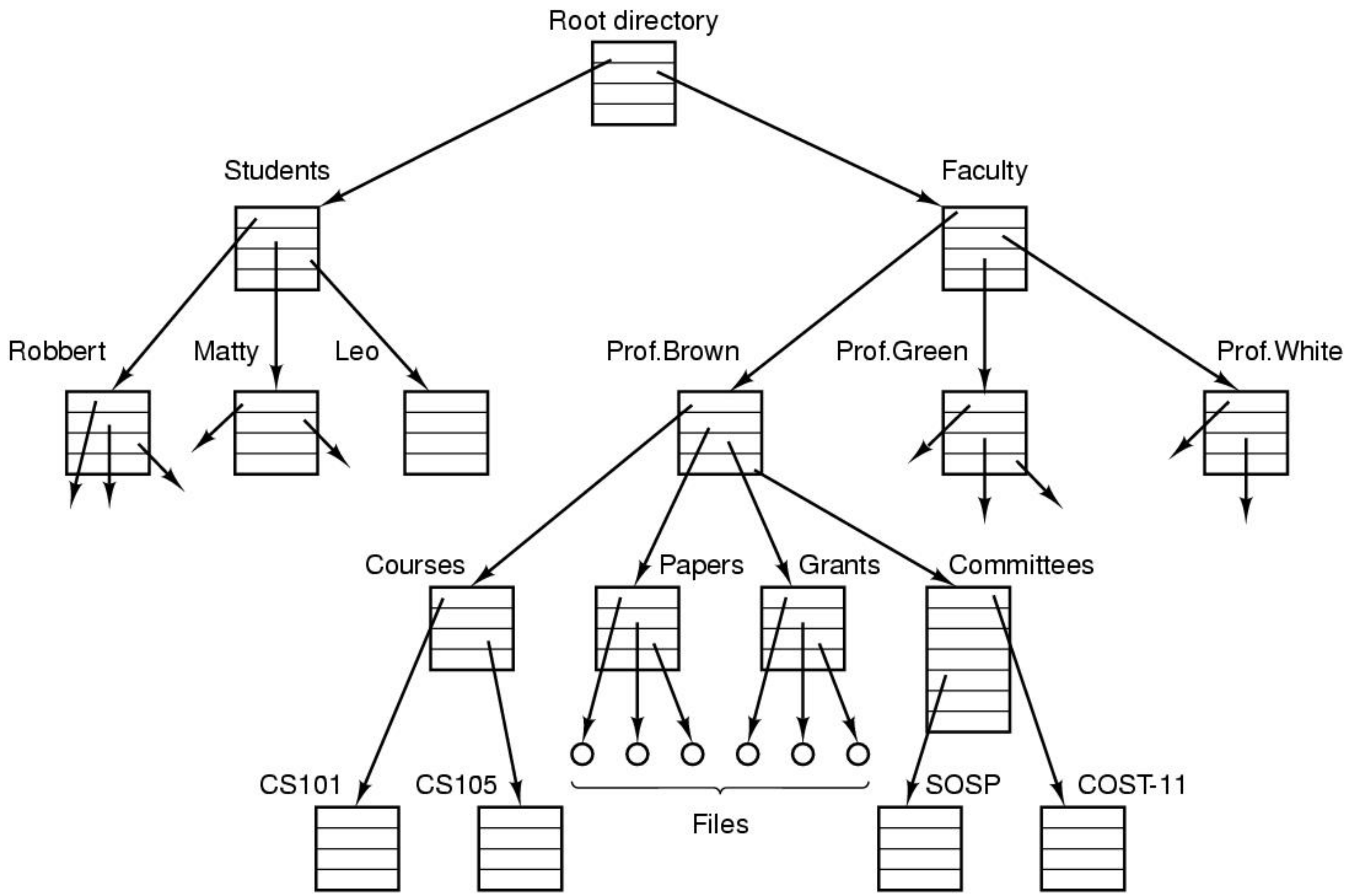

Figure 1-14. A file system for a university department.

Files (2)

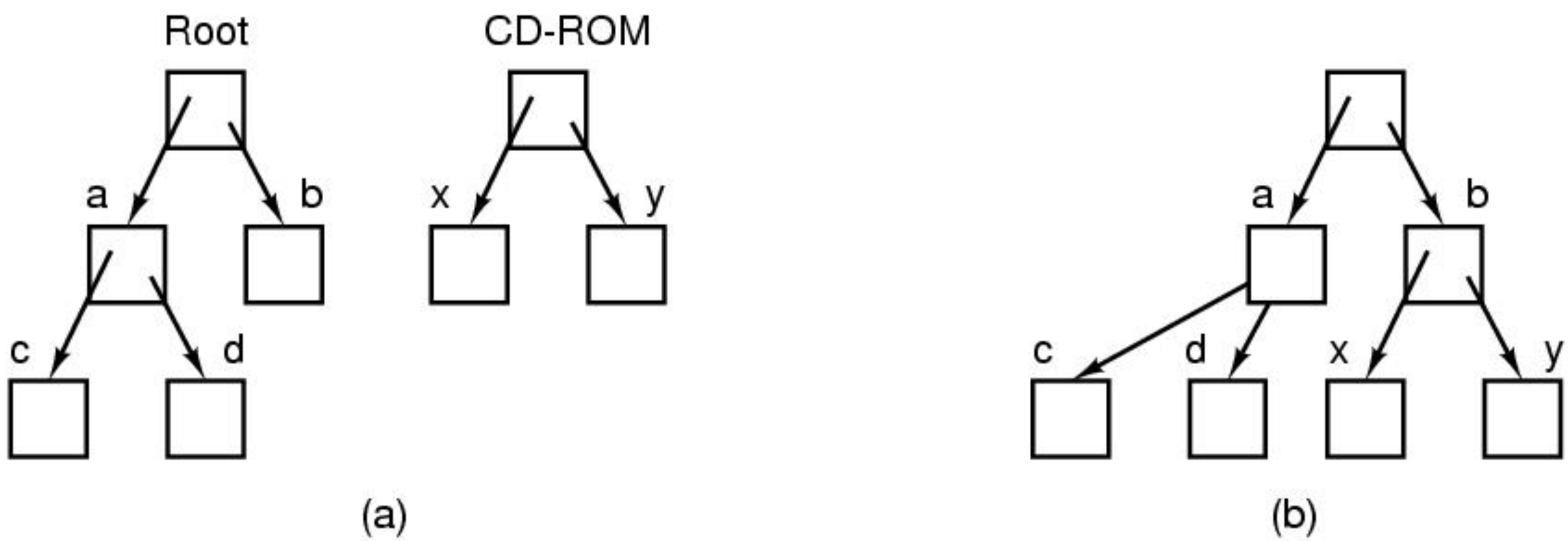

Figure 1-15. (a) Before mounting, the files on the CD-ROM are not accessible. (b) After mounting, they are part of the file hierarchy.

Files (3)

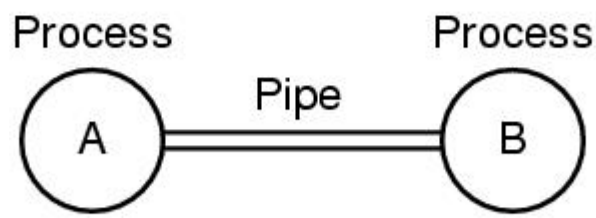

Figure 1-16. Two processes connected by a pipe.

System Calls

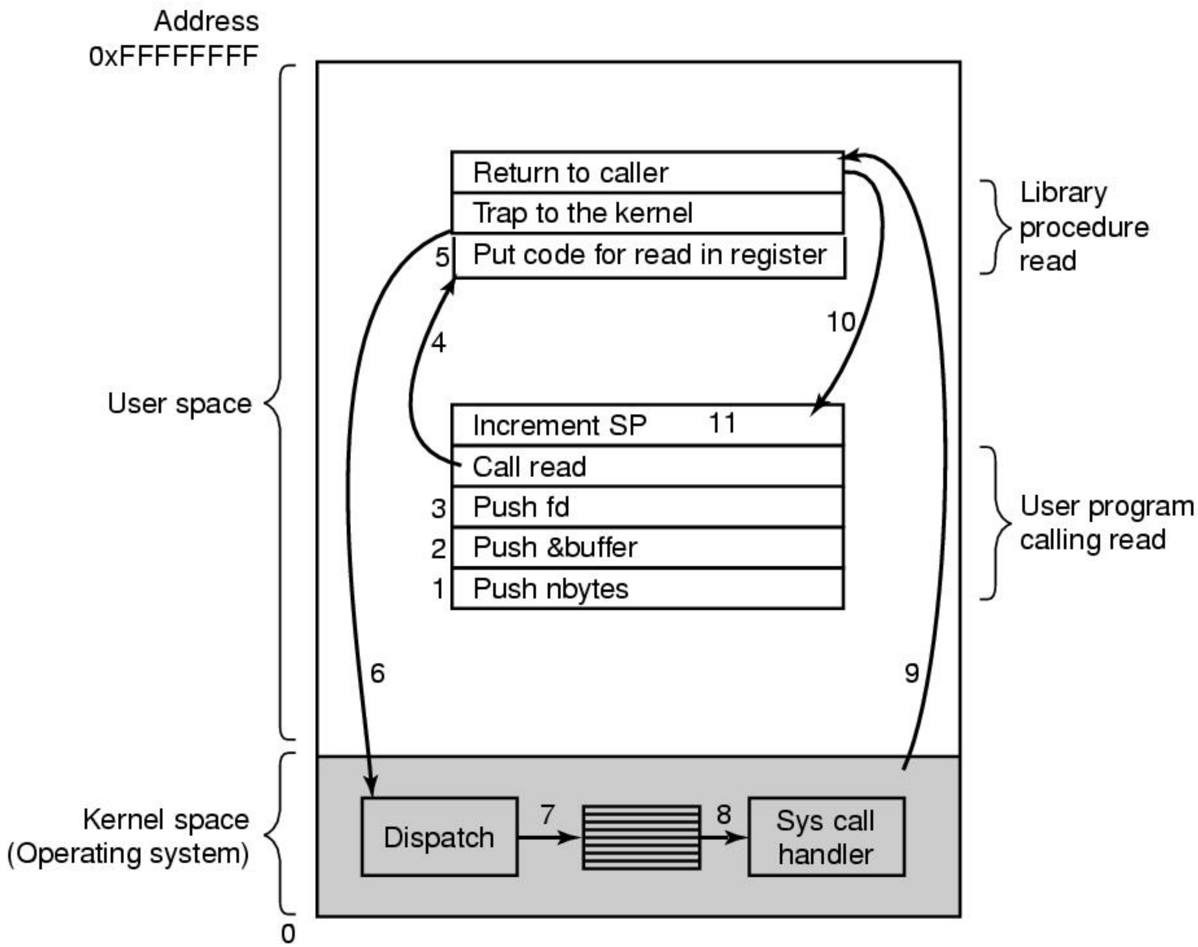

Figure 1-17. The 11 steps in making the system call read(fd, buffer, nbytes).

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.

A Simple Shell

```
#define TRUE 1

while (TRUE) {                                  /* repeat forever */
    type_prompt( );                             /* display prompt on the screen */
    read_command(command, parameters);          /* read input from terminal */

    if (fork( ) != 0) {                         /* fork off child process */
        /* Parent code. */
        waitpid(−1, &status, 0);                /* wait for child to exit */
    } else {
        /* Child code. */
        execve(command, parameters, 0);         /* execute command */
    }
}
```

Figure 1-19. A stripped-down shell.

Memory Layout

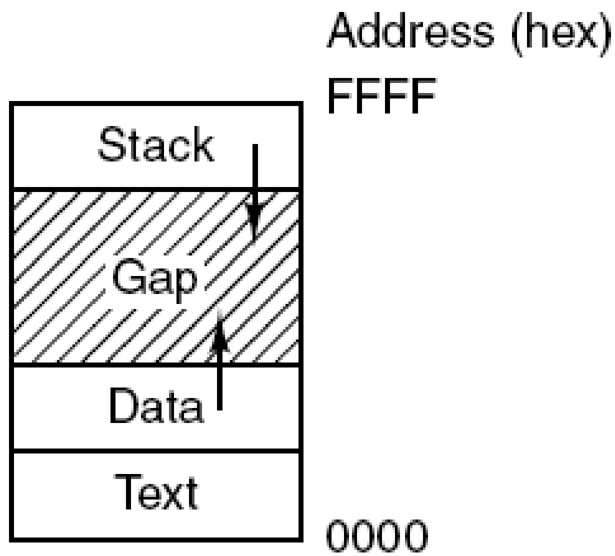

Figure 1-20. Processes have three segments: text, data, and stack.

Linking

/usr/ast

16	mail
81	games
40	test

/usr/jim

31	bin
70	memo
59	f.c.
38	prog1

(a)

/usr/ast

16	mail
81	games
40	test
70	note

/usr/jim

31	bin
70	memo
59	f.c.
38	prog1

(b)

Figure 1-21. (a) Two directories before linking */usr/jim/memo* to ast’s directory. (b) The same directories after linking.

Mounting

Figure 1-22. (a) File system before the mount.
(b) File system after the mount.

Windows Win32 API

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

Figure 1-23. The Win32 API calls that roughly correspond to the UNIX calls of Fig. 1-18.

Operating Systems Structure

Monolithic systems – basic structure:

- A main program that invokes the requested service procedure.
- A set of service procedures that carry out the system calls.
- A set of utility procedures that help the service procedures.

Monolithic Systems

```
#define FALSE  0
#define TRUE   1
#define N      2                            /* number of processes */

int turn;                                   /* whose turn is it? */
int interested[N];                          /* all values initially 0 (FALSE) */

void enter_region(int process);             /* process is 0 or 1 */
{
      int other;                            /* number of the other process */

      other = 1 – process;                  /* the opposite of process */
      interested[process] = TRUE;           /* show that you are interested */
      turn = process;                       /* set flag */
      while (turn == process && interested[other] == TRUE) /* null statement */ ;
}

void leave_region(int process)              /* process: who is leaving */
{
      interested[process] = FALSE;          /* indicate departure from critical region */
}
```

Figure 1-24. A simple structuring model for a monolithic system.

Layered Systems

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Figure 1-25. Structure of the THE operating system.

Microkernels

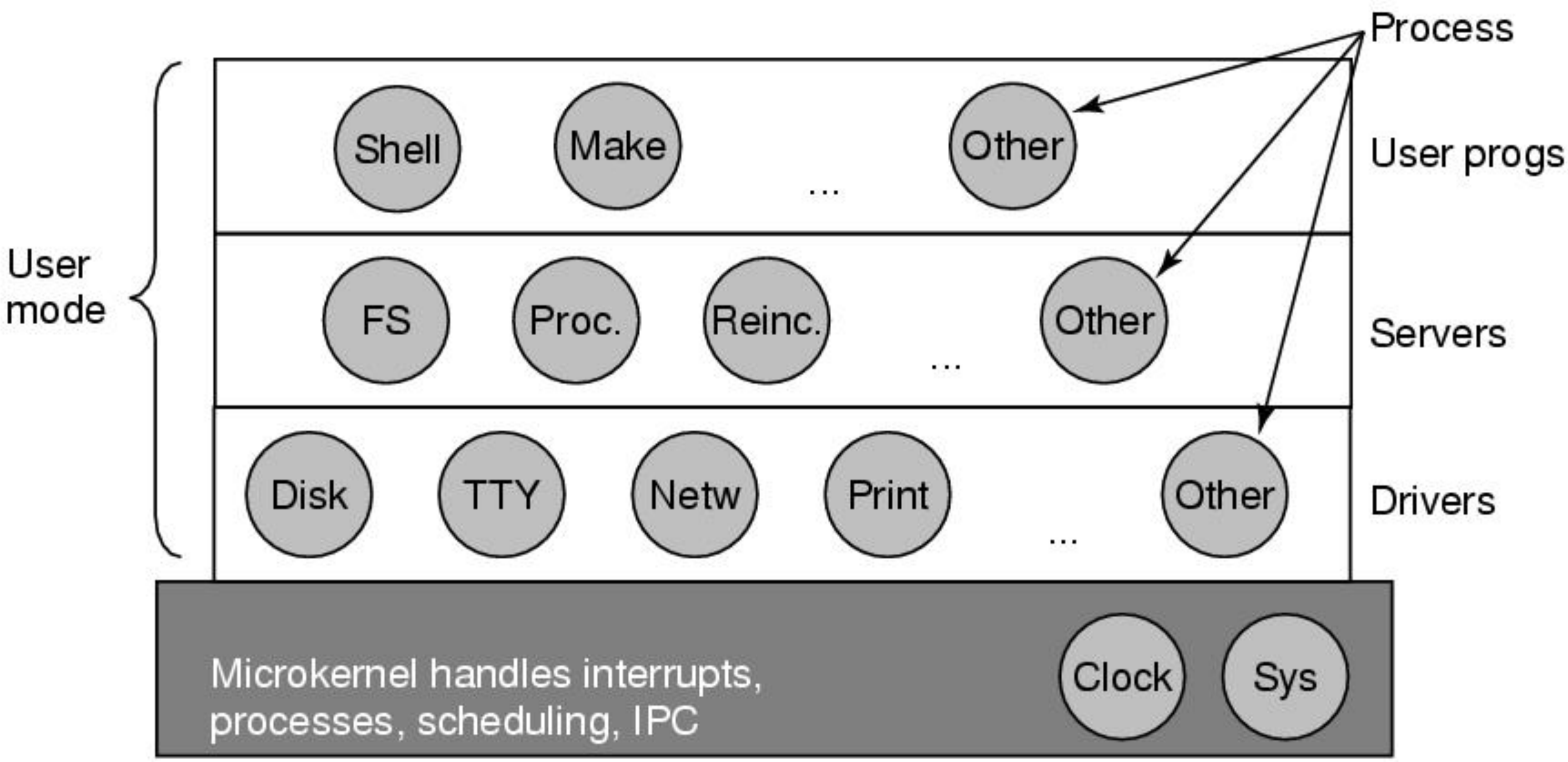

Figure 1-26. Structure of the MINIX 3 system.

Client-Server Model

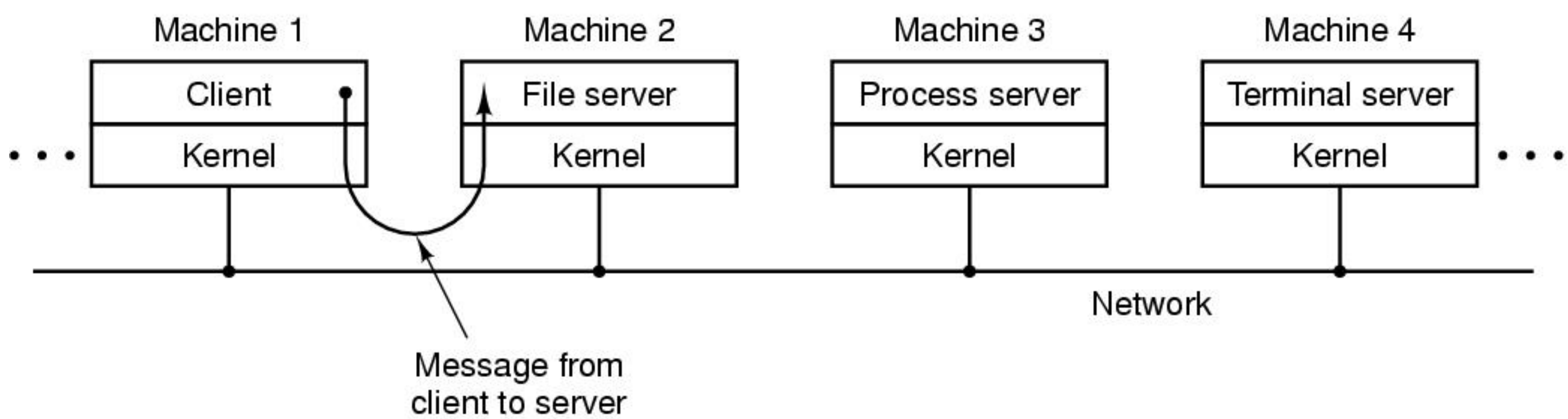

Figure 1-27. The client-server model over a network.

Virtual Machines (1)

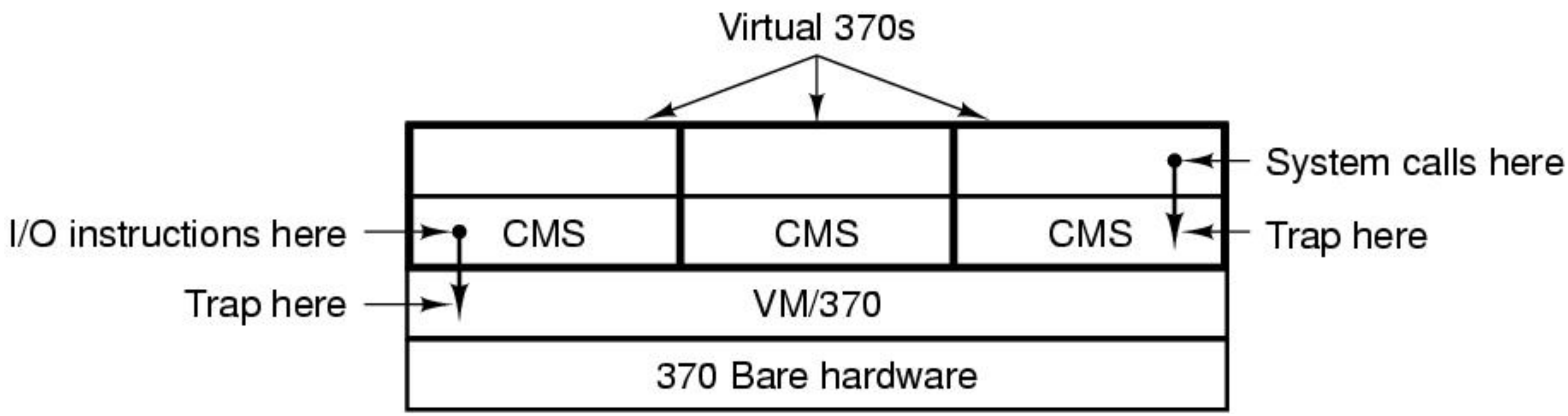

Figure 1-28. The structure of VM/370 with CMS.

Virtual Machines (2)

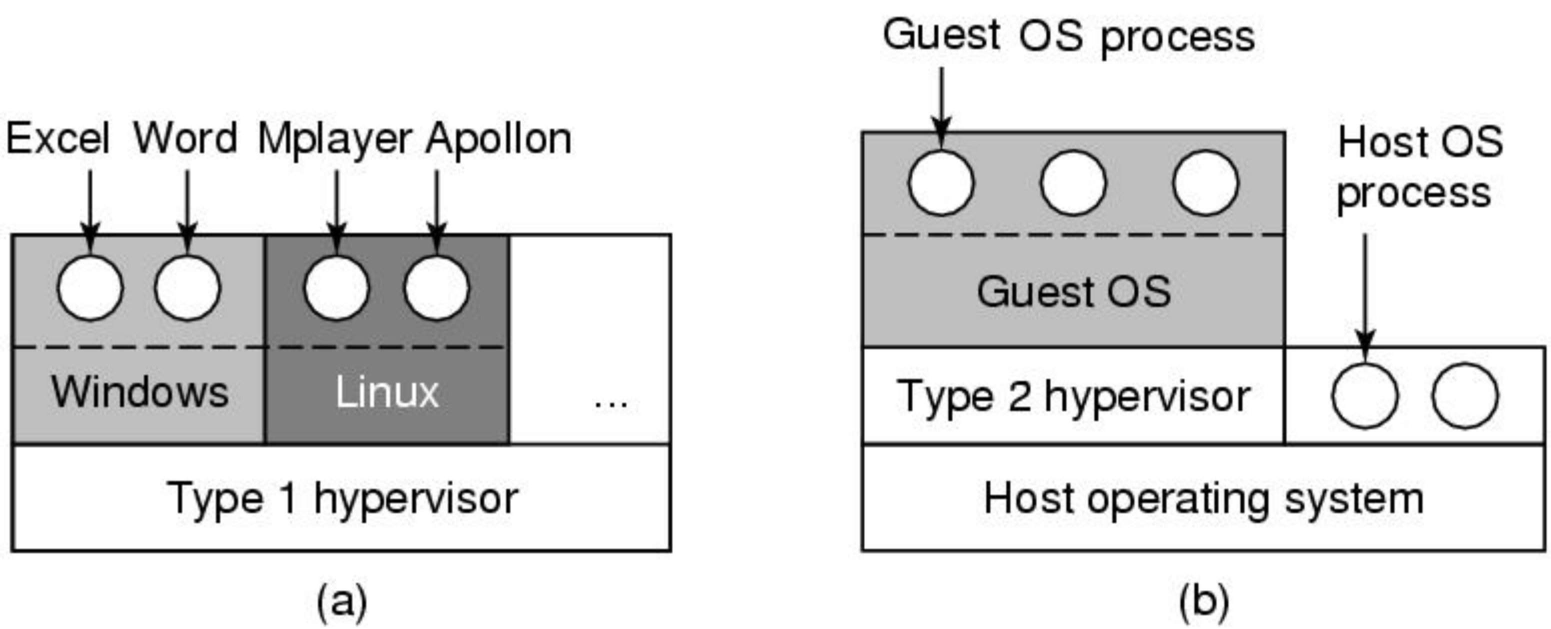

Figure 1-29. (a) A type 1 hypervisor. (b) A type 2 hypervisor.

The World According to C

- The C language
- Header files
- Large programming projects
- The model of run time

The Model of Run Time

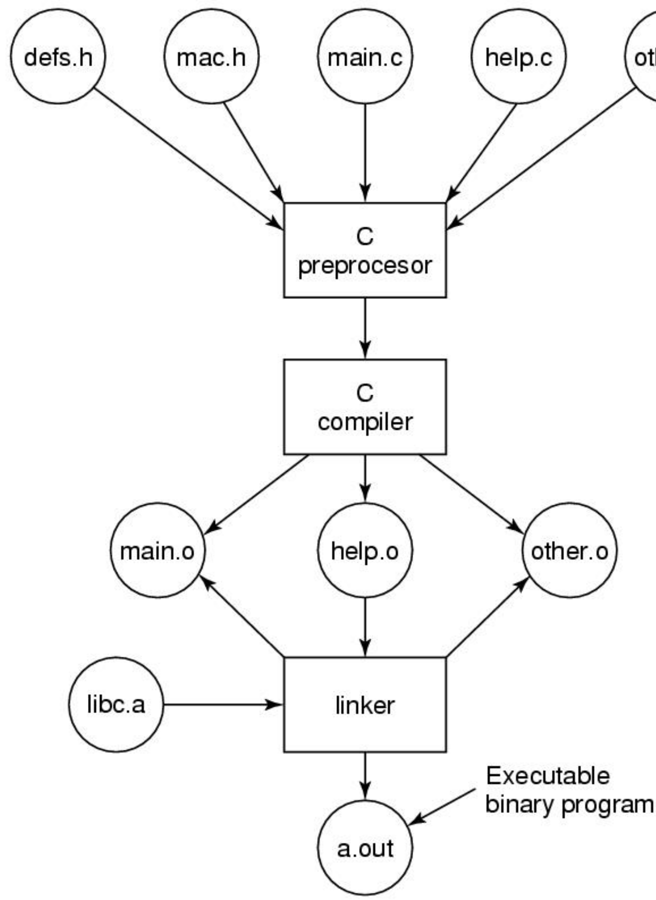

Figure 1-30. The process of compiling C and header files to make an executable.