1

INTRODUCTION

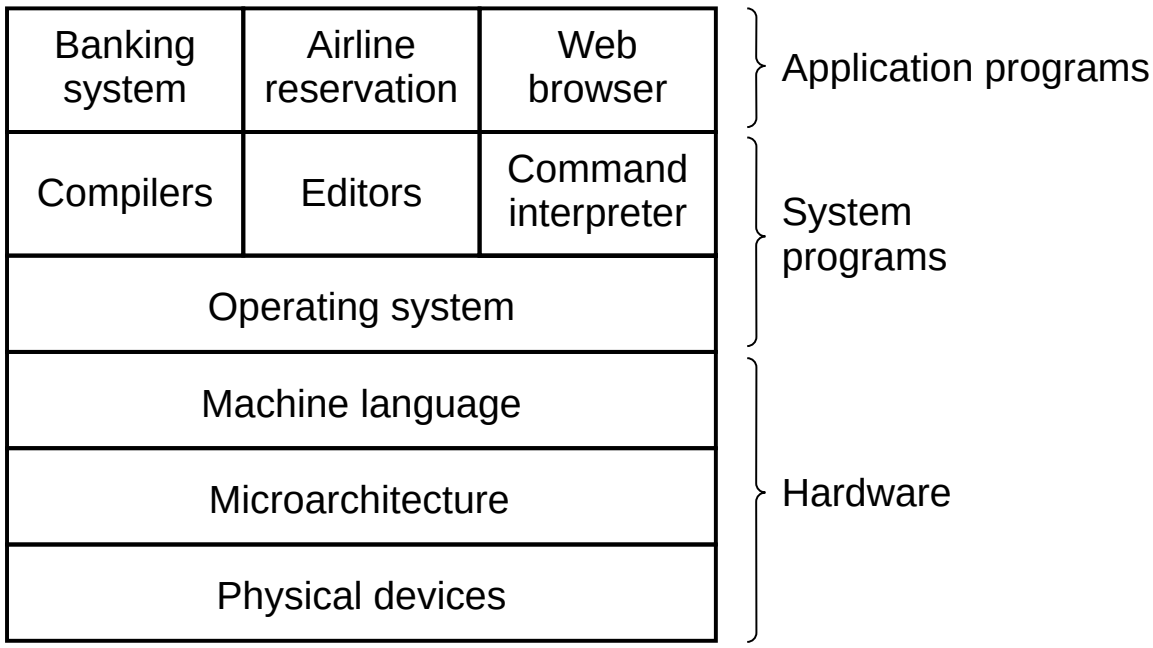

Fig. 1-1. A computer system consists of hardware, system programs, and application programs.

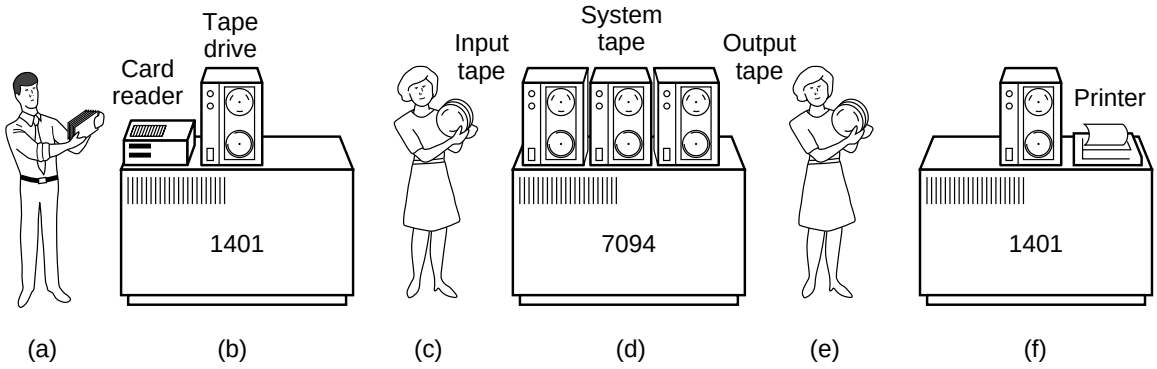

Fig. 1-2. An early batch system. (a) Programmers bring cards to 1401. (b) 1401 reads batch of jobs onto tape. (c) Operator carries input tape to 7094. (d) 7094 does computing. (e) Operator carries output tape to 1401. (f) 1401 prints output.

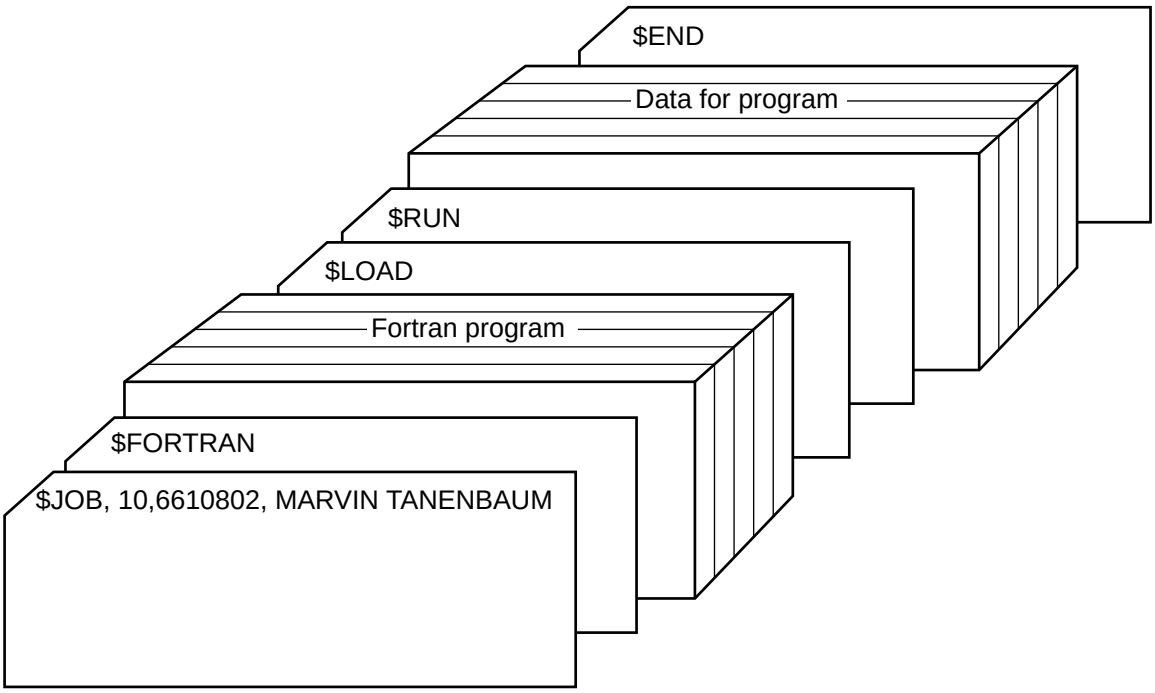

Fig. 1-3. Structure of a typical FMS job.

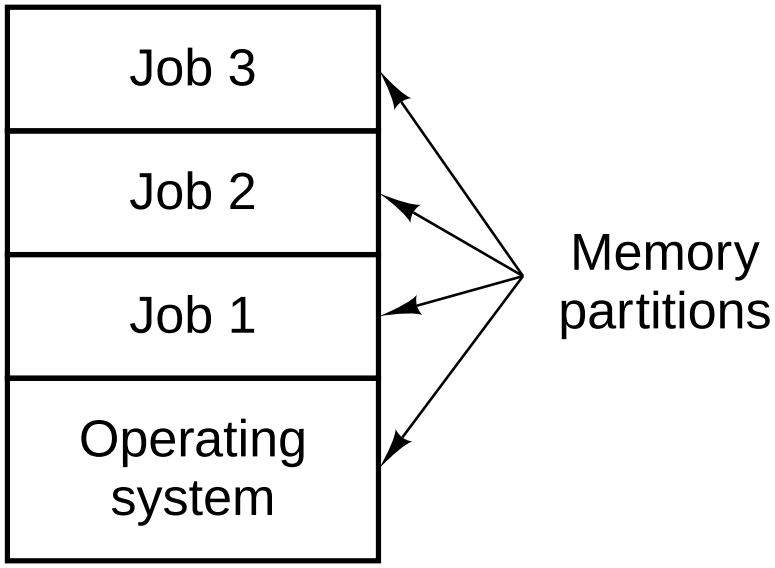

Fig. 1-4. A multiprogramming system with three jobs in memory.

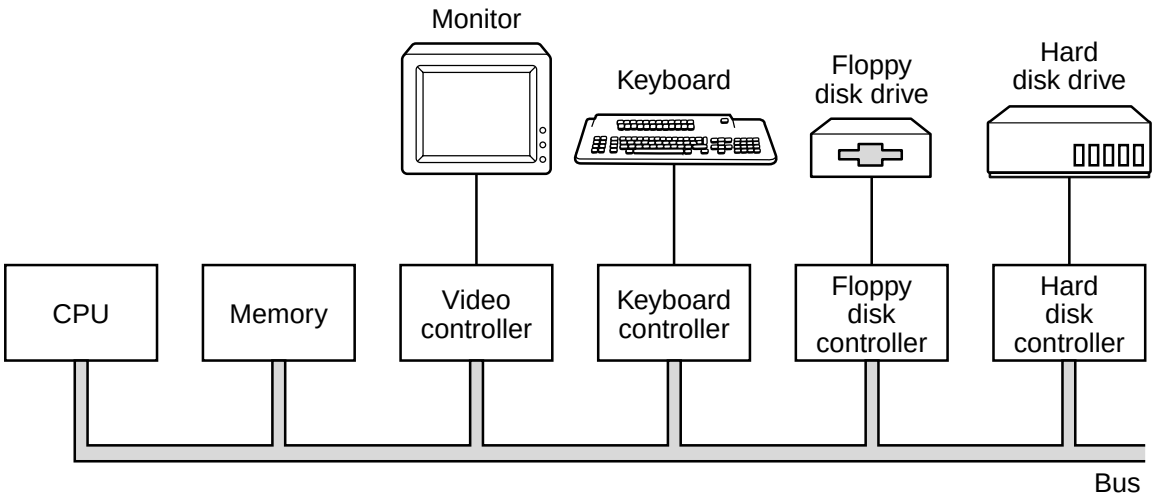

Fig. 1-5. Some of the components of a simple personal computer.

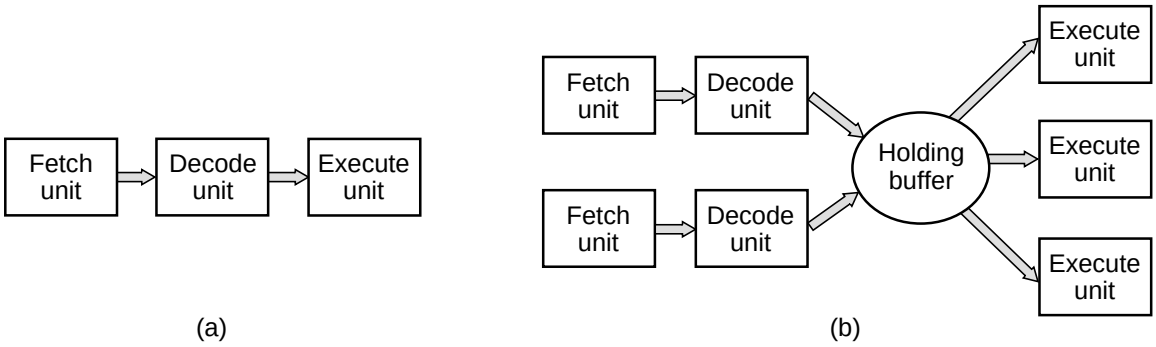

Fig. 1-6. (a) A three-stage pipeline. (b) A superscalar CPU.

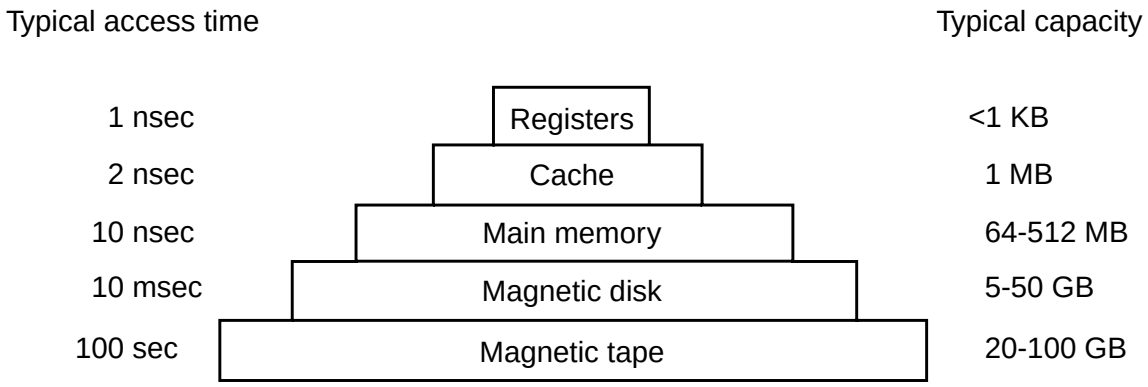

Fig. 1-7. A typical memory hierarchy. The numbers are very rough approximations.

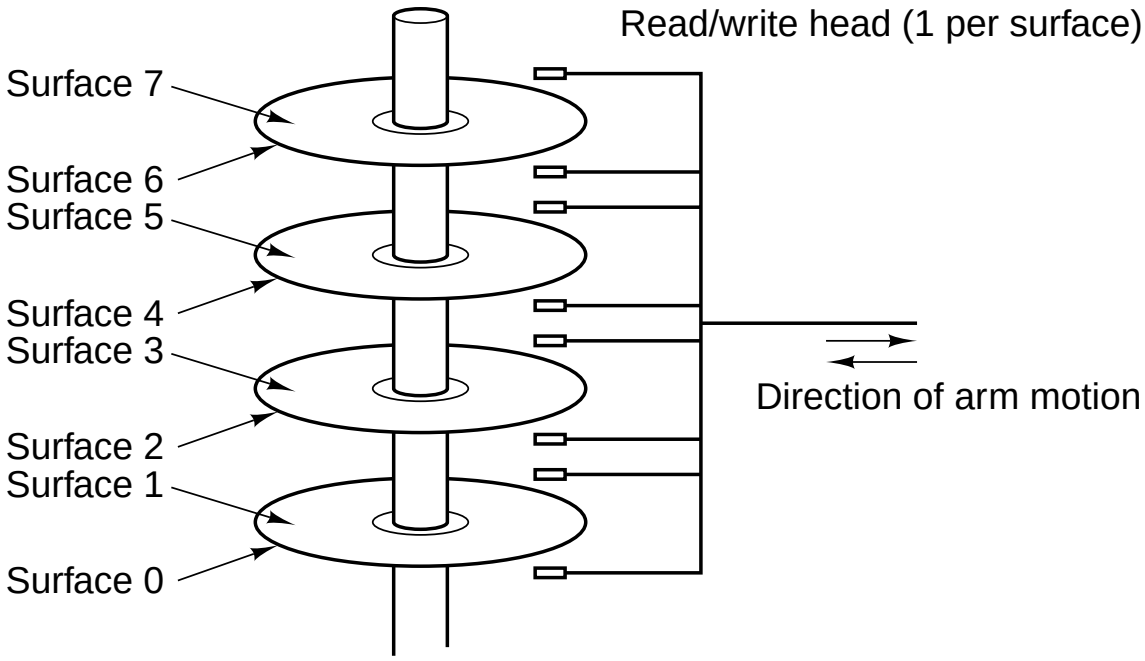

Fig. 1-8. Structure of a disk drive.

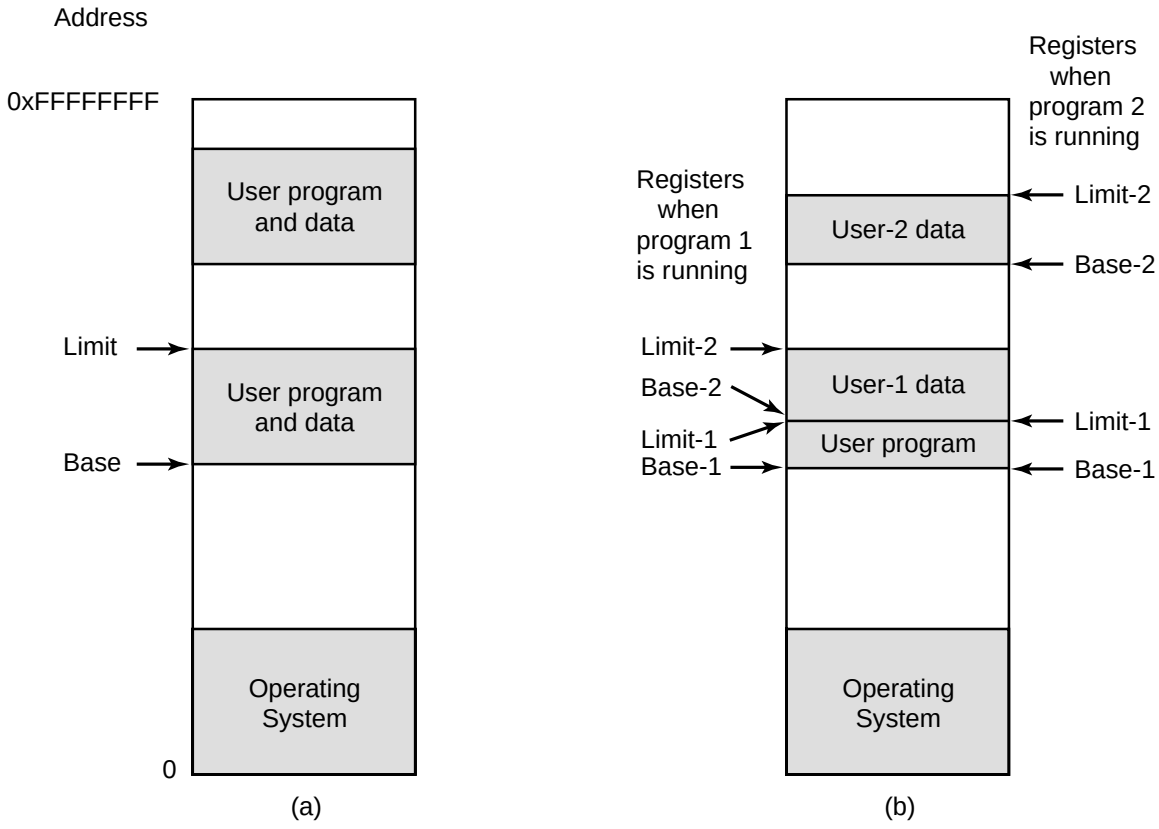

Fig. 1-9. (a) Use of one base-limit pair. The program can access memory between the base and the limit. (b) Use of two base-limit pairs. The program code is between Base-1 and Limit-1 whereas the data are between Base-2 and Limit-2.

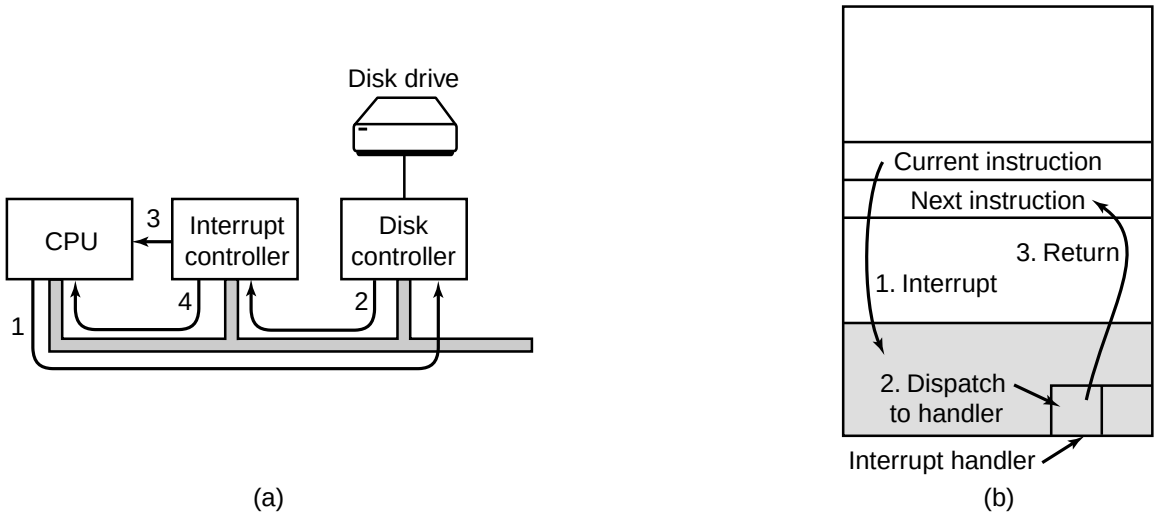

Fig. 1-10. (a) The steps in starting an I/O device and getting an interrupt. (b) Interrupt processing involves taking the interrupt, running the interrupt handler, and returning to the user program.

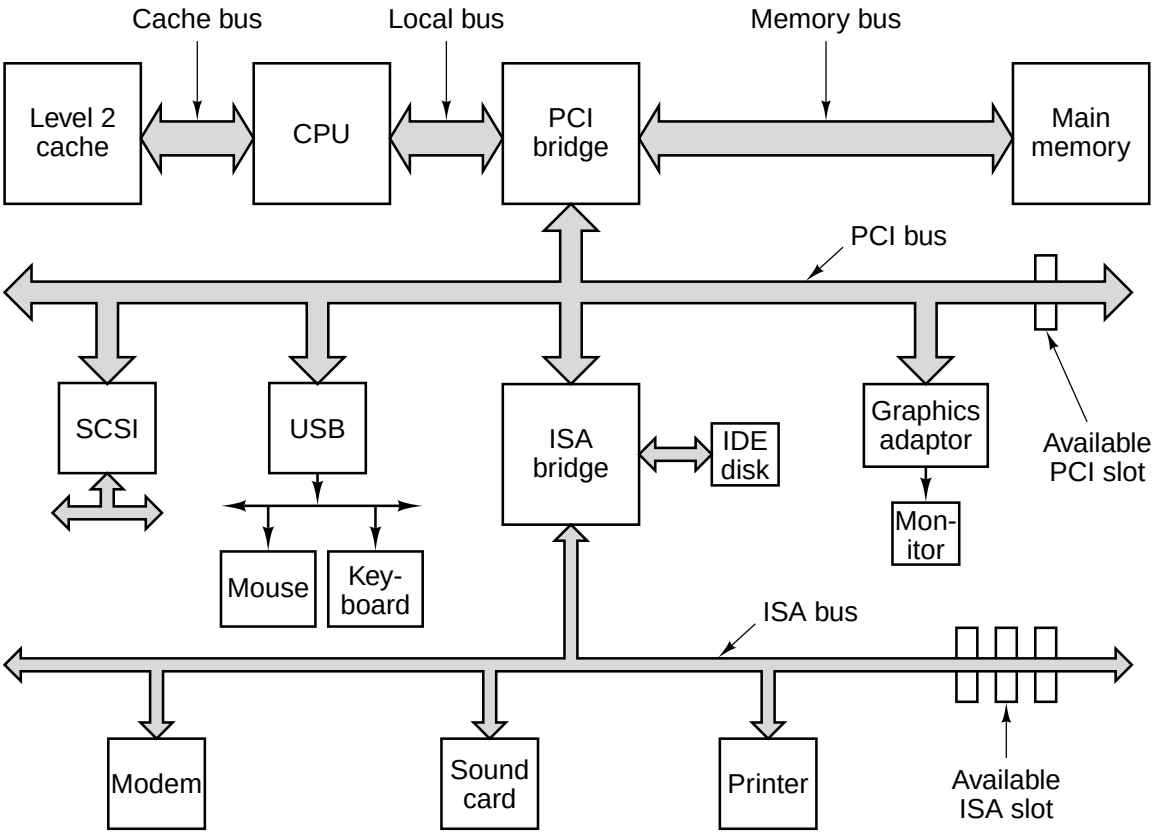

Fig. 1-11. The structure of a large Pentium system

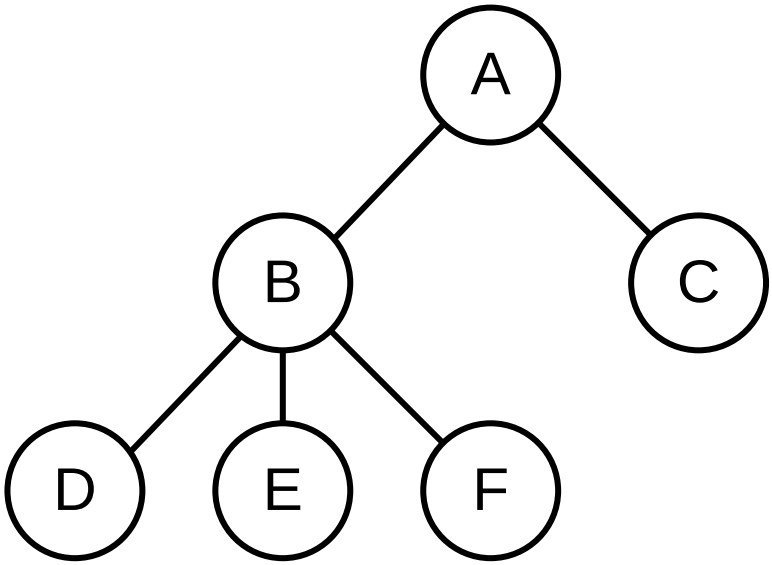

Fig. 1-12. A process tree. Process *A* created two child processes, *B* and *C*. Process *B* created three child processes, *D*, *E*, and *F*.

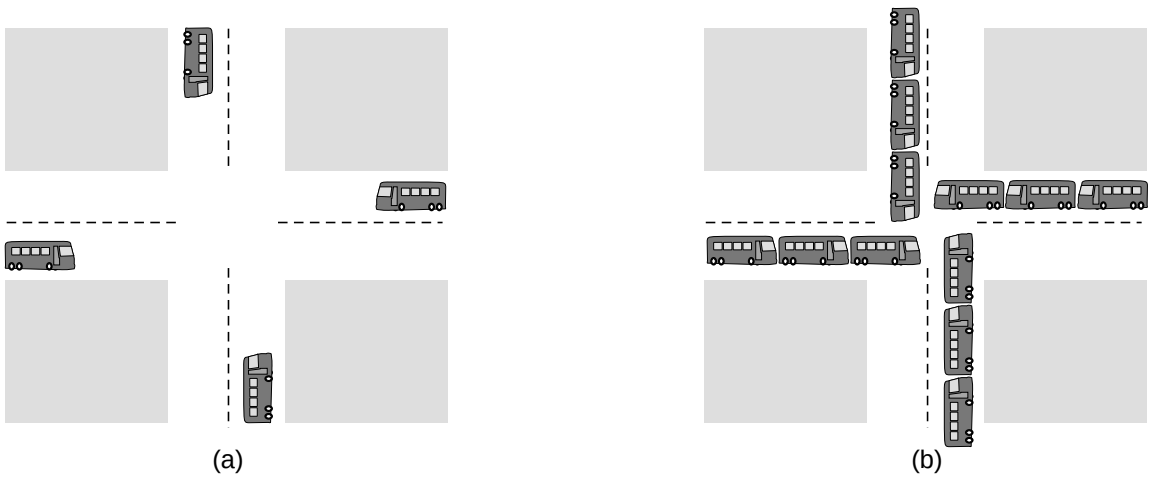

Fig. 1-13. (a) A potential deadlock. (b) An actual deadlock.

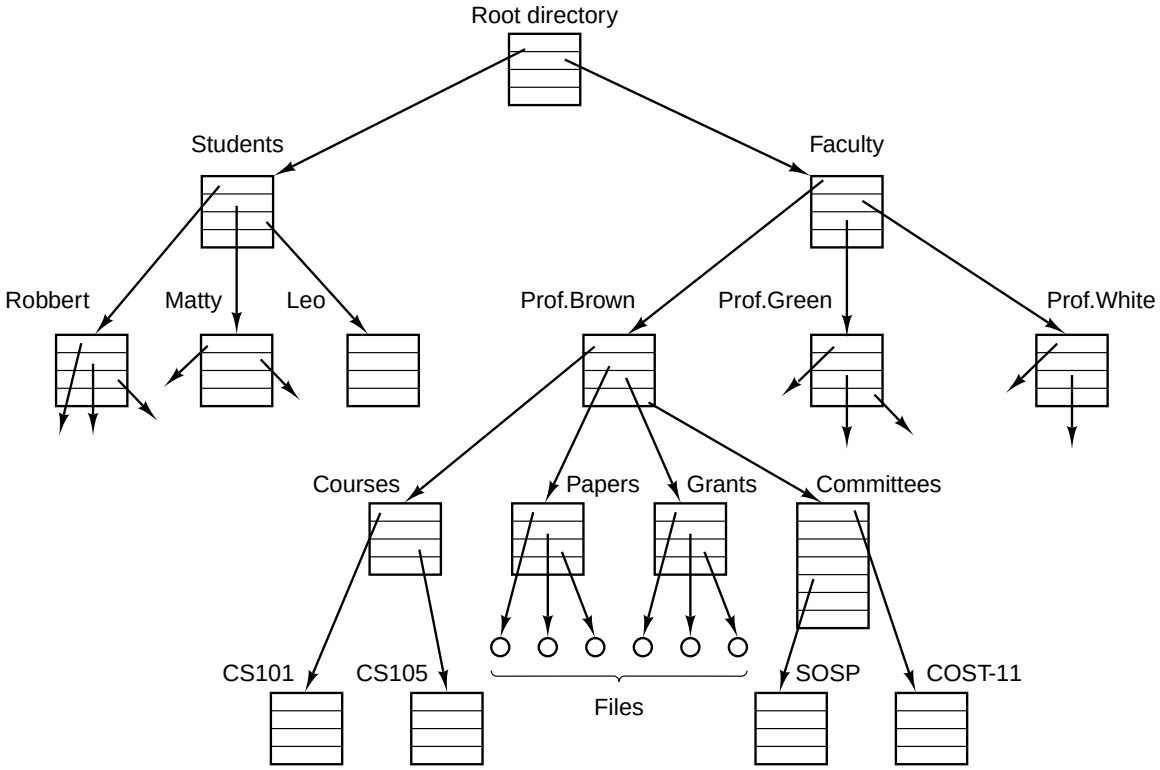

Fig. 1-14. A file system for a university department.

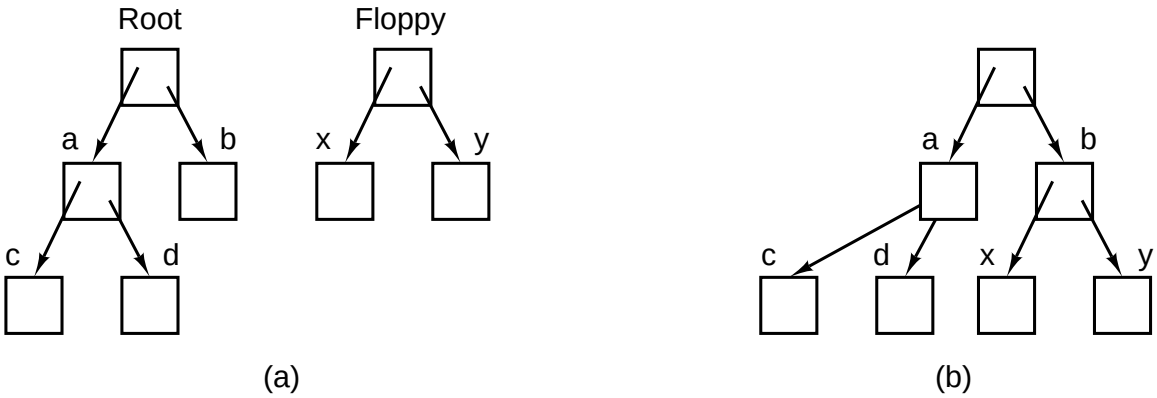

Fig. 1-15. (a) Before mounting, the files on drive 0 are not accessible. (b) After mounting, they are part of the file hierarchy.

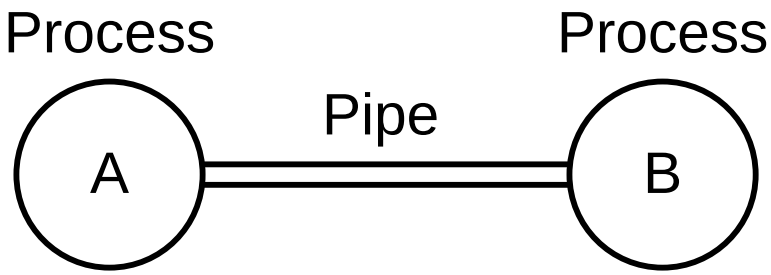

Fig. 1-16. Two processes connected by a pipe.

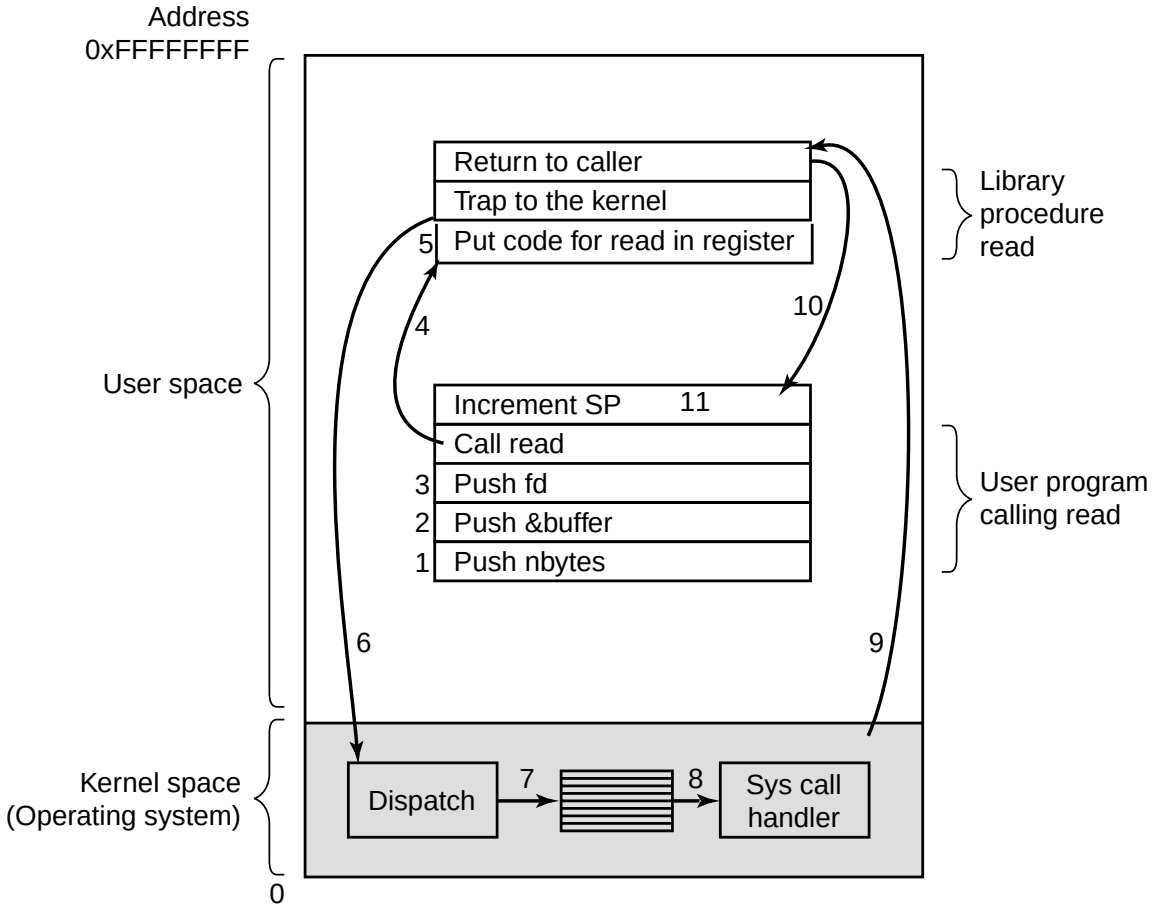

Fig. 1-17. The 11 steps in making the system call read(fd, buffer, nbytes).

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

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

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

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

Fig. 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. The parameters are explained in the text.

```
#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 */
     }
}
```

Fig. 1-19. A stripped-down shell. Throughout this book, *TRUE* is assumed to be defined as 1.

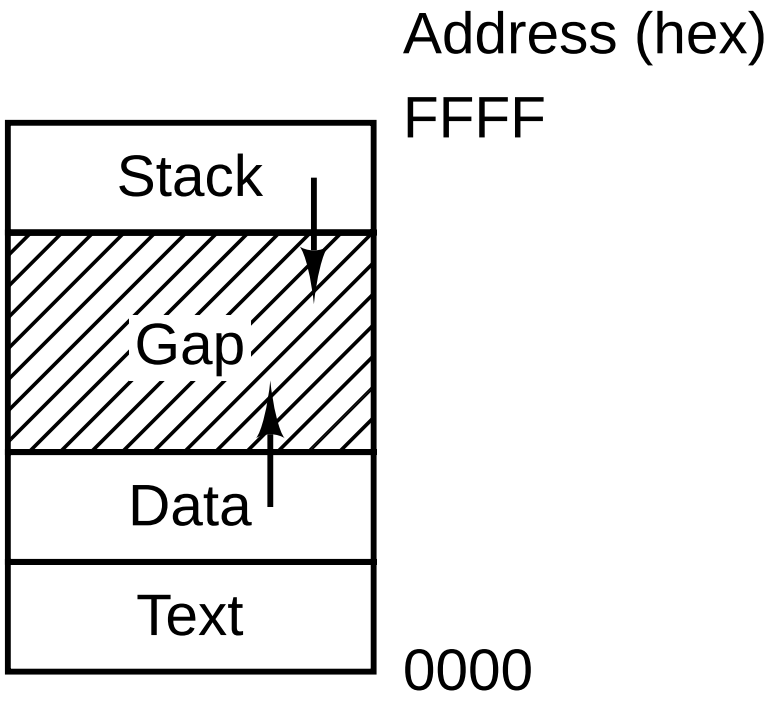

Fig. 1-20. Processes have three segments: text, data, and stack.

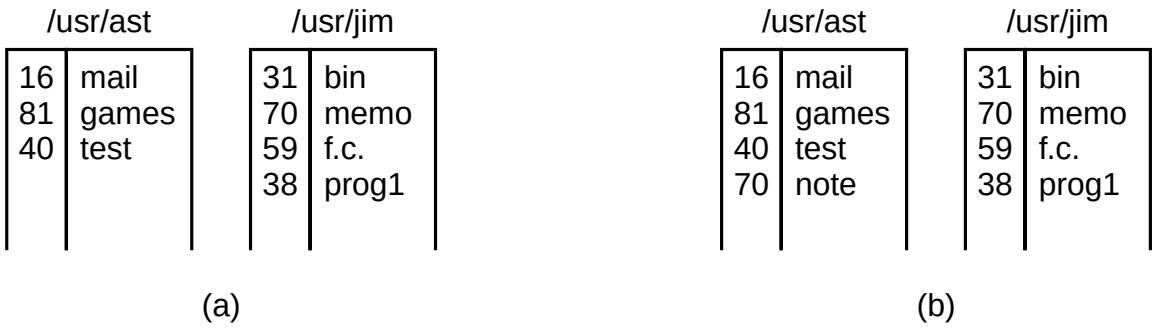

Fig. 1-21. (a) Two directories before linking */usr/jim/memo* to ast's directory. (b) The same directories after linking.

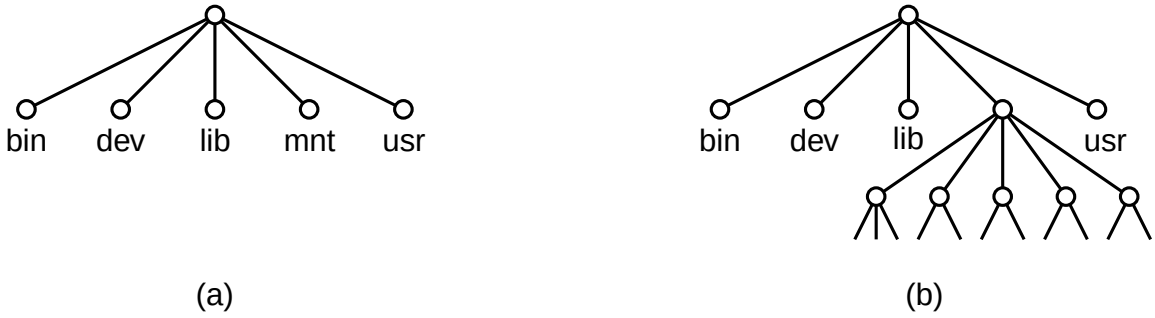

Fig. 1-22. (a) File system before the mount. (b) File system after the mount.

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

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

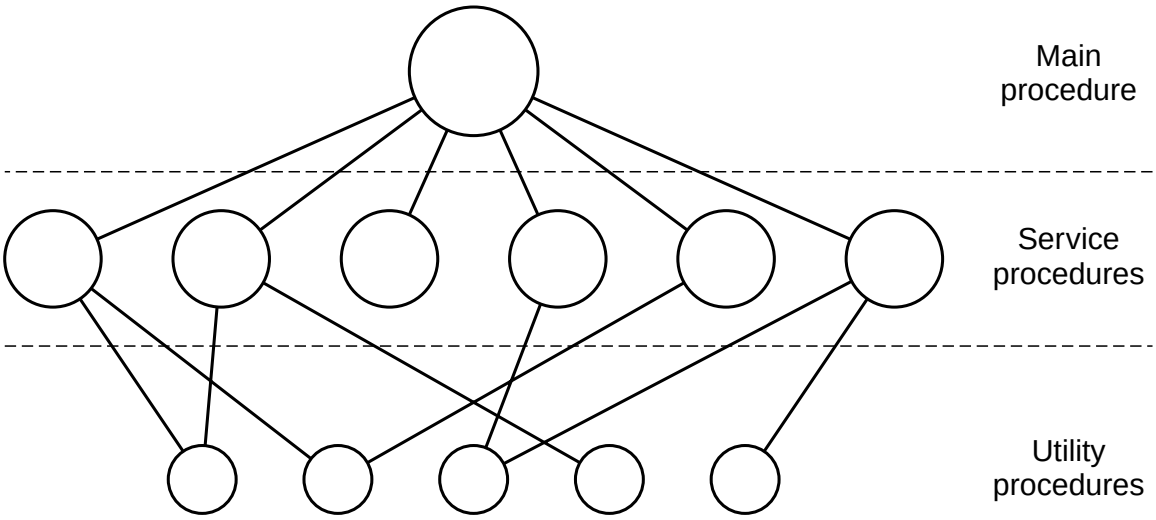

Fig. 1-24. A simple structuring model for a monolithic system.

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

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

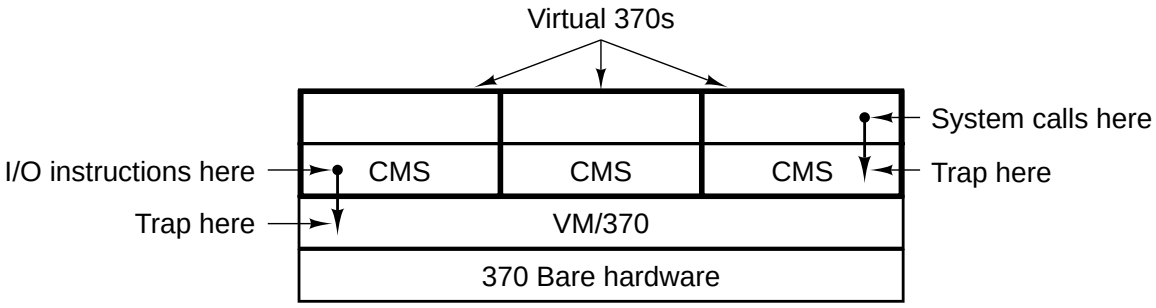

Fig. 1-26. The structure of VM/370 with CMS.

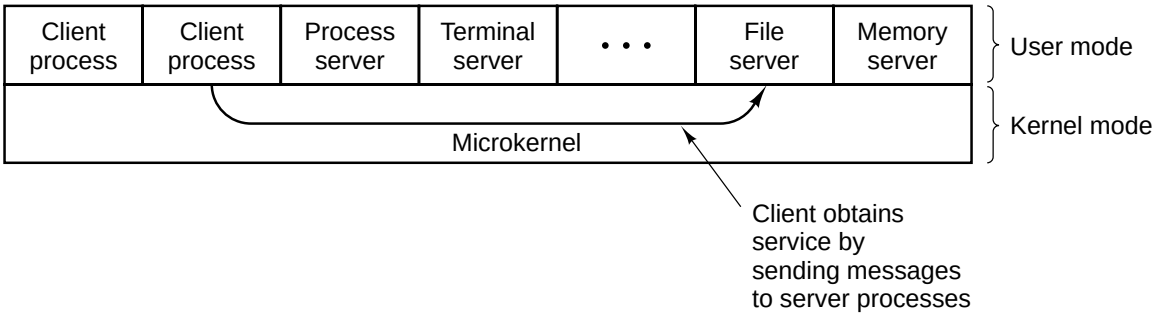

Fig. 1-27. The client-server model.

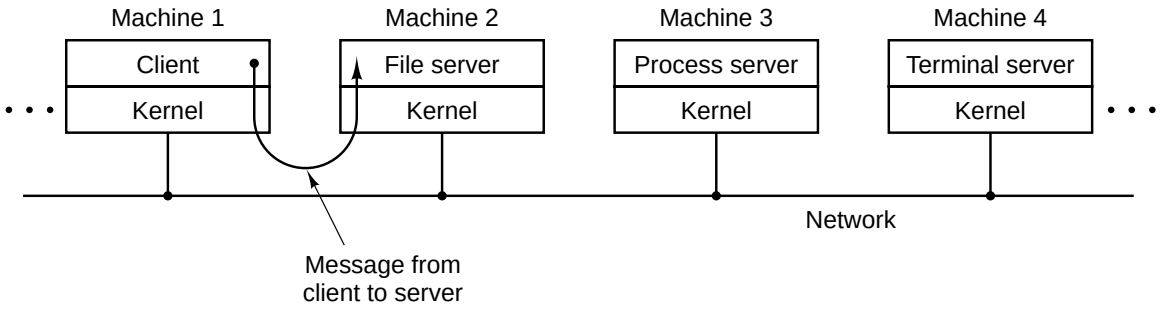

Fig. 1-28. The client-server model in a distributed system.

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

Fig. 1-29. The principal metric prefixes.