Introduction

Chapter 1

Components of a Modern Computer (1)

- One or more processors
- Main memory
- Disks
- Printers
- Keyboard
- Mouse
- Display
- Network interfaces
- I/O devices

Components of a Modern Computer (2)

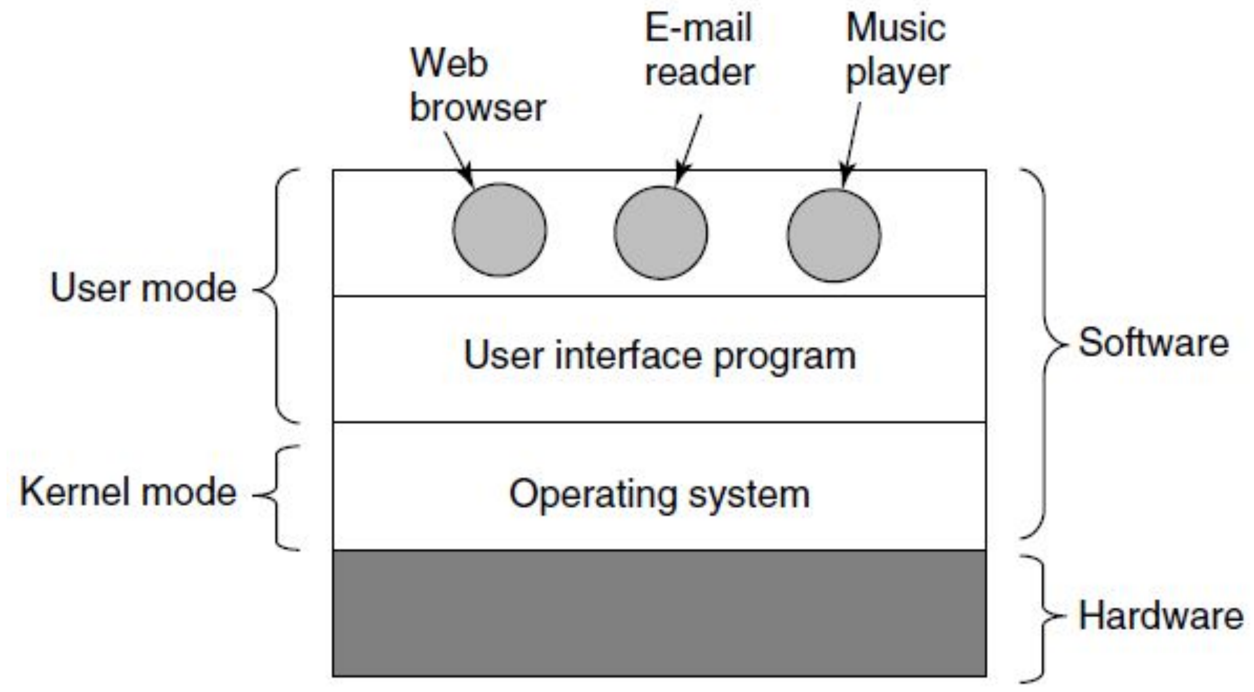


Figure 1-1. Where the operating system fits in.

The Operating System as an Extended Machine

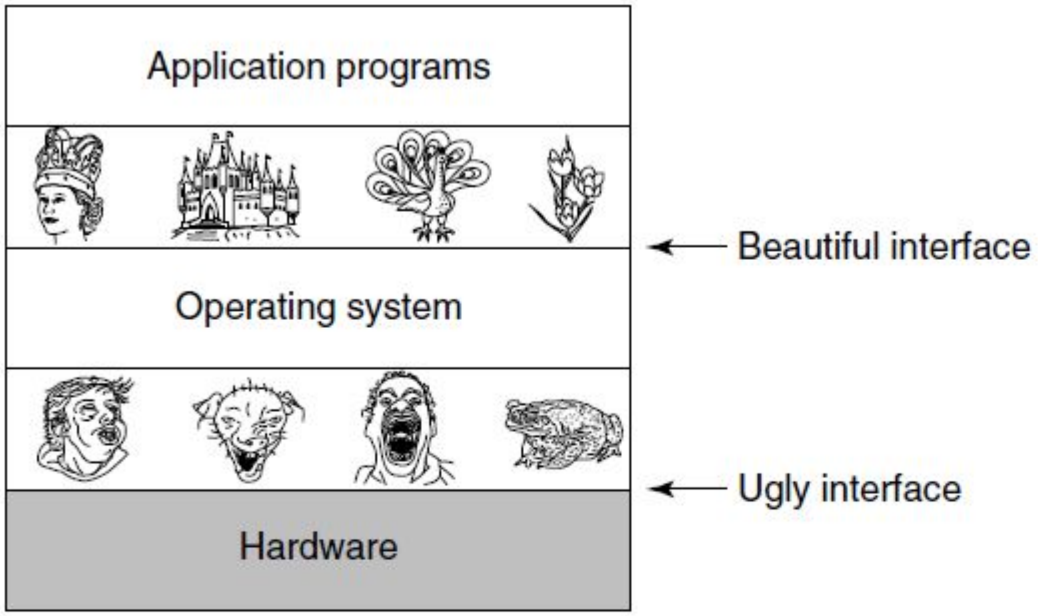


Figure 1-2. Operating systems turn ugly hardware into beautiful abstractions.

The Operating System as a Resource Manager

- Top down view
  - Provide abstractions to application programs
- Bottom up view
  - Manage pieces of complex system
- Alternative view
  - Provide orderly, controlled allocation of resources

History of Operating Systems

- The first generation (1945–55) vacuum tubes
- The second generation (1955–65) transistors and batch systems
- The third generation (1965–1980) ICs and multiprogramming
- The fourth generation (1980–present) personal computers
- The fifth generation (1990–present) mobile computers

Transistors and Batch Systems (1)

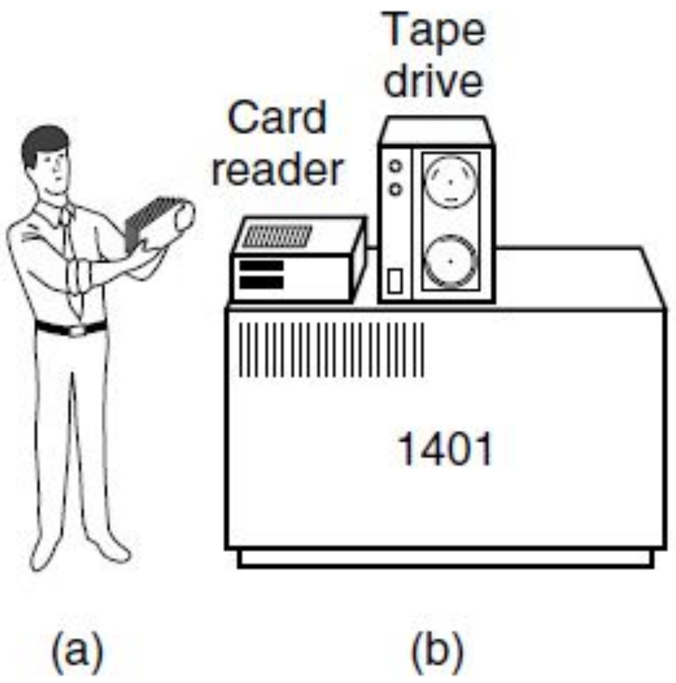


Figure 1-3. An early batch system. (a) Programmers bring cards to 1401. (b) 1401 reads batch of jobs onto tape.

Transistors and Batch Systems (2)

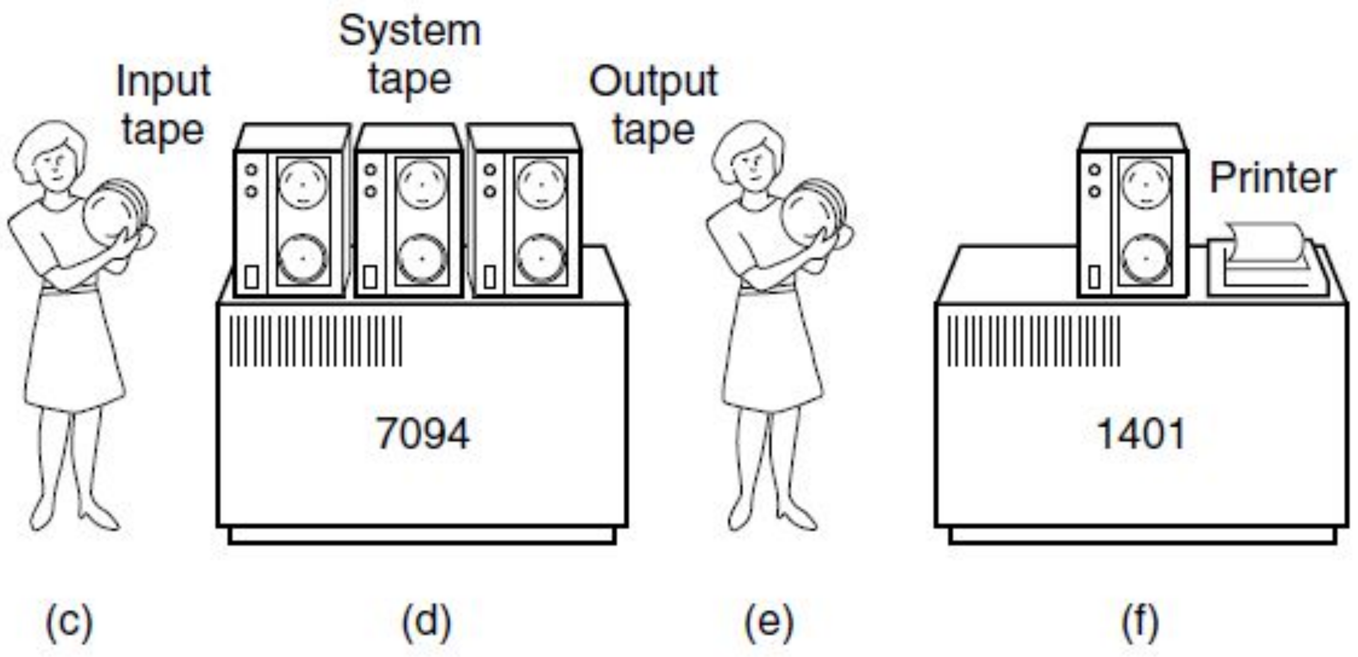


Figure 1-3. An early batch system. (c) Operator carries input tape to 7094. (d)7094 does computing. (e) Operator carries output tape to 1401. (f) 1401 prints output.

Transistors and Batch Systems (3)

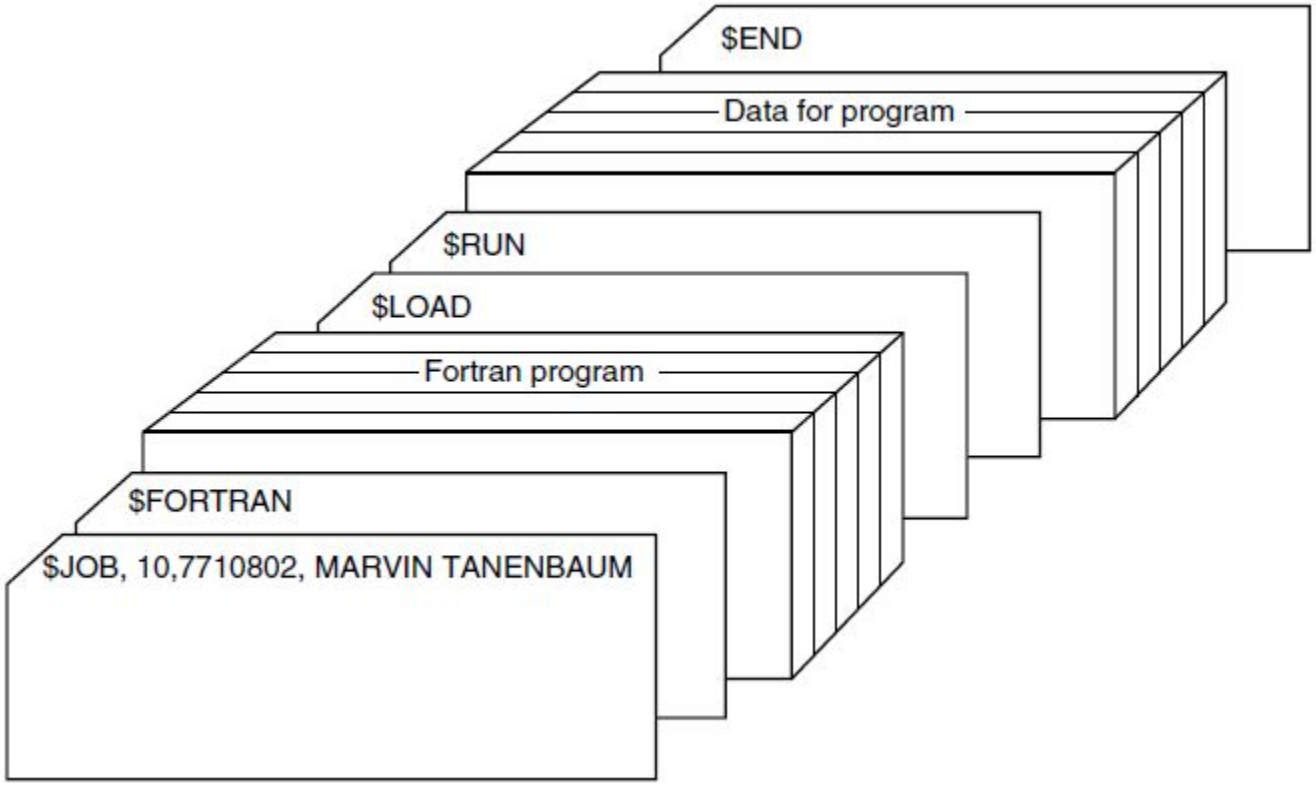


Figure 1-4. Structure of a typical FMS job.

ICs and Multiprogramming

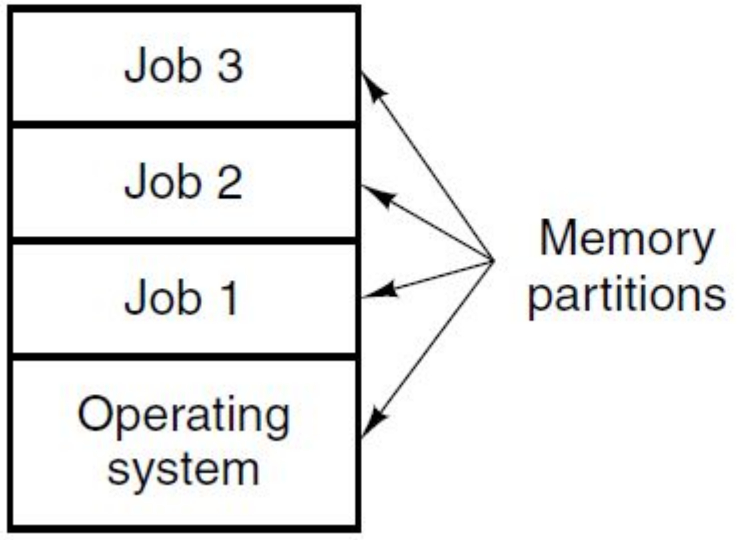


Figure 1-5. A multiprogramming system with three jobs in memory.

Processors (1)

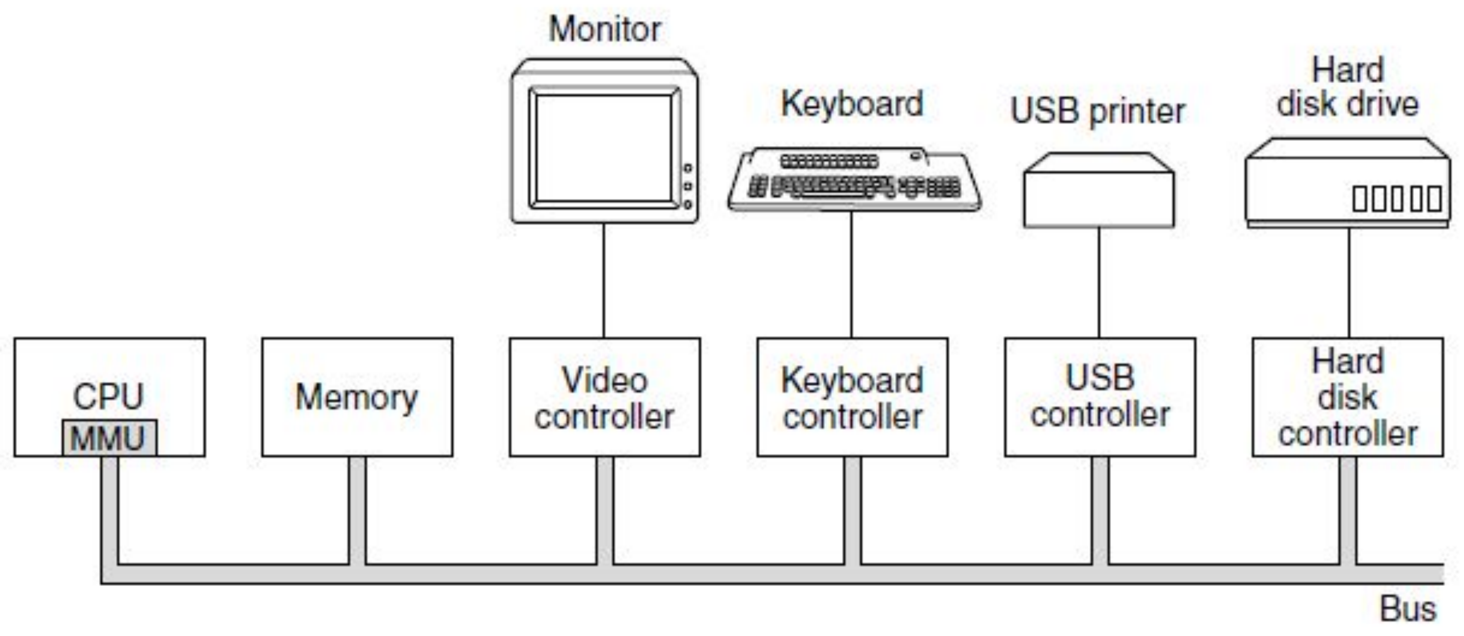


Figure 1-6. Some of the components of a simple personal computer.

Processors (2)

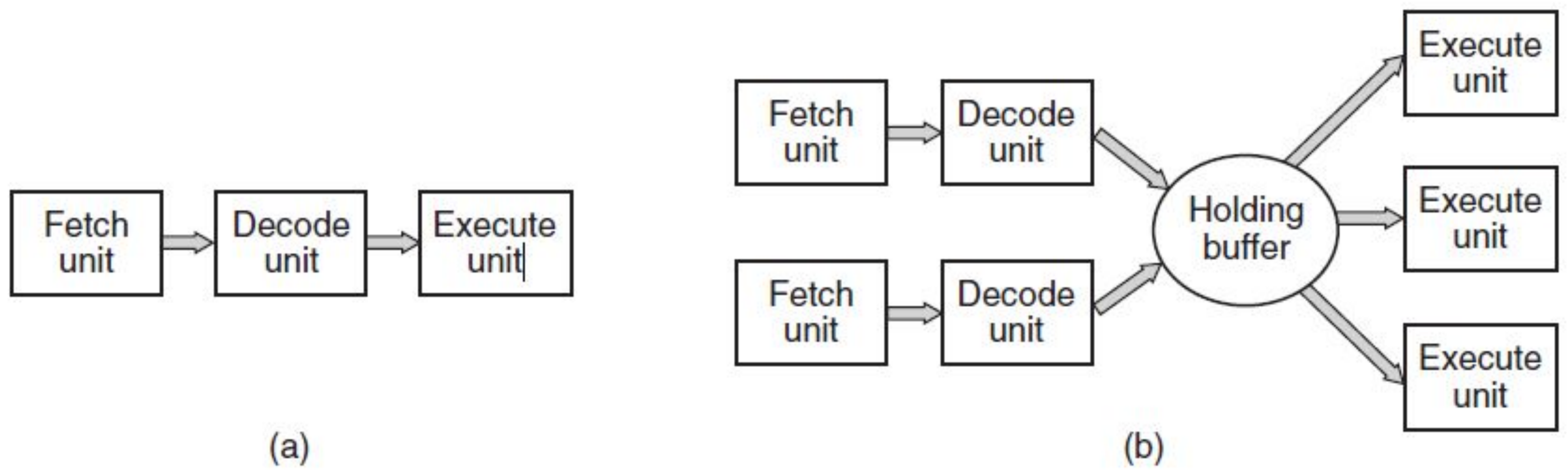


Figure 1-7. (a) A three-stage pipeline. (b) A superscalar CPU.

Memory (1)

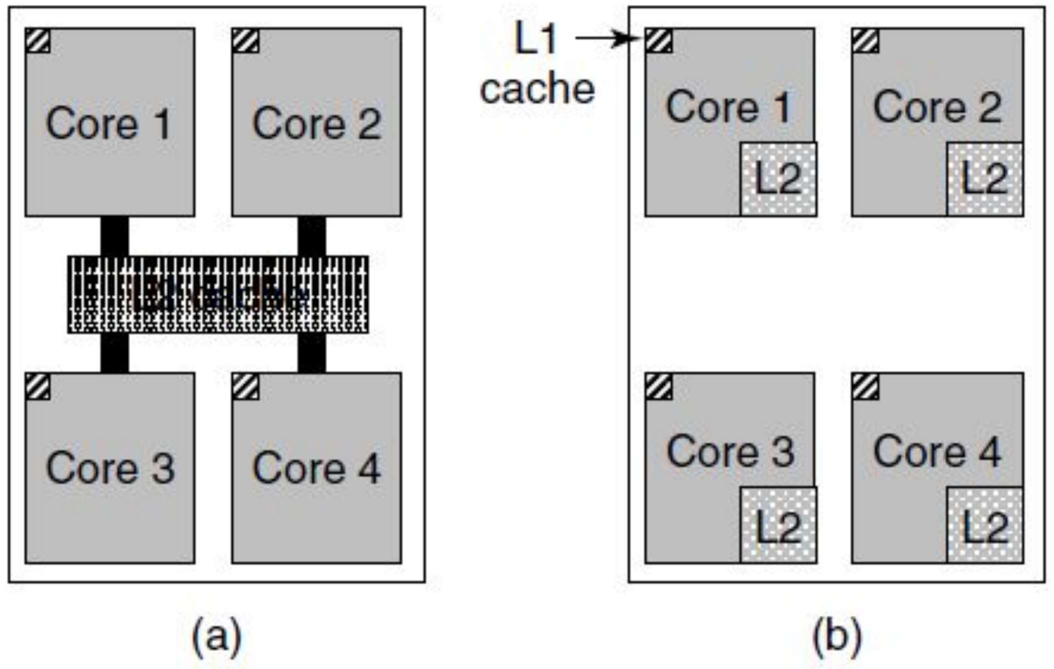


Figure 1-8. (a) A quad-core chip with a shared L2 cache.
(b) A quad-core chip with separate L2 caches.

Memory (2)

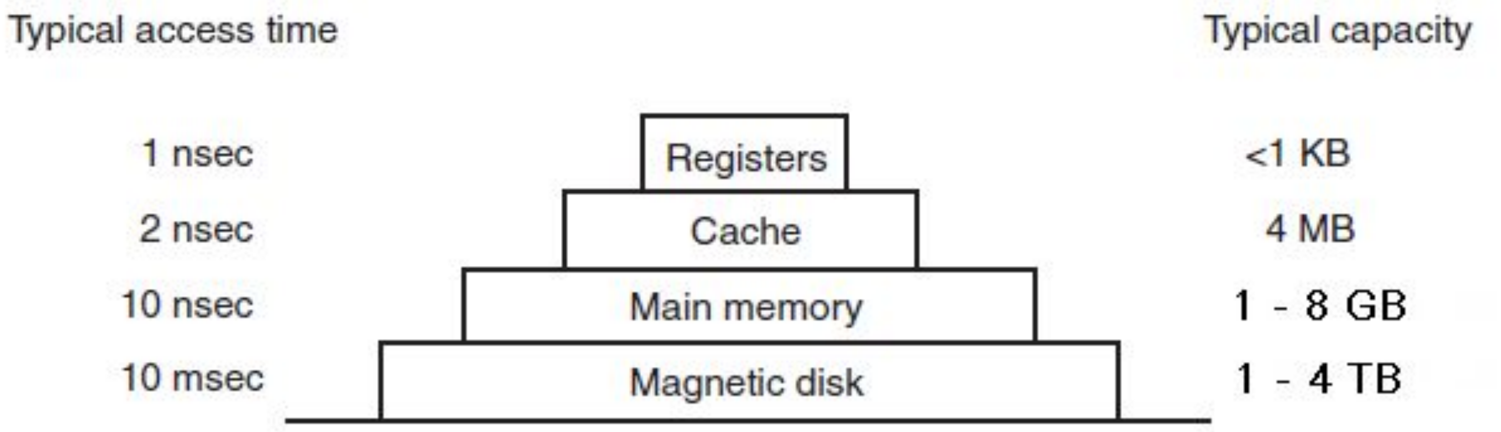


Figure 1-9. A typical memory hierarchy. The numbers are very rough approximations.

Memory (3)

Caching system issues:

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

Disks

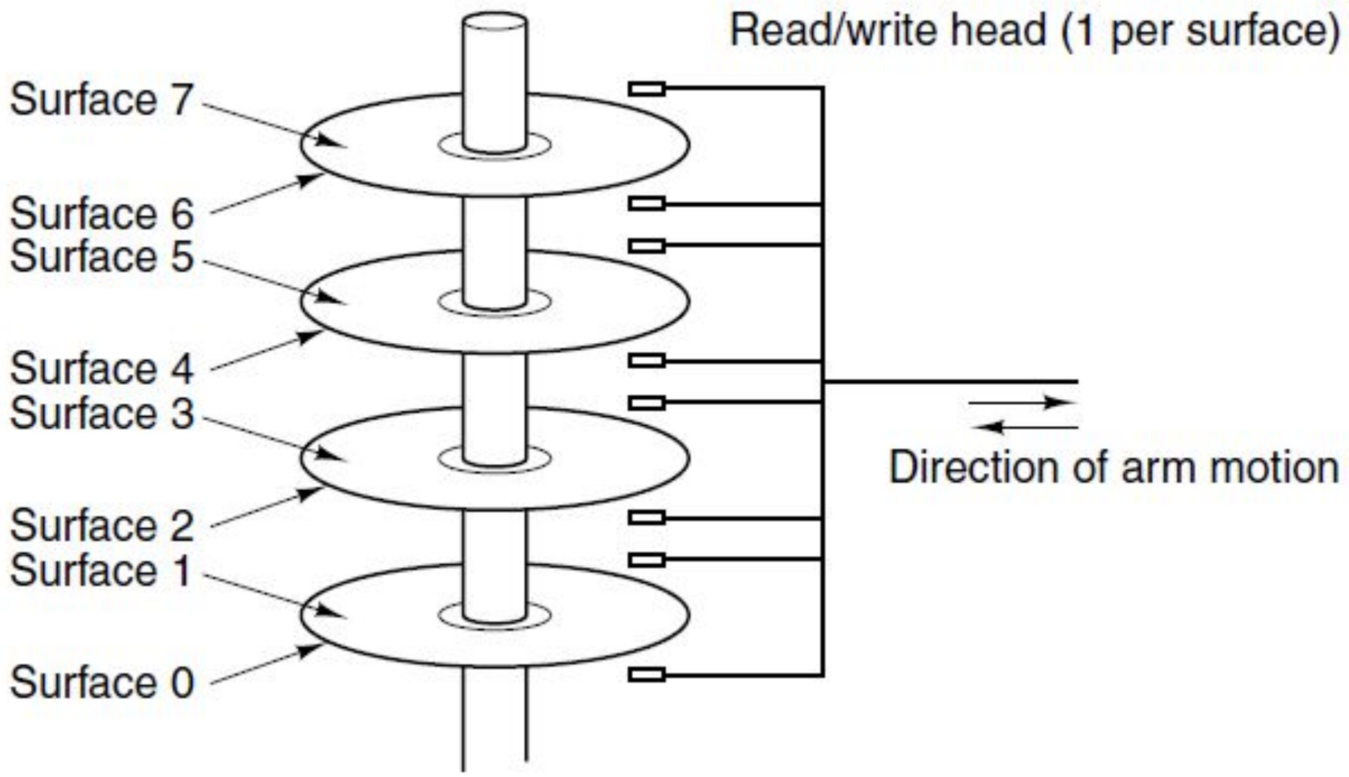


Figure 1-10. Structure of a disk drive.

I/O Devices

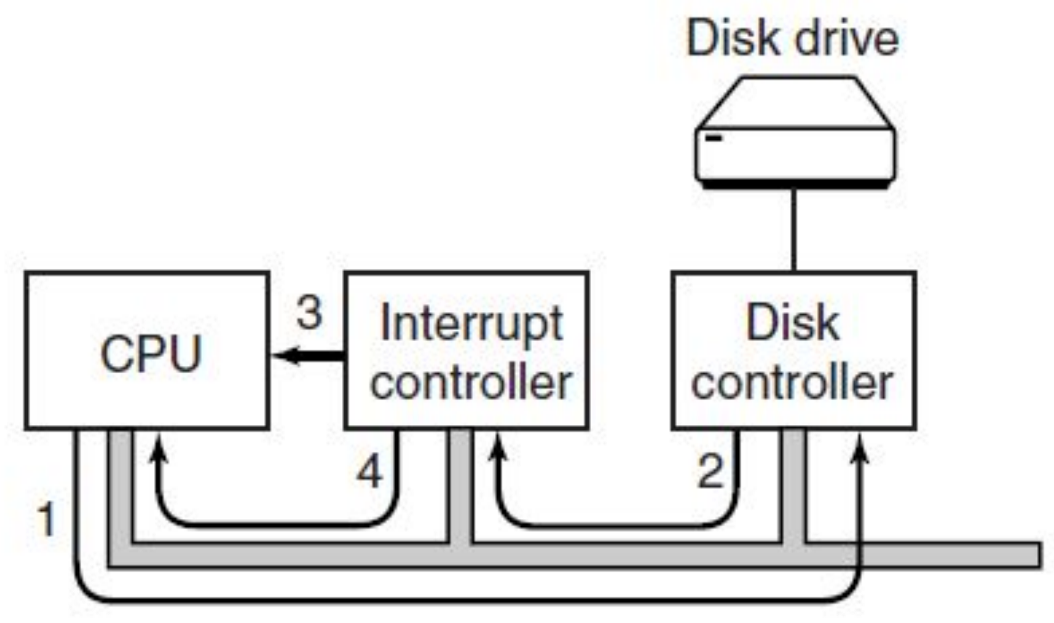


Figure 1-11. (a) The steps in starting an I/O device and getting an interrupt.

I/O Devices

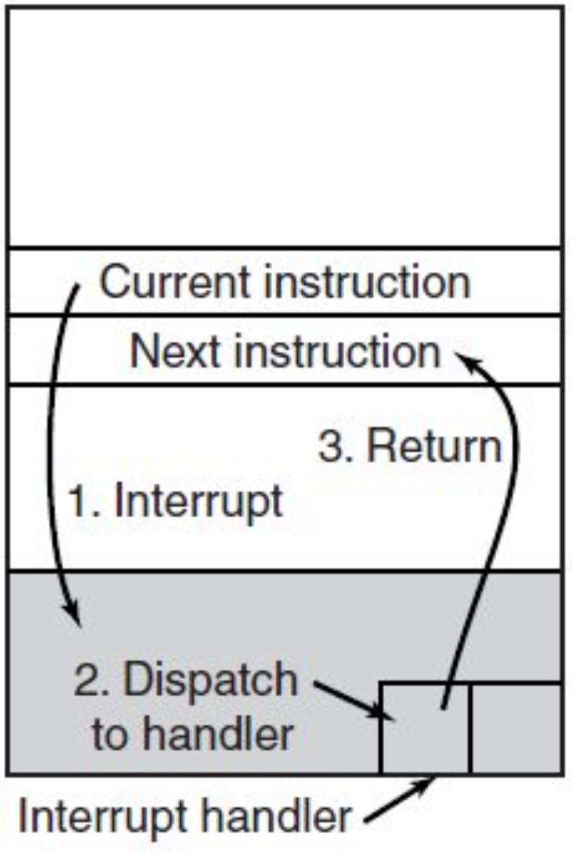


Figure 1-11. (b) Interrupt processing involves taking the interrupt, running the interrupt handler, and returning to the user program.

Buses

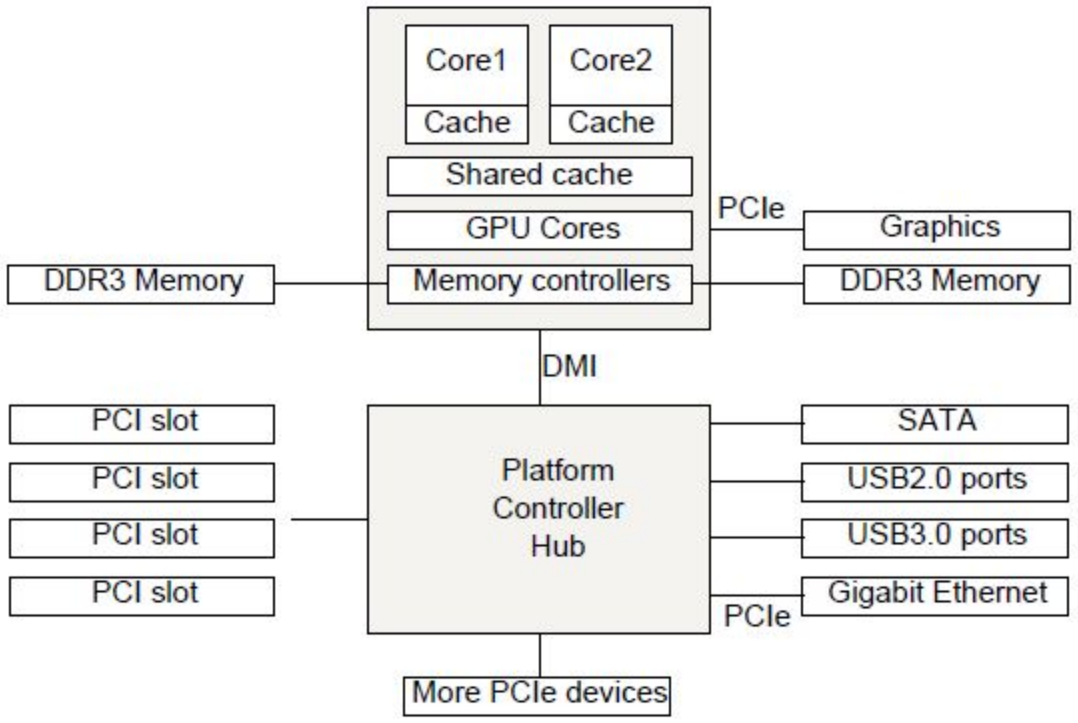


Figure 1-12. The structure of a large x86 system

The Operating System Zoo

- Mainframe Operating Systems
- Server Operating Systems
- Multiprocessor Operating Systems
- Personal Computer Operating Systems
- Handheld Computer Operating Systems
- Embedded Operating Systems
- Sensor Node Operating Systems
- Real-Time Operating Systems
- Smart Card Operating Systems

Processes (1)

- Key concept in all operating systems
- Definition: a program in execution
- Process is associated with an address space
- Also associated with set of resources
- Process can be thought of as a container
  - Holds all information needed to run program

Processes (2)

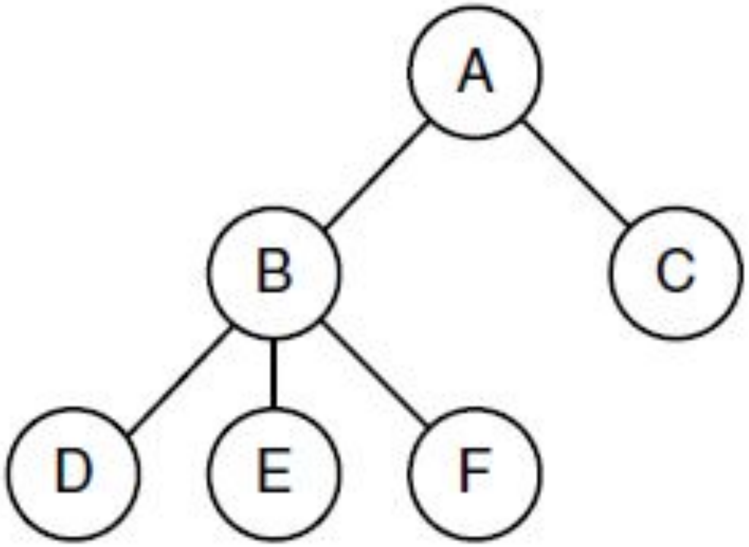


Figure 1-13. A process tree. Process A created two child processes, B and C. Process B created three child processes, D, E, and F.

Files (1)

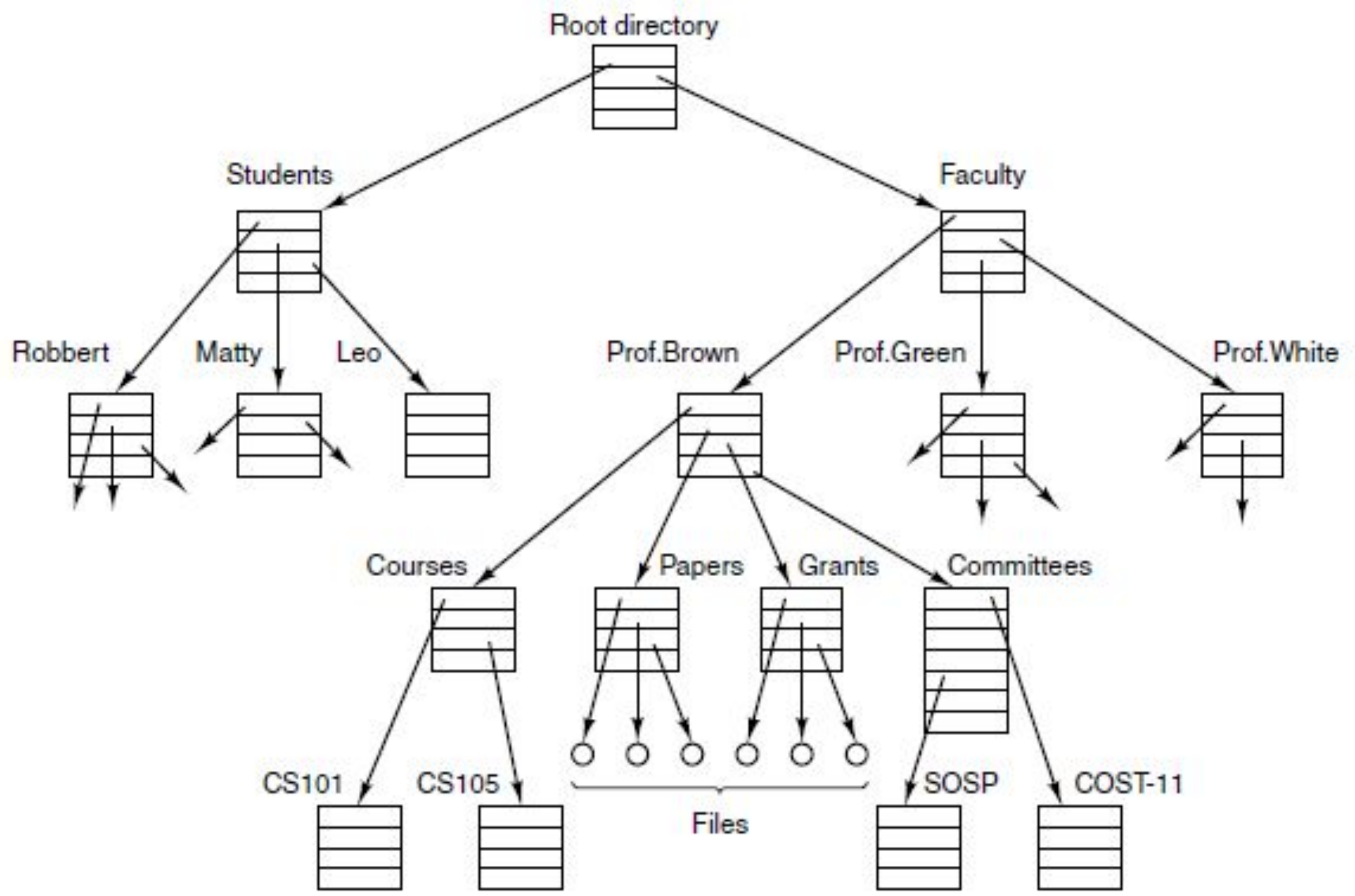


Figure 1-14. A file system for a university department.

Files (2)

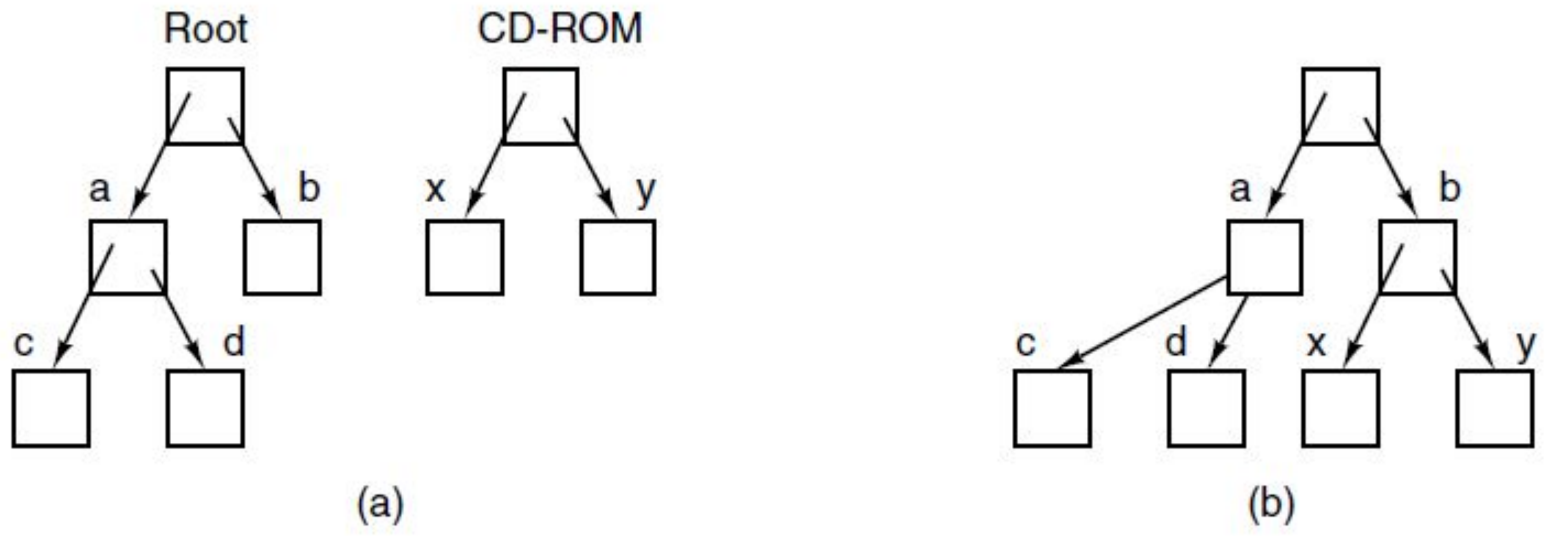


Figure 1-15. (a) Before mounting, the files on the CD-ROM are not accessible. (b) After mounting, they are part of the file hierarchy.

Files (3)

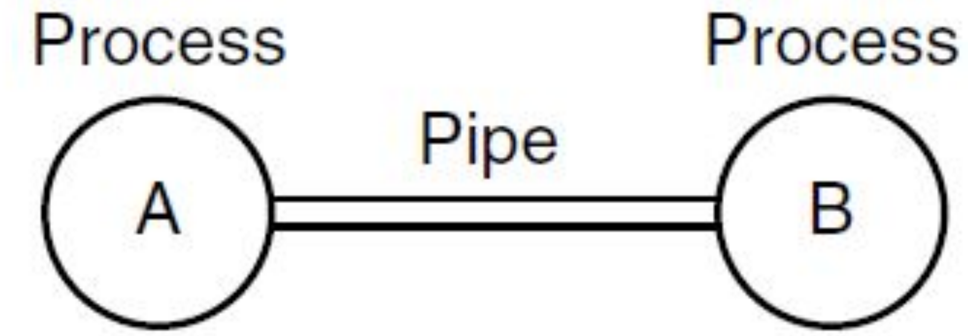


Figure 1-16. Two processes connected by a pipe.

Ontogeny Recapitulates Phylogeny (Embryo repeats the evolution of species)

- Each new “species” of computer
  - Goes through same development as “ancestors”
- Consequence of impermanence
  - Text often looks at “obsolete” concepts
  - Changes in technology may bring them back
- Happens with large memory, protection hardware, disks, virtual memory

System Calls (1)

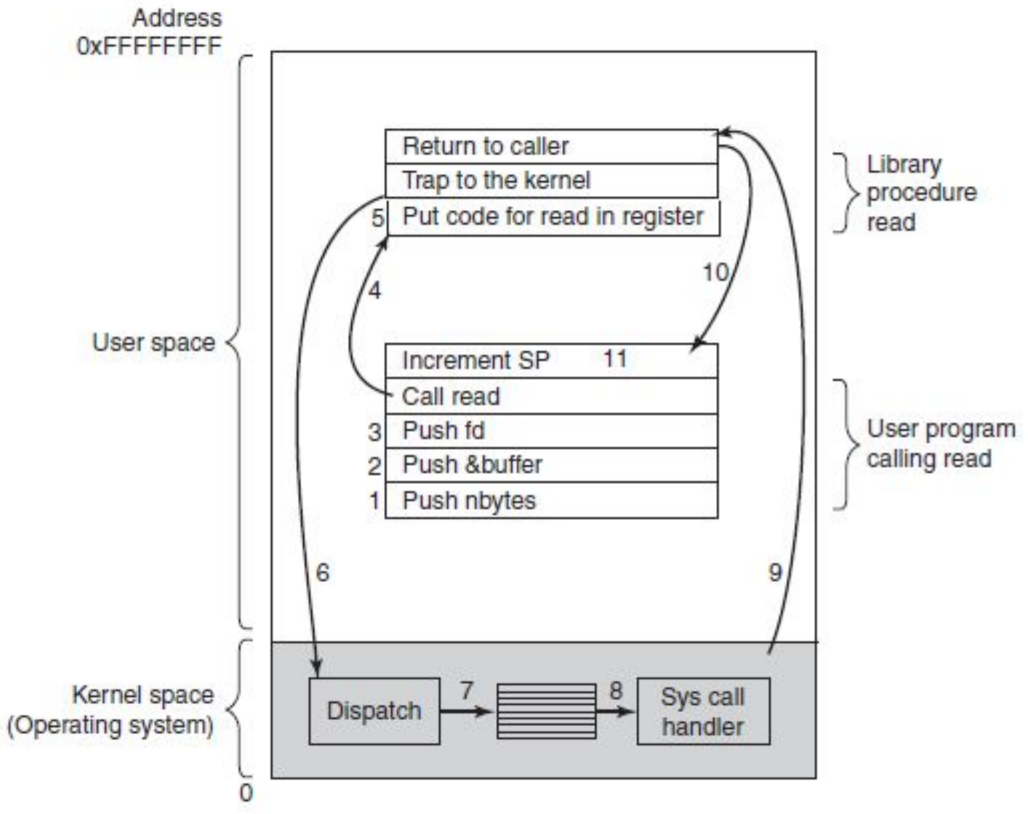
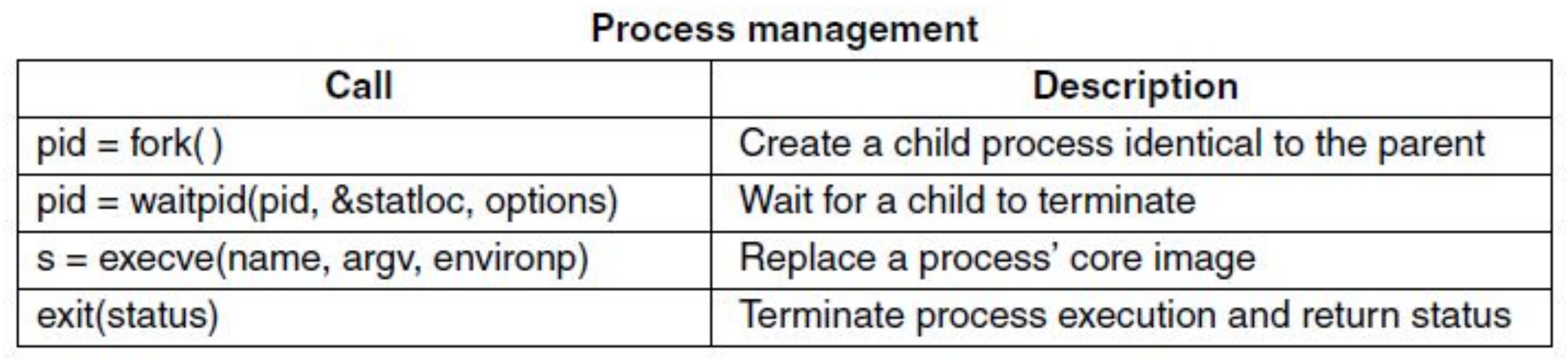


Figure 1-17. The 11 steps in making the system call *read(fd, buffer, nbytes).*

System Calls (2)

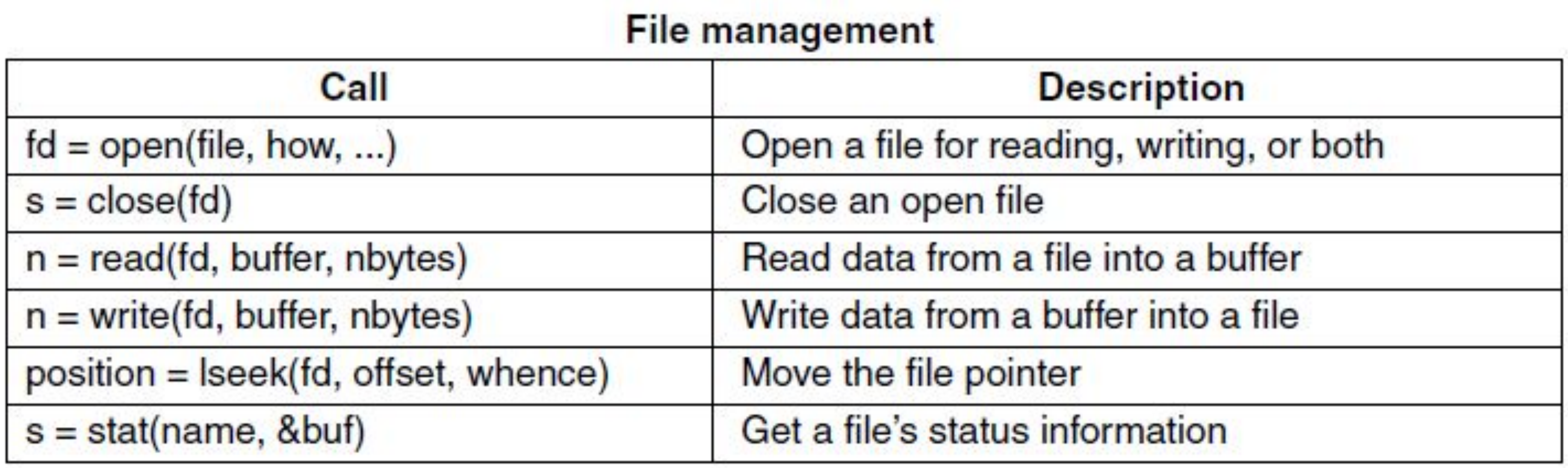


Process management

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. The return code *s* is −1 if an error has occurred. The return codes are as follows: *pid* is a process id, *fd* is a file descriptor, *n* is a byte count, *position* is an offset within the file, and *seconds* is the elapsed time.

System Calls (3)

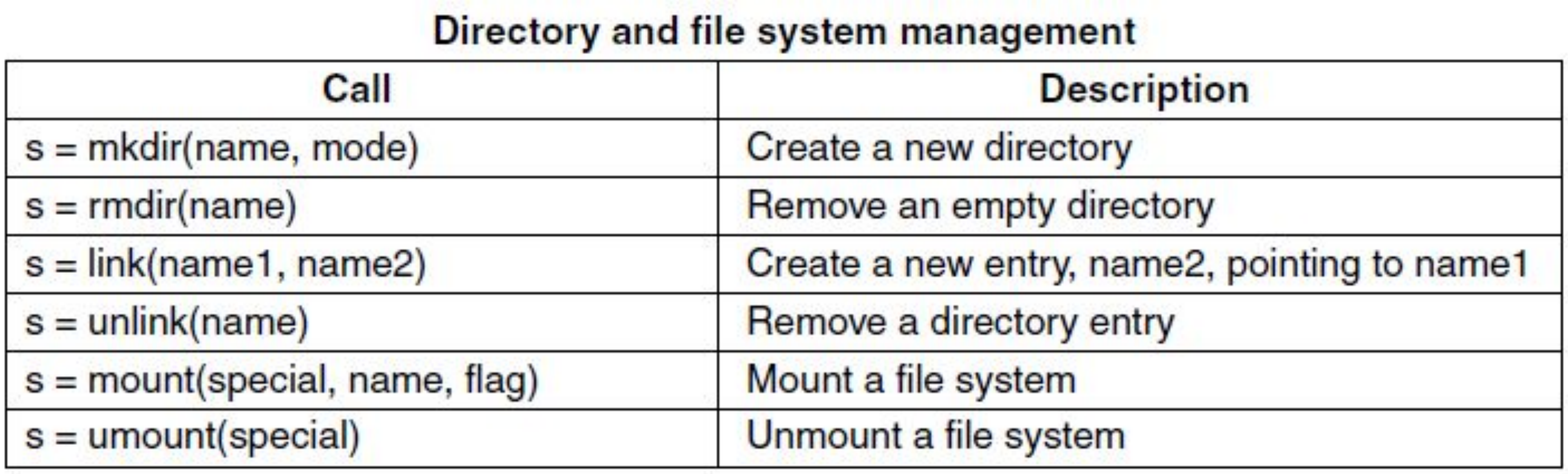


File management

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. The return code *s* is −1 if an error has occurred. The return codes are as follows: *pid* is a process id, *fd* is a file descriptor, *n* is a byte count, *position* is an offset within the file, and *seconds* is the elapsed time.

System Calls (4)



Directory and file system management

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. The return code *s* is −1 if an error has occurred. The return codes are as follows: *pid* is a process id, *fd* is a file descriptor, *n* is a byte count, *position* is an offset within the file, and *seconds* is the elapsed time.

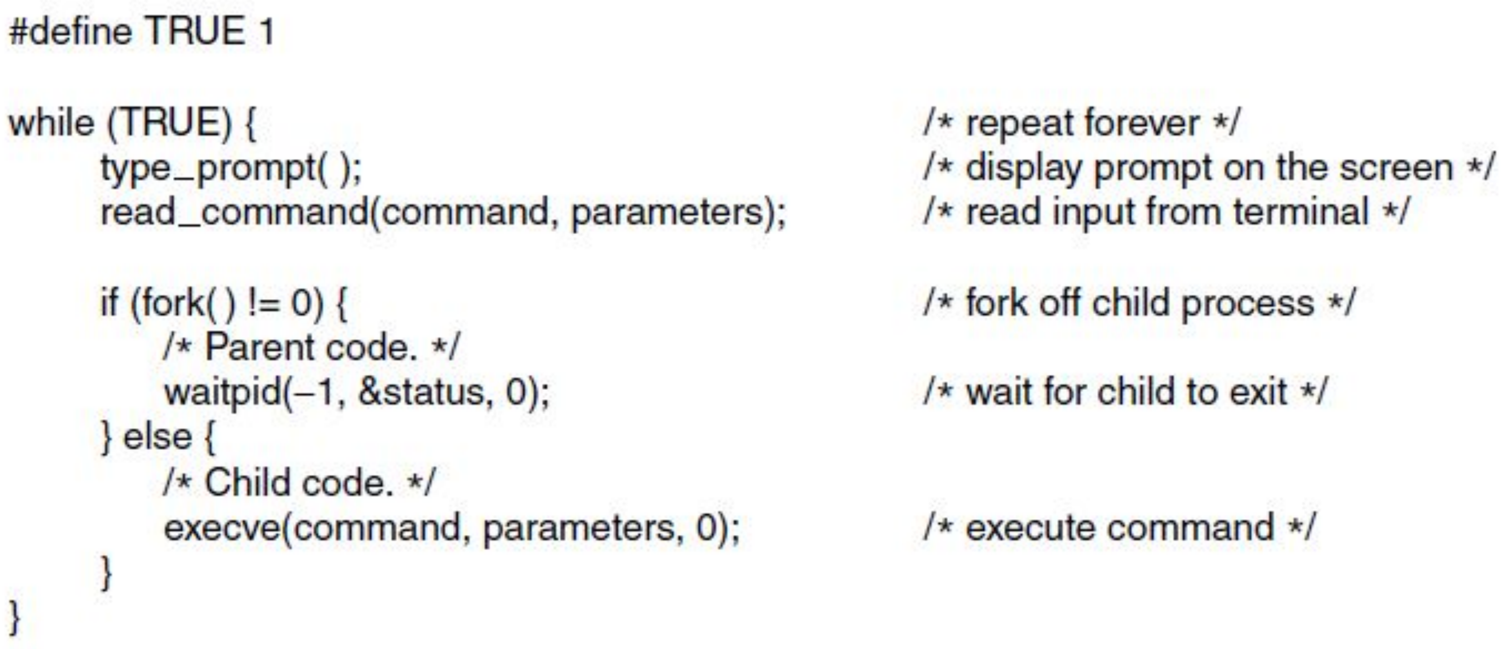
System Calls (5)

Miscellaneous

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. The return code *s* is −1 if an error has occurred. The return codes are as follows: *pid* is a process id, *fd* is a file descriptor, *n* is a byte count, *position* is an offset within the file, and *seconds* is the elapsed time.

System Calls for Process Management



```
#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. Throughout this book, *TRUE* is assumed to be defined as 1.

System Calls for File Management

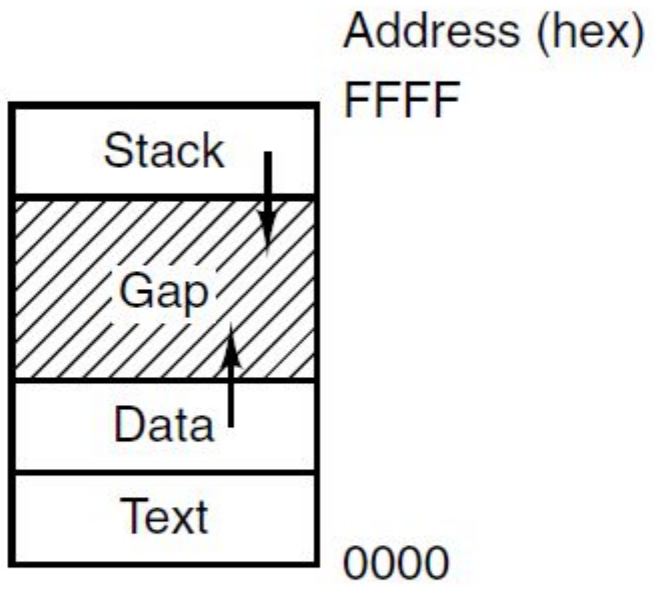


Figure 1-20. Processes have three segments: text, data, and stack

System Calls for Directory Management (1)

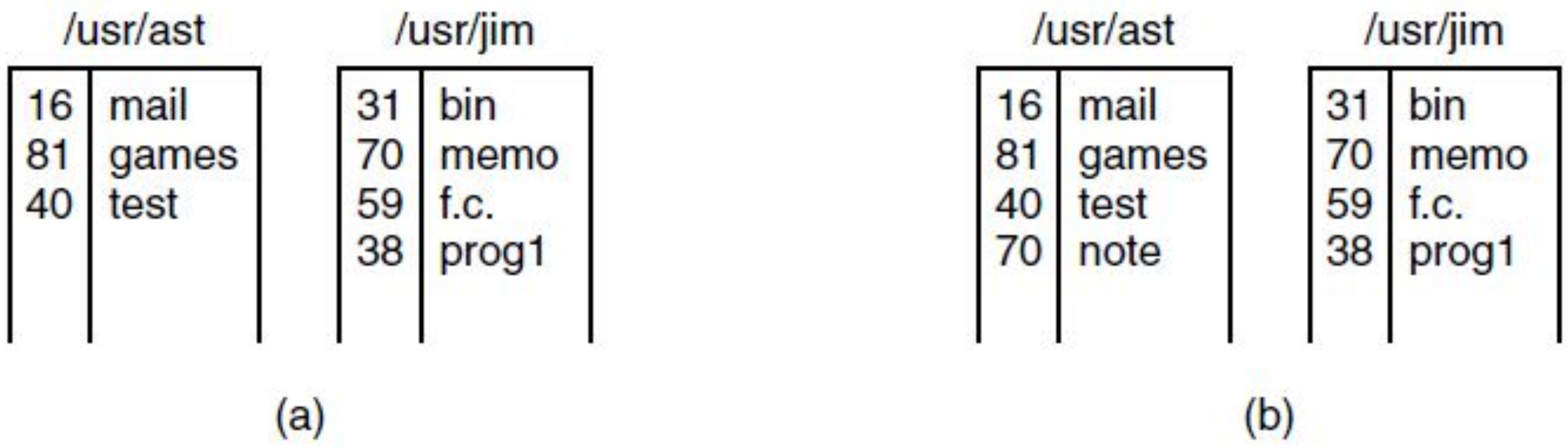
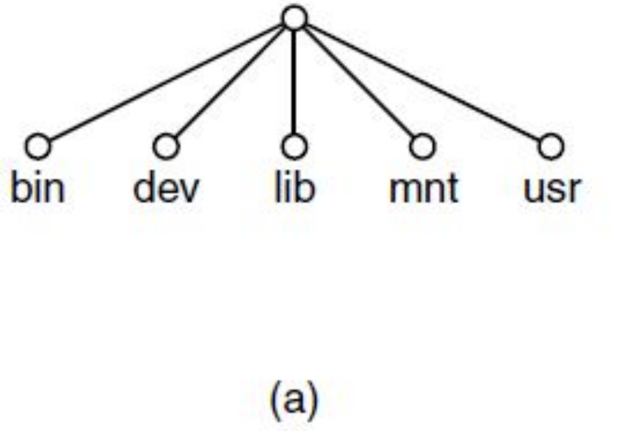


Figure 1-21. (a) Two directories before linking *usr/jim/memo* to *ast*'s directory. (b) The same directories after linking.

System Calls for Directory Management (2)



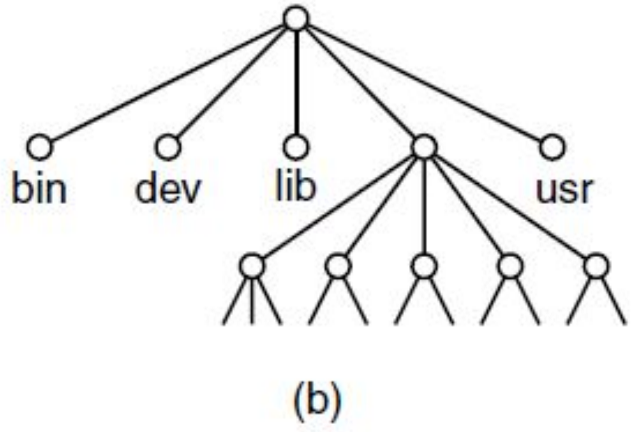
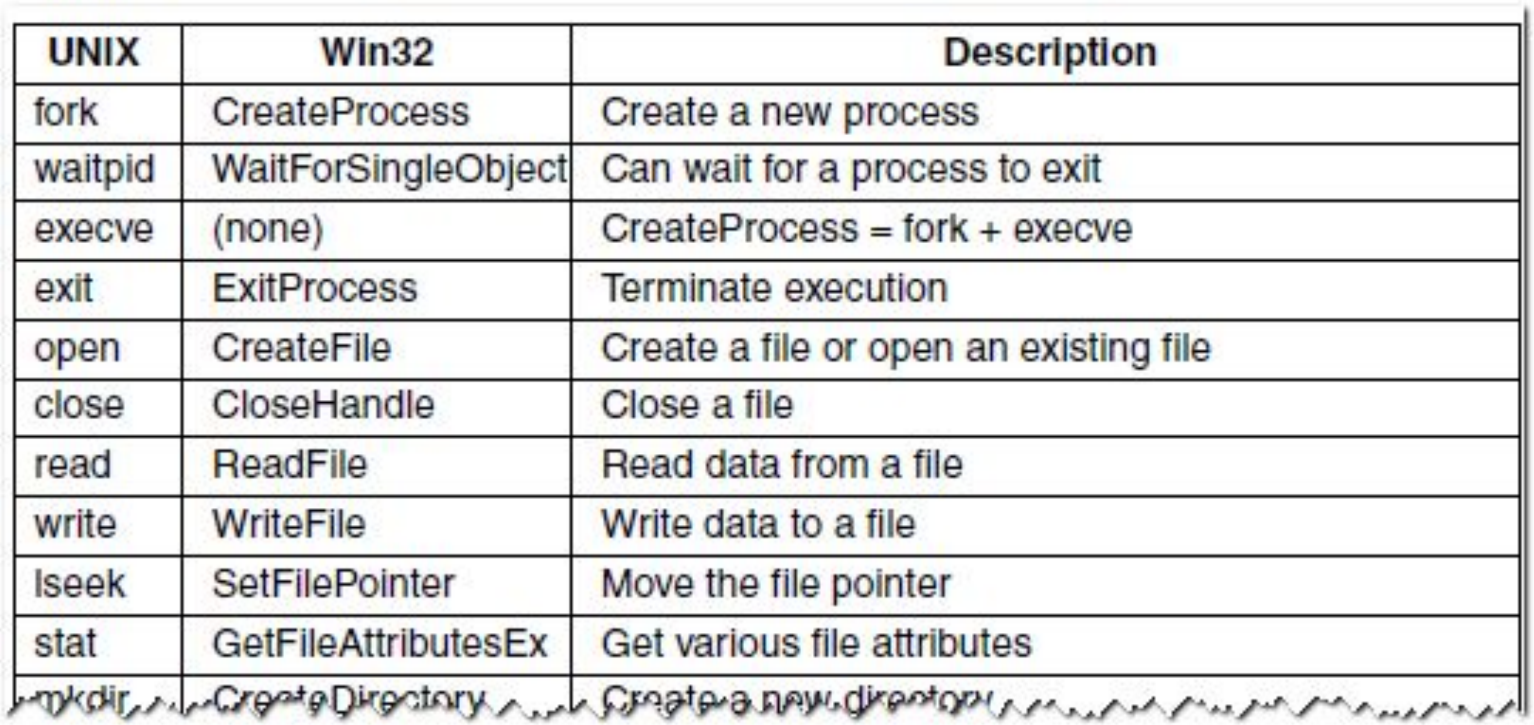


Figure 1-22. (a) File system before the mount.
(b) File system after the mount.

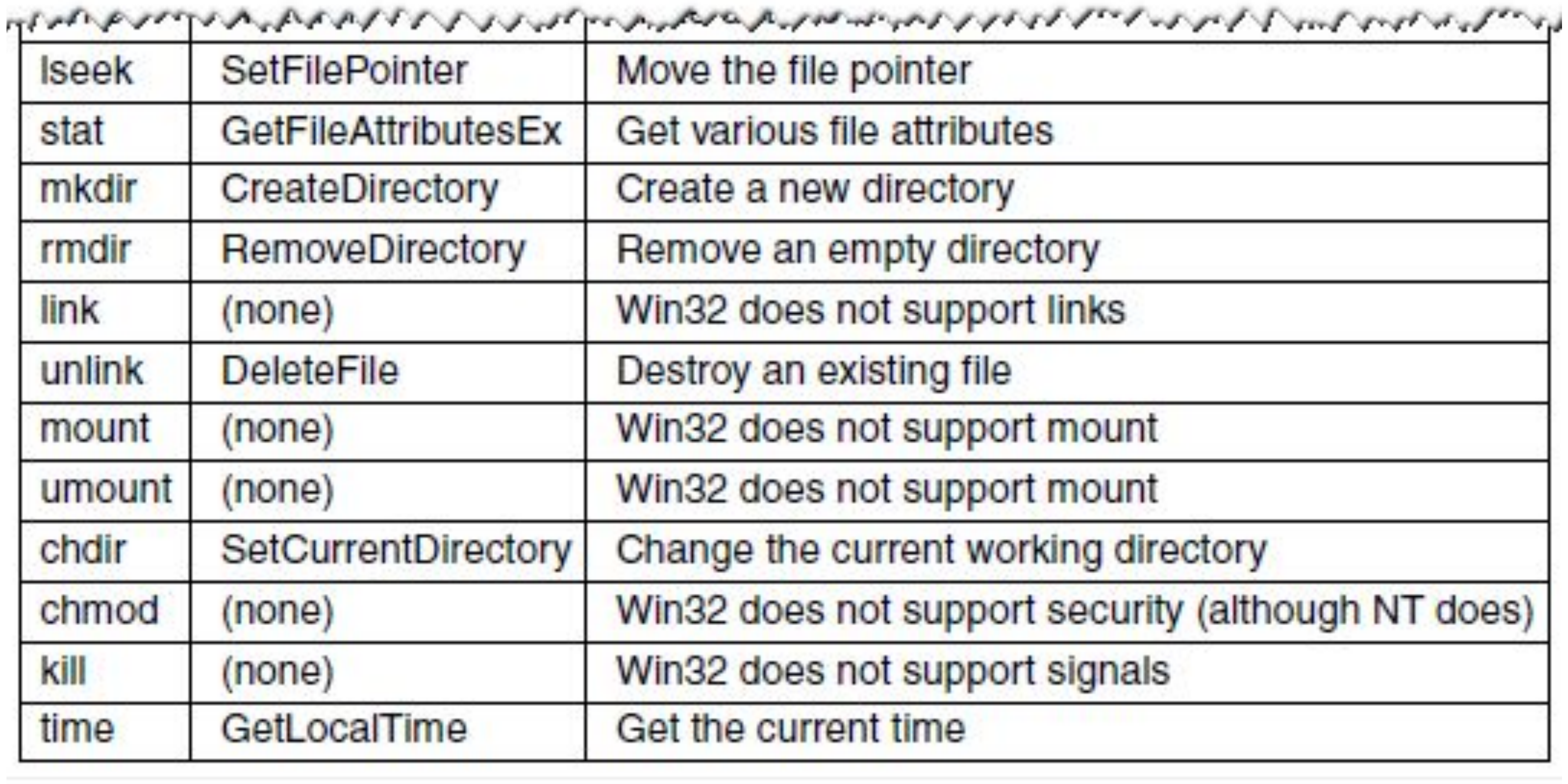
The Windows Win32 API (1)



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

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

The Windows Win32 API (2)



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.

Monolithic Systems (1)

Basic structure of OS

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

Monolithic Systems (2)

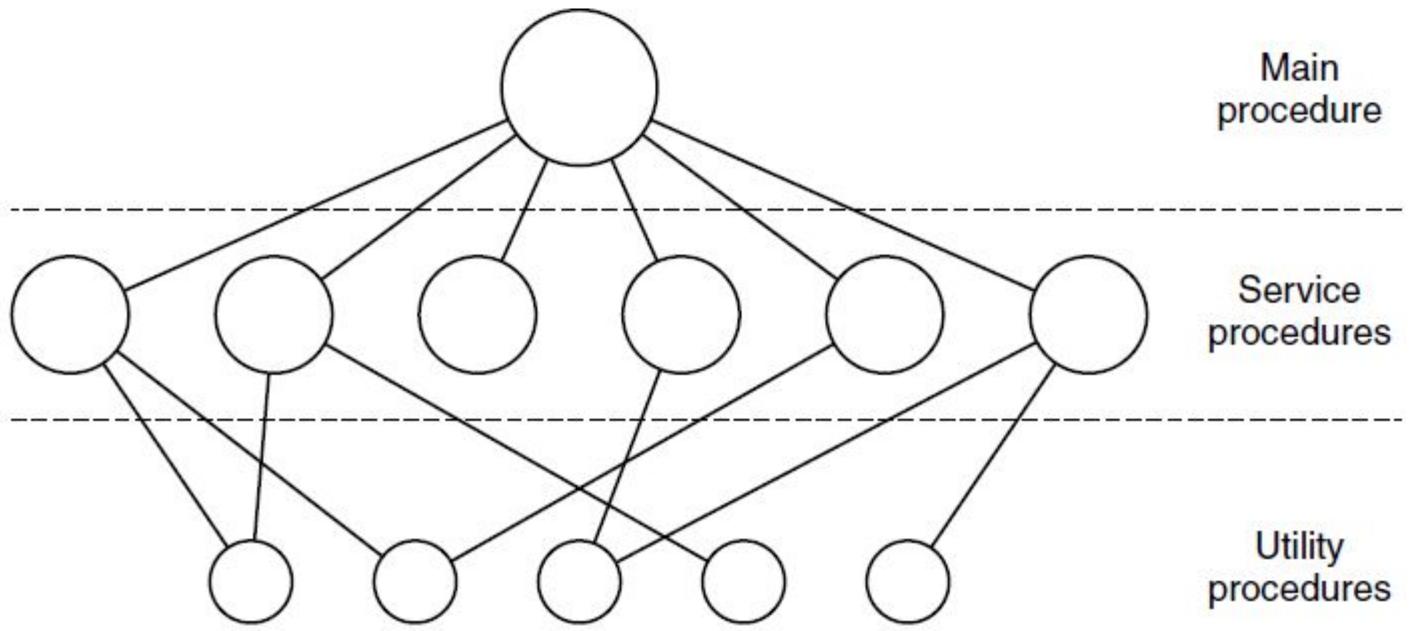
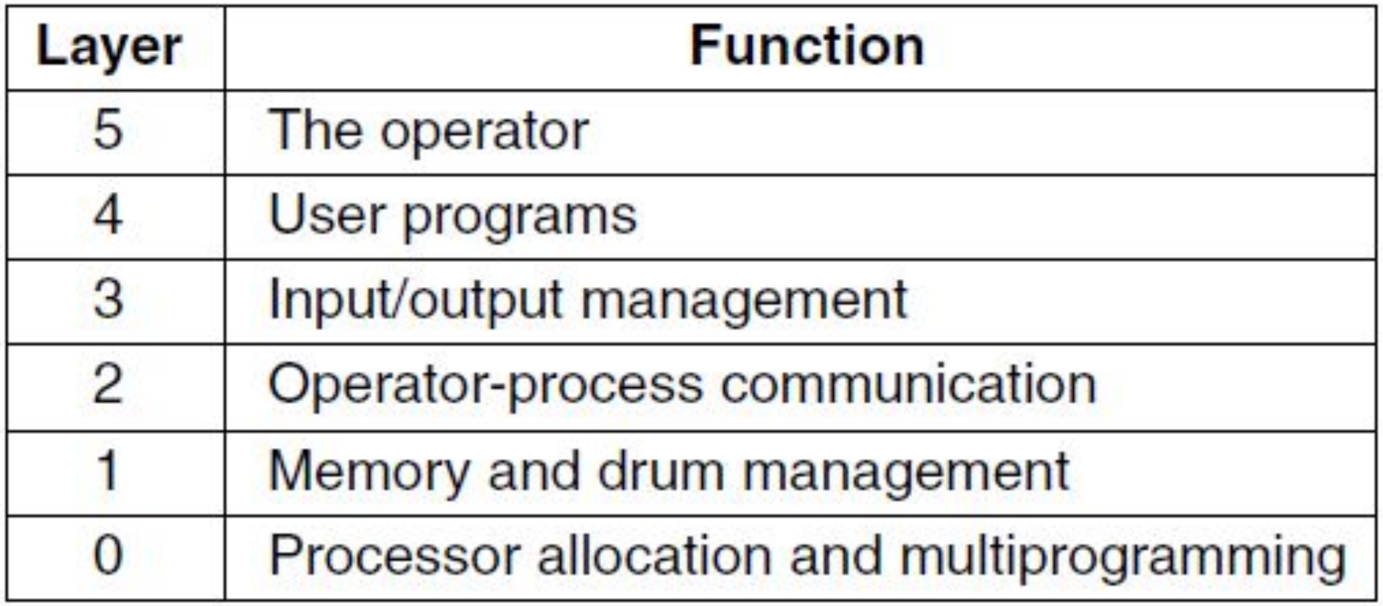


Figure 1-24. A simple structuring model for a monolithic system.

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

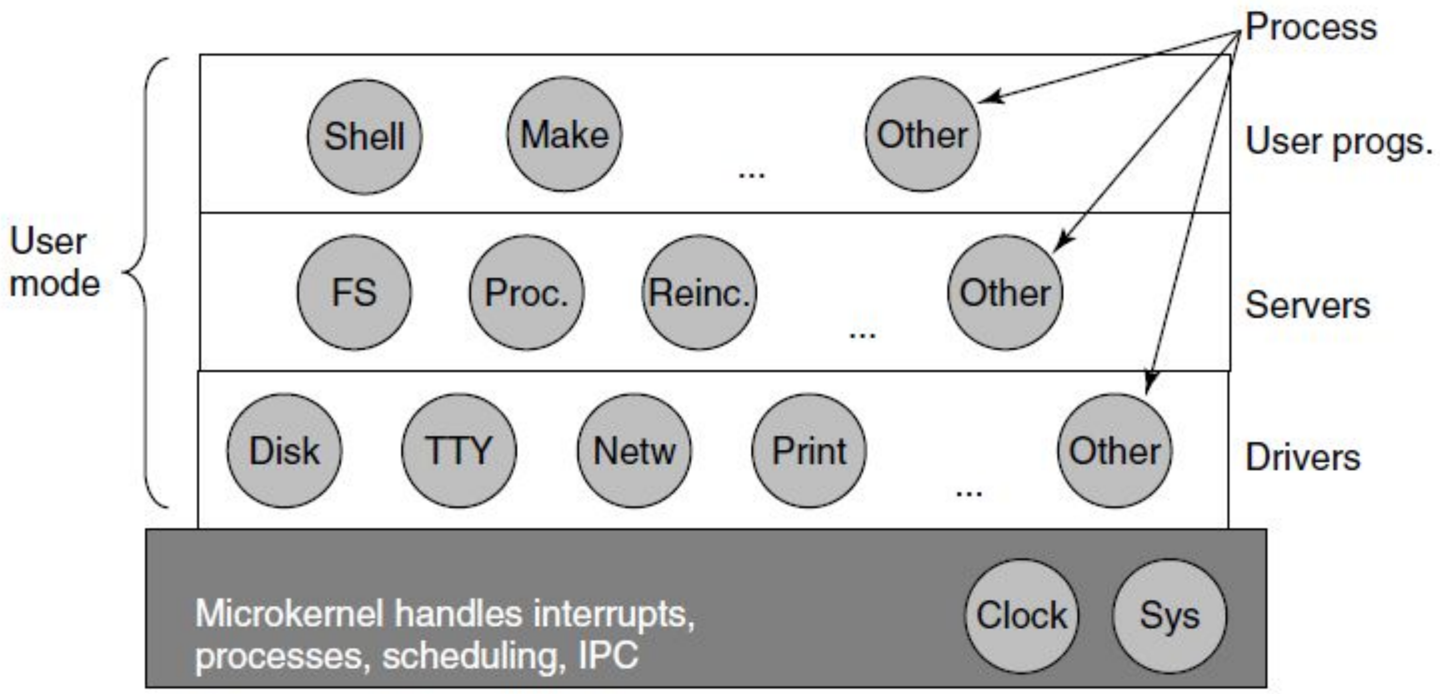


Figure 1-26. Simplified structure of the MINIX 3 system.

Client-Server Model

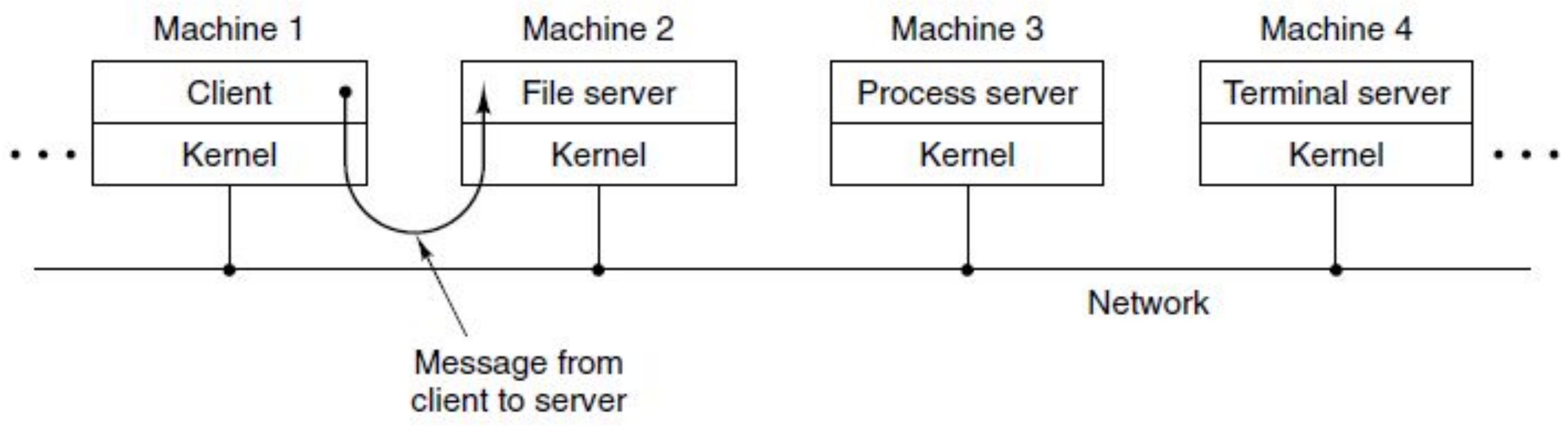


Figure 1-27. The client-server model over a network.

Virtual Machines

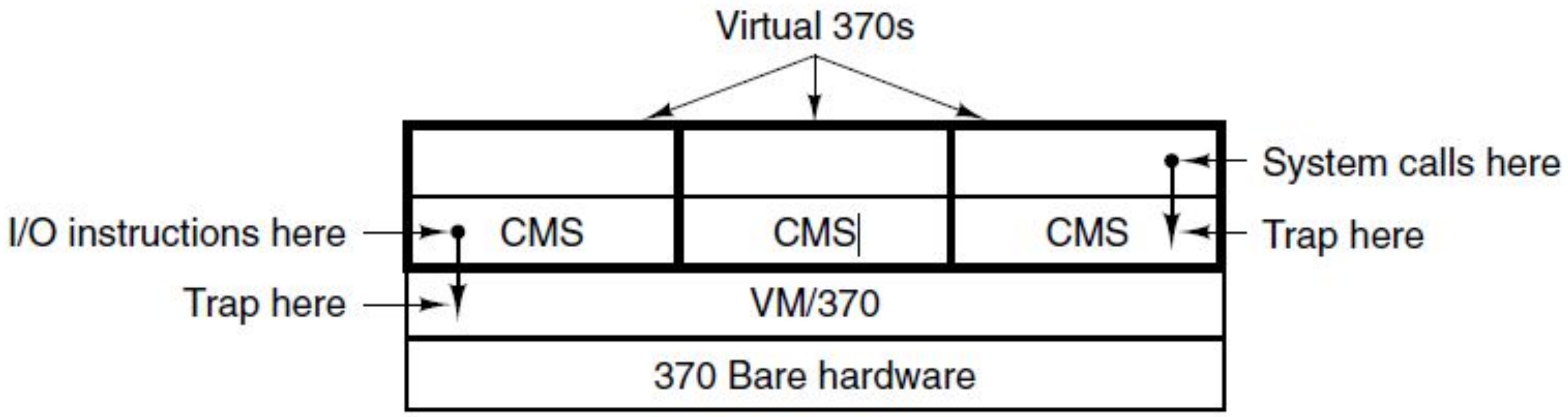


Figure 1-28. The structure of VM/370 with CMS.

Virtual Machines Rediscovered

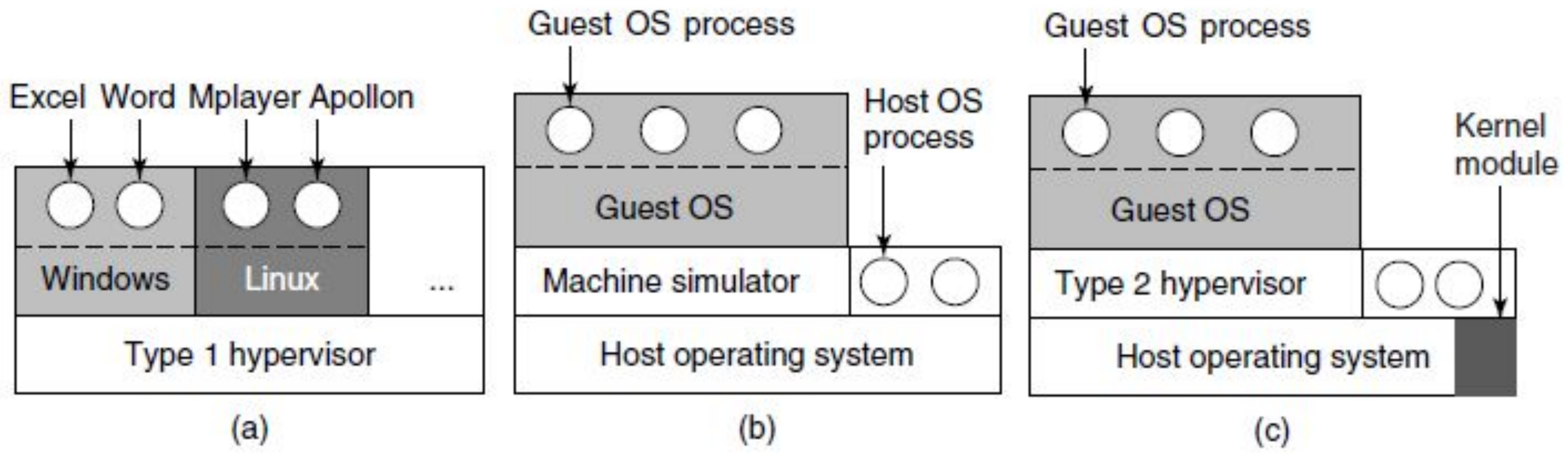


Figure 1-29. (a) A type 1 hypervisor. (b) A pure type 2 hypervisor. (c) A practical type 2 hypervisor.

Large Programming Projects

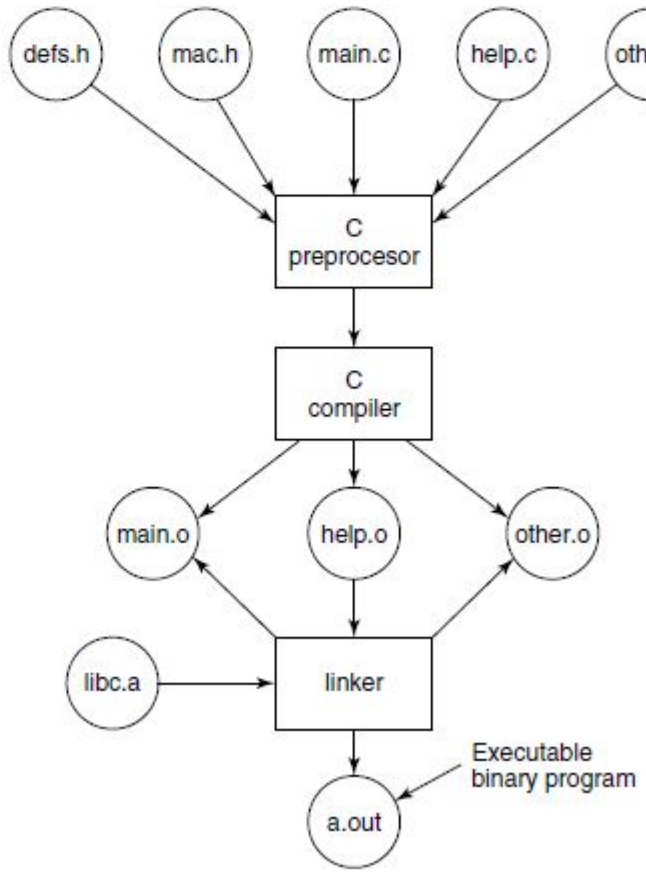
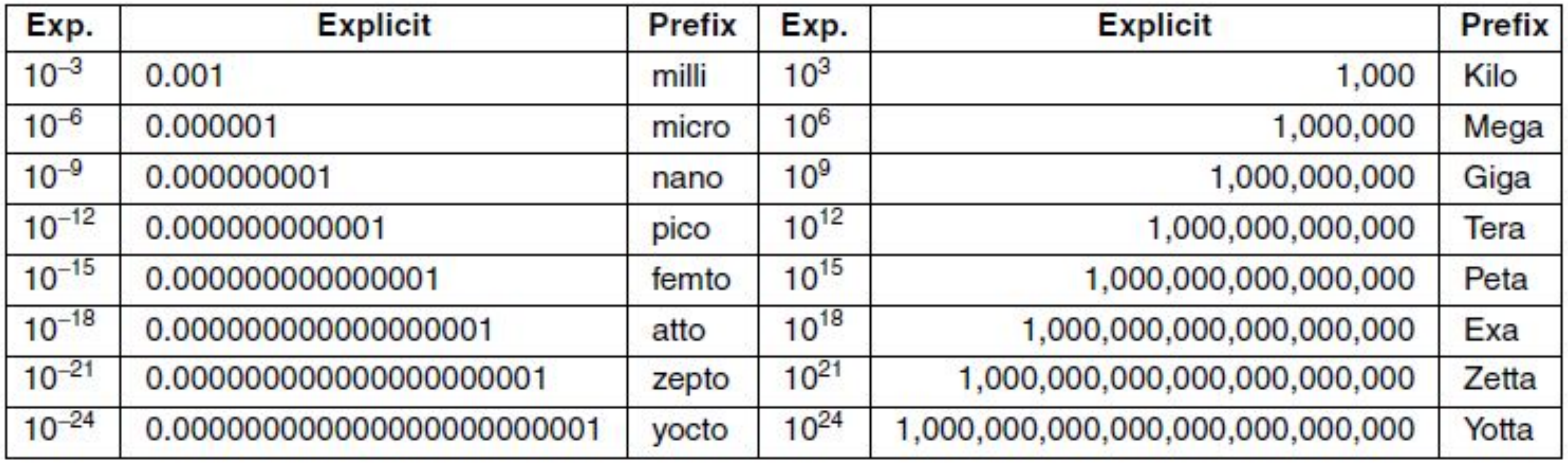


Figure 1-30. The process of compiling C and header files to make an executable.

Metric Units



Exp.	Explicit	Prefix	Exp.	Explicit	Prefix
10^{-3}	0.001	milli	10^{3}	1,000	Kilo
10^{-6}	0.000001	micro	10^{6}	1,000,000	Mega
10^{-9}	0.000000001	nano	10^{9}	1,000,000,000	Giga
10^{-12}	0.000000000001	pico	10^{12}	1,000,000,000,000	Tera
10^{-15}	0.000000000000001	femto	10^{15}	1,000,000,000,000,000	Peta
10^{-18}	0.000000000000000001	atto	10^{18}	1,000,000,000,000,000,000	Exa
10^{-21}	0.000000000000000000001	zepto	10^{21}	1,000,000,000,000,000,000,000	Zetta
10^{-24}	0.000000000000000000000001	yocto	10^{24}	1,000,000,000,000,000,000,000,000	Yotta

Figure 1-31. The principal metric prefixes.

End

Chapter 1