MODERN OPERATING SYSTEMS

Third Edition
ANDREW S. TANENBAUM

Chapter 10
Case Study 1: LINUX

History of UNIX and Linux

- UNICS
- PDP-11 UNIX
- Portable UNIX
- Berkeley UNIX
- Standard UNIX
- MINIX
- Linux

UNIX/Linux Goals

- Designed by programmers, for programmers
- Designed to be:
  - Simple
  - Elegant
  - Consistent
  - Powerful
  - Flexible

Interfaces to Linux

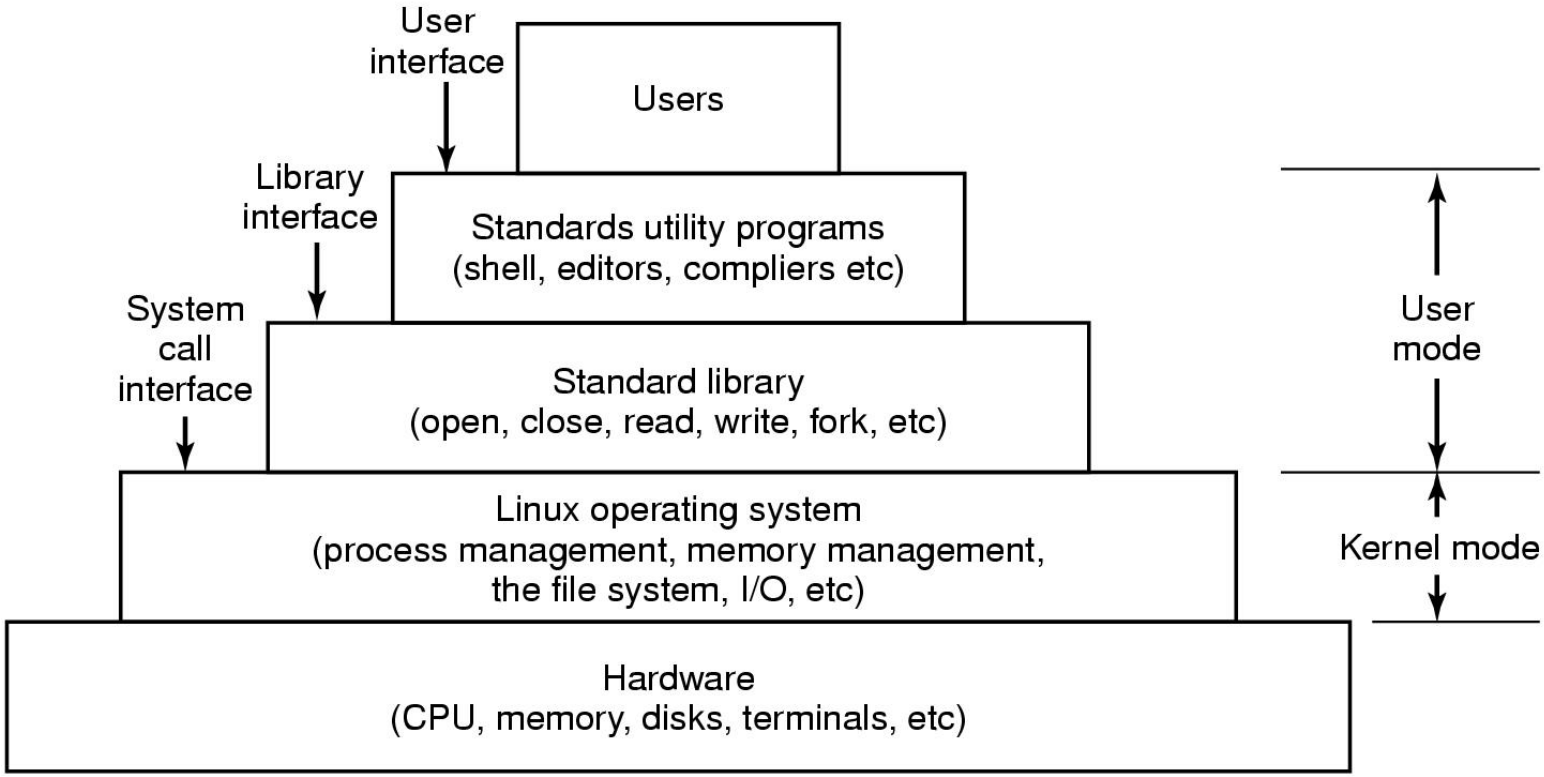

Figure 10-1. The layers in a Linux system.

Linux Utility Programs (1)

Categories of utility programs:

- File and directory manipulation commands.
- Filters.
- Program development tools, such as editors and compilers.
- Text processing.
- System administration.
- Miscellaneous.

Linux Utility Programs (2)

Program	Typical use
cat	Concatenate multiple files to standard output
chmod	Change file protection mode
cp	Copy one or more files
cut	Cut columns of text from a file
grep	Search a file for some pattern
head	Extract the first lines of a file
ls	List directory
make	Compile files to build a binary
mkdir	Make a directory
od	Octal dump a file
paste	Paste columns of text into a file
pr	Format a file for printing
ps	List running processes
rm	Remove one or more files
rmdir	Remove a directory
sort	Sort a file of lines alphabetically
tail	Extract the last lines of a file
tr	Translate between character sets

Figure 10-2. A few of the common Linux utility programs required by POSIX.

Kernel Structure

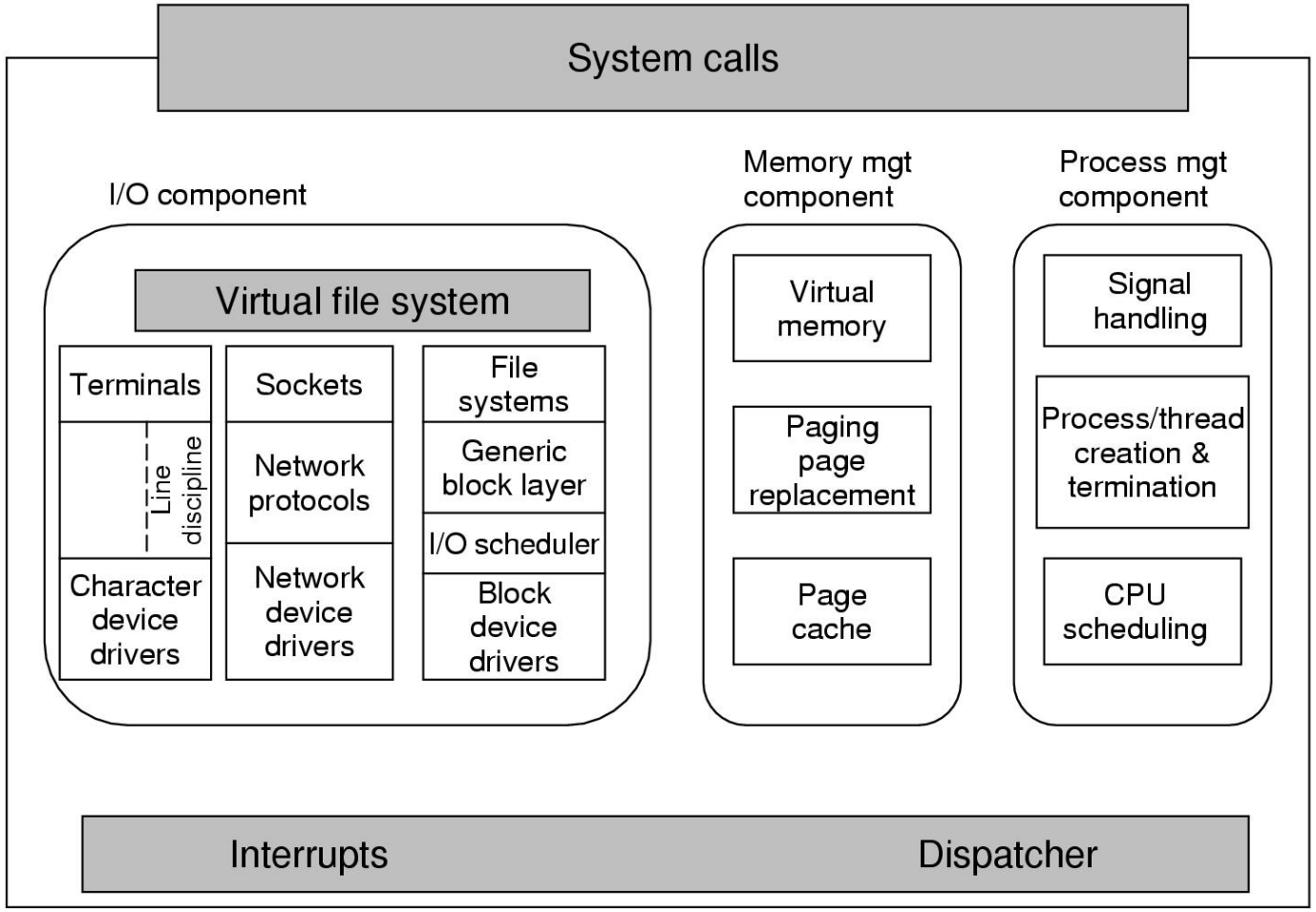

Figure 10-3. Structure of the Linux kernel

Processes in Linux

```
pid = fork( );                  /* if the fork succeeds, pid > 0 in the parent */
if (pid < 0) {
        handle_error( );        /* fork failed (e.g., memory or some table is full) */
} else if (pid > 0) {
                                /* parent code goes here. /*/
} else {
                                /* child code goes here. /*/
}
```

Figure 10-4. Process creation in Linux.

Signals in Linux (1)

Signal	Cause
SIGABRT	Sent to abort a process and force a core dump
SIGALRM	The alarm clock has gone off
SIGFPE	A floating-point error has occurred (e.g., division by 0)
SIGHUP	The phone line the process was using has been hung up
SIGILL	The user has hit the DEL key to interrupt the process
SIGQUIT	The user has hit the key requesting a core dump
SIGKILL	Sent to kill a process (cannot be caught or ignored)
SIGPIPE	The process has written to a pipe which has no readers
SIGSEGV	The process has referenced an invalid memory address
SIGTERM	Used to request that a process terminate gracefully
SIGUSR1	Available for application-defined purposes
SIGUSR2	Available for application-defined purposes

Figure 10-5. The signals required by POSIX.

Process Management System Calls in Linux

System call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, opts)	Wait for a child to terminate
s = execve(name, argv, envp)	Replace a process' core image
exit(status)	Terminate process execution and return status
s = sigaction(sig, &act, &oldact)	Define action to take on signals
s = sigreturn(&context)	Return from a signal
s = sigprocmask(how, &set, &old)	Examine or change the signal mask
s = sigpending(set)	Get the set of blocked signals
s = sigsuspend(sigmask)	Replace the signal mask and suspend the process
s = kill(pid, sig)	Send a signal to a process
residual = alarm(seconds)	Set the alarm clock
s = pause( )	Suspend the caller until the next signal

Figure 10-6. Some system calls relating to processes.

A Simple Linux Shell

```
while (TRUE) {                                  /* repeat forever /*/
        type_prompt( );                         /* display prompt on the screen */
        read_command(command, params);          /* read input line from keyboard */

        pid = fork( );                          /* fork off a child process */
        if (pid < 0) {
                printf("Unable to fork0);       /* error condition */
                continue;                       /* repeat the loop */
        }

        if (pid != 0) {
                waitpid (–1, &status, 0);       /* parent waits for child */
        } else {
                execve(command, params, 0);     /* child does the work */
        }
}
```

Figure 10-7. A highly simplified shell.

Implementation of Processes and Threads

Categories of information in the process descriptor:

- Scheduling parameters
- Memory image
- Signals
- Machine registers
  System call state
- File descriptor table
- Accounting
- Kernel stack
- Miscellaneous

Implementation of Exec

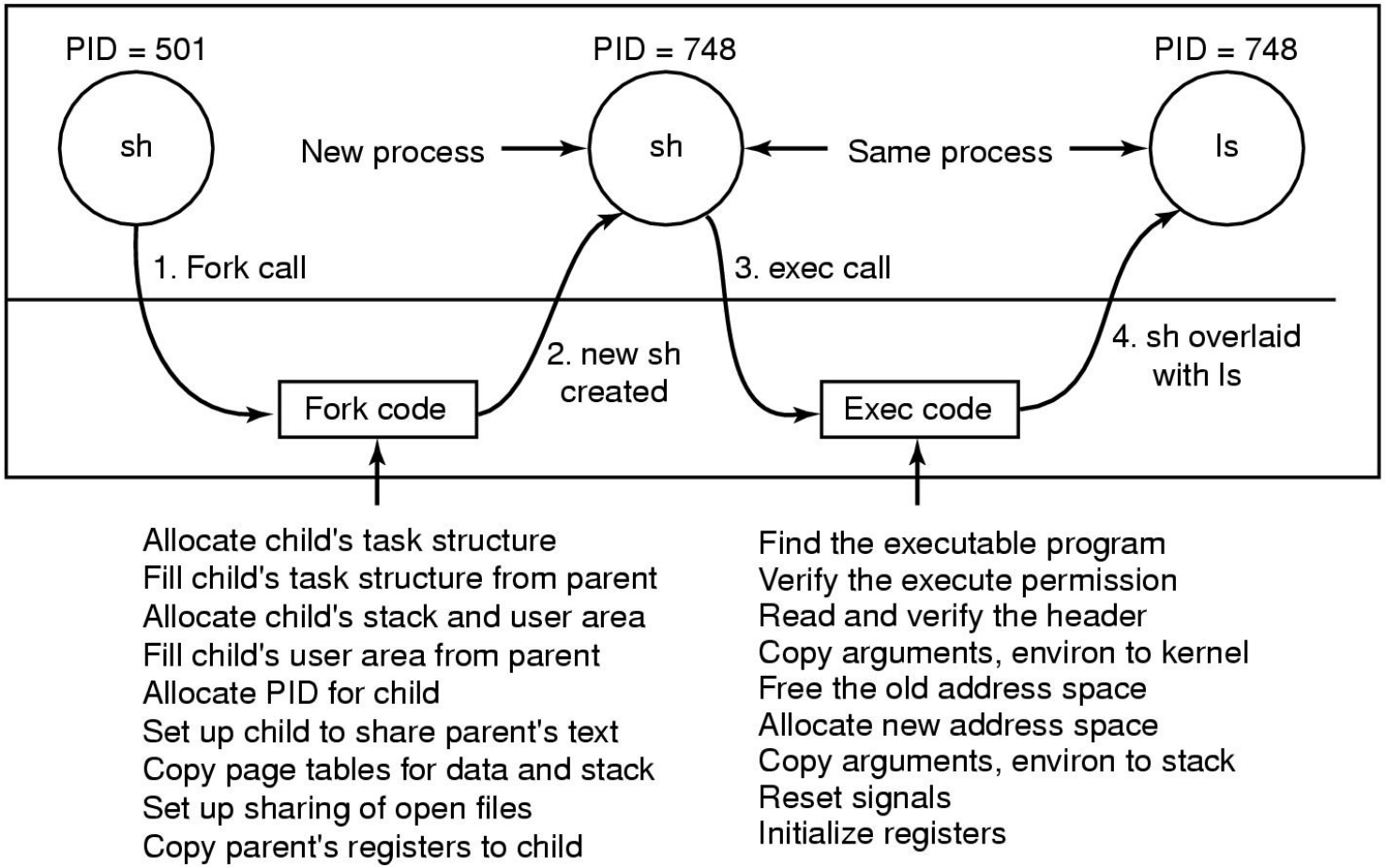

Figure 10-8. The steps in executing the command *ls* typed to the shell.

The Clone System Call

Flag	Meaning when set	Meaning when cleared
CLONE_VM	Create a new thread	Create a new process
CLONE_FS	Share umask, root, and working dirs	Do not share them
CLONE_FILES	Share the file descriptors	Copy the file descriptors
CLONE_SIGHAND	Share the signal handler table	Copy the table
CLONE_PID	New thread gets old PID	New thread gets own PID
CLONE_PARENT	New thread has same parent as caller	New thread's parent is caller

Figure 10-9. Bits in the sharing_flags bitmap.

Scheduling in Linux (1)

Three classes of threads for scheduling purposes:

- Real-time FIFO.
- Real-time round robin.
- Timesharing.

Scheduling in Linux (2)

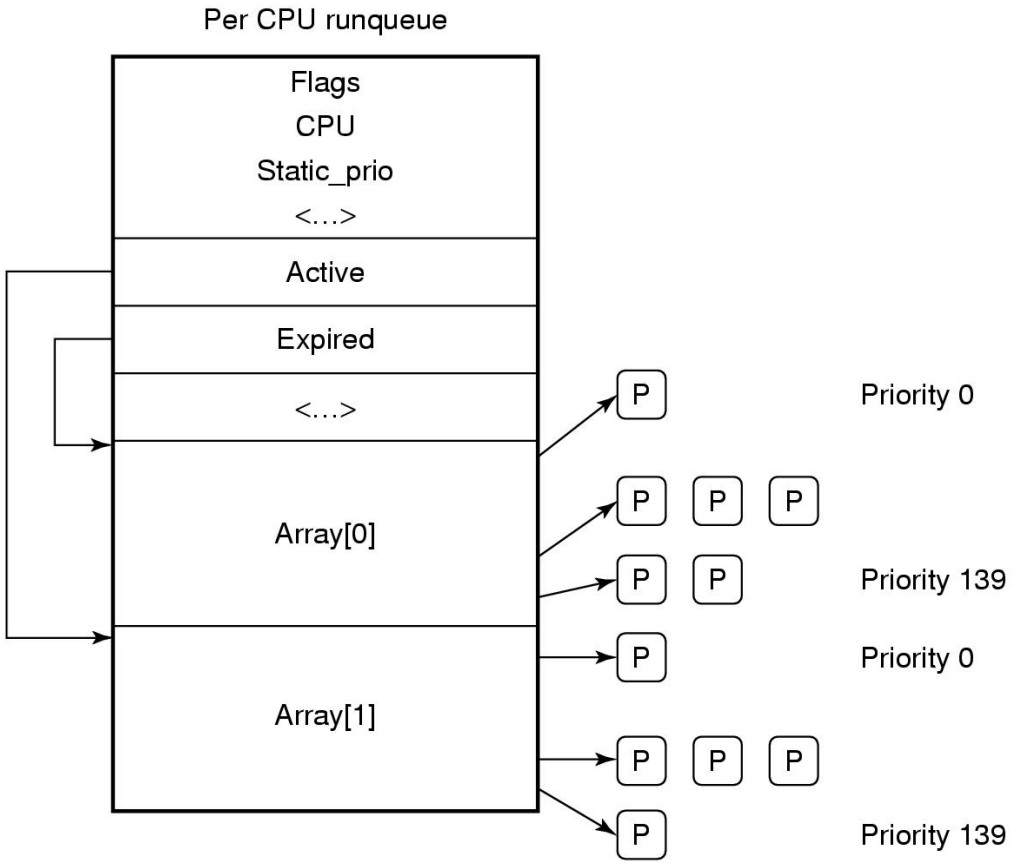

Figure 10-10. Illustration of Linux runqueue and priority arrays.

Booting Linux

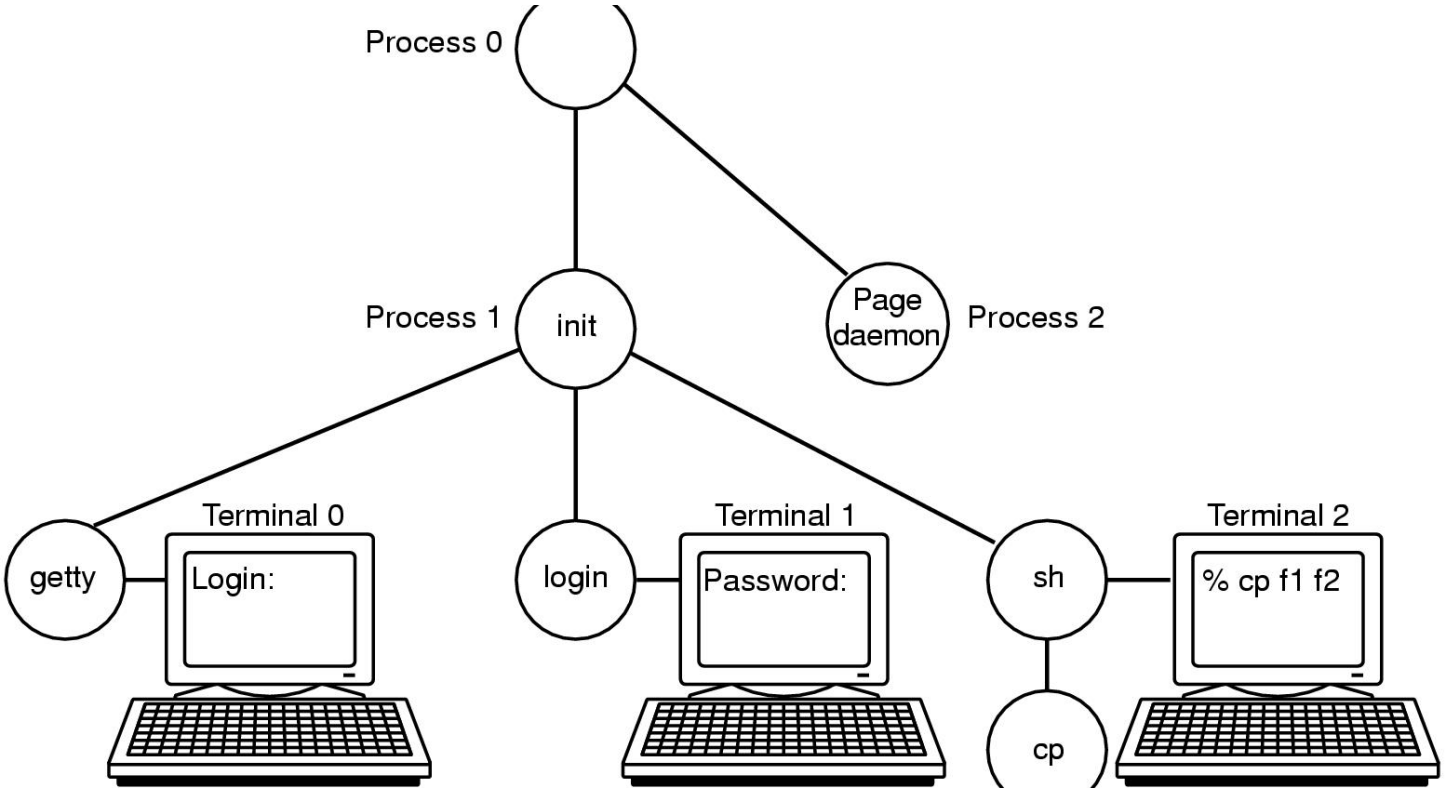

Figure 10-11. The sequence of processes used to boot some Linux systems.

Memory Management in Linux (1)

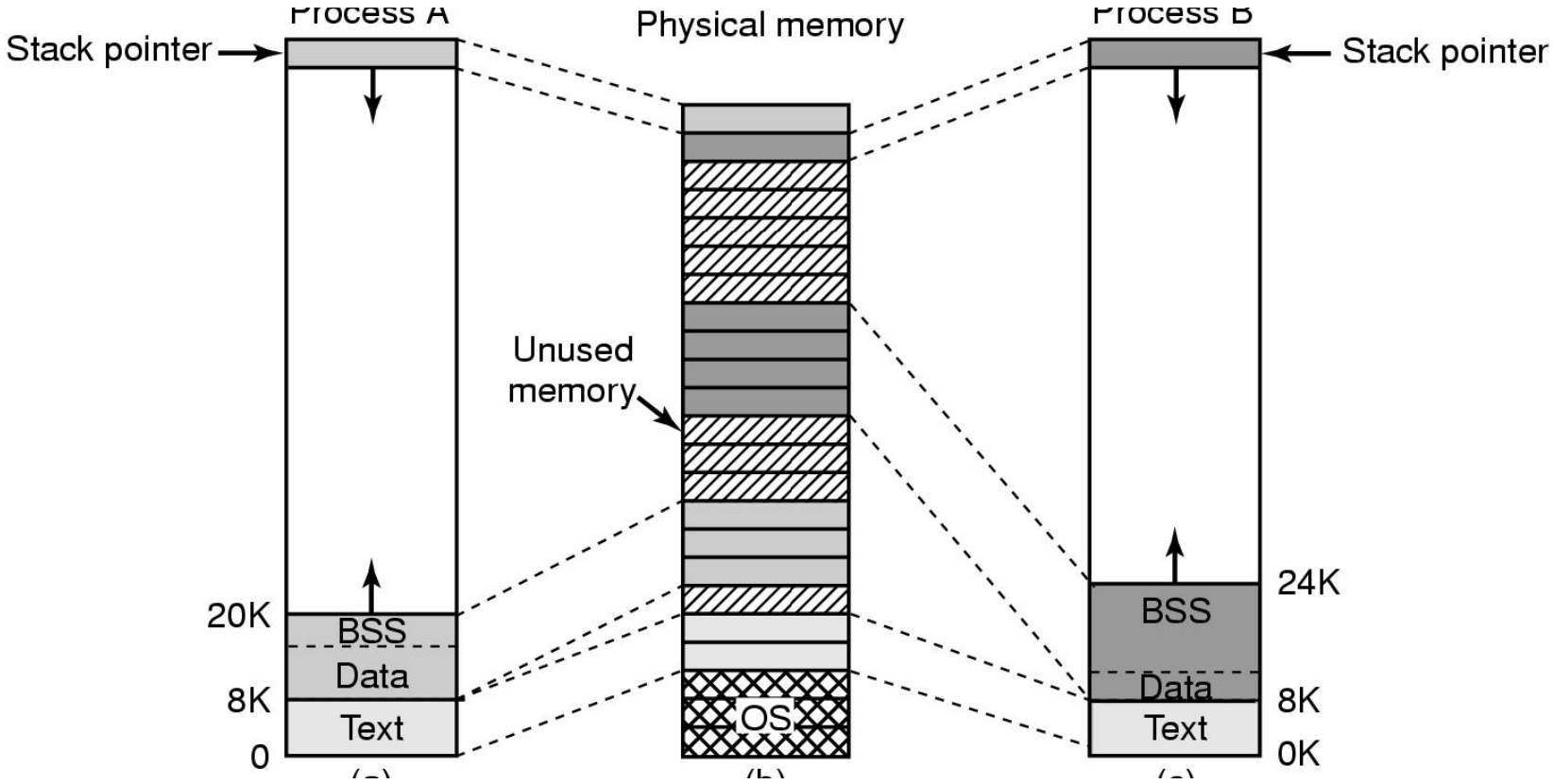

Figure 10-12. (a) Process A's virtual address space. (b) Physical memory. (c) Process B's virtual address space.

Memory Management in Linux (2)

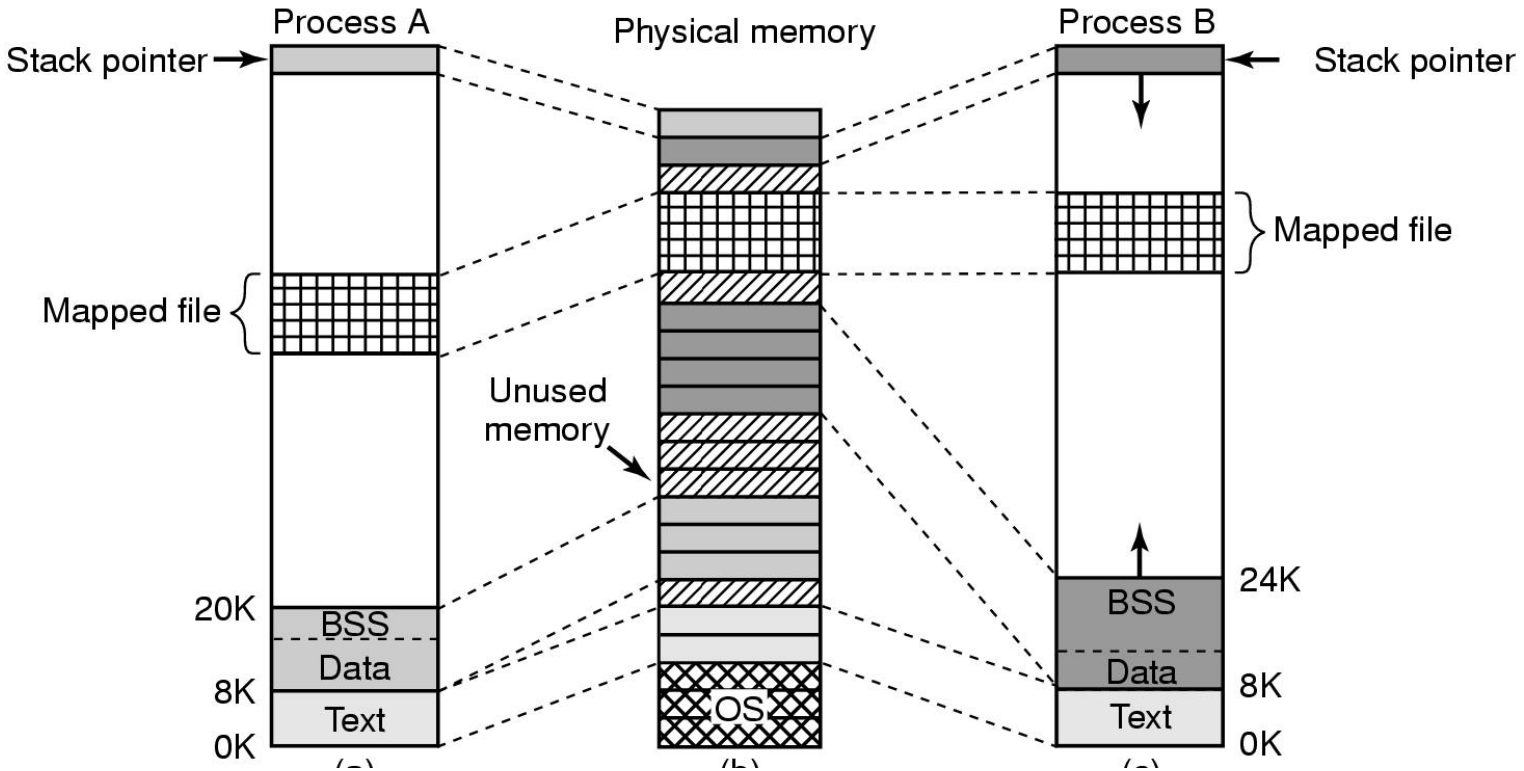

Figure 10-13. Two processes can share a mapped file.

Memory Management System Calls in Linux

System call	Description
s = brk(addr)	Change data segment size
a = mmap(addr, len, prot, flags, fd, offset)	Map a file in
s = unmap(addr, len)	Unmap a file

Figure 10-14. Some system calls relating to memory management.

Physical Memory Management (1)

Linux distinguishes between three memory zones:

- ZONE_DMA - pages that can be used for DMA operations.
- ZONE_NORMAL - normal, regularly mapped pages.
- ZONE_HIGHMEM - pages with high-memory addresses, which are not permanently mapped.

Physical Memory Management (2)

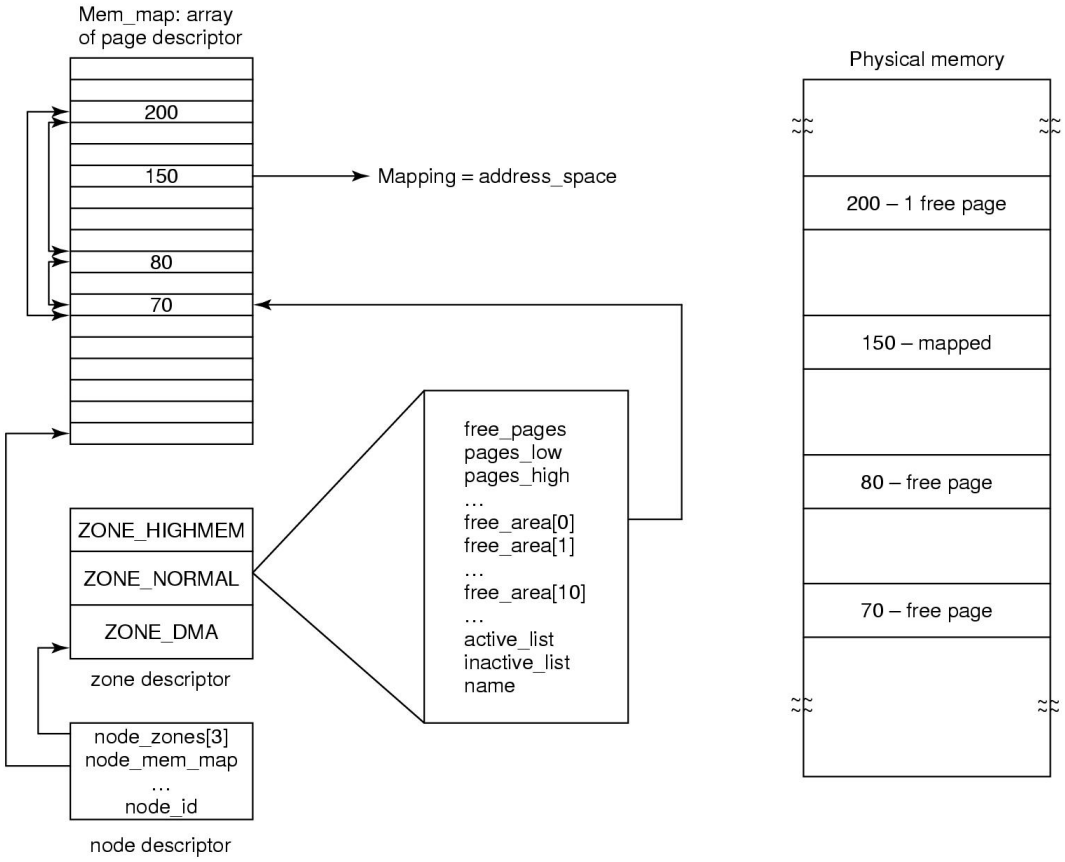

Figure 10-15. Linux main memory representation.

Physical Memory Management (3)

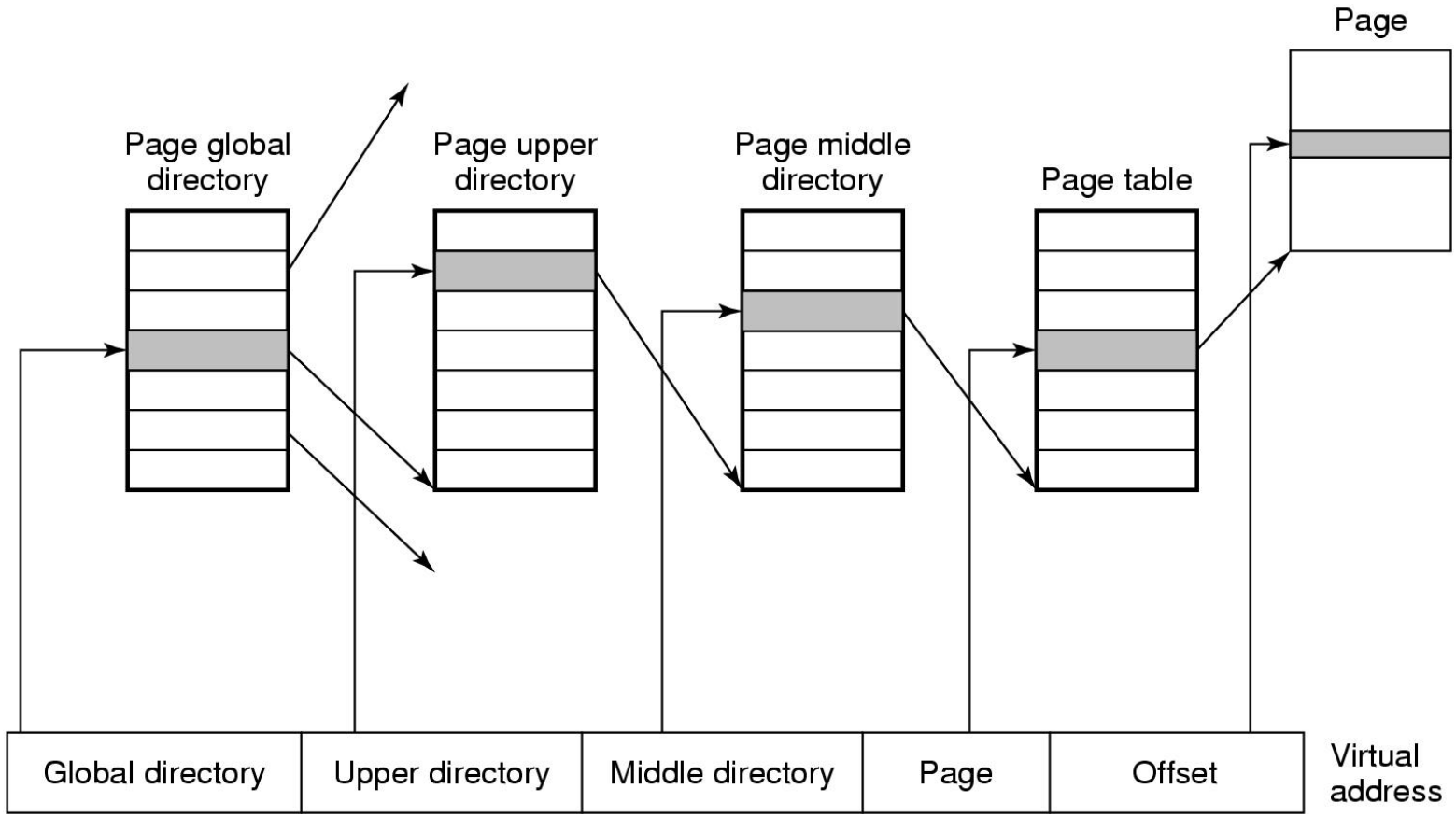

Figure 10-16. Linux uses four-level page tables.

Memory Allocation Mechanisms

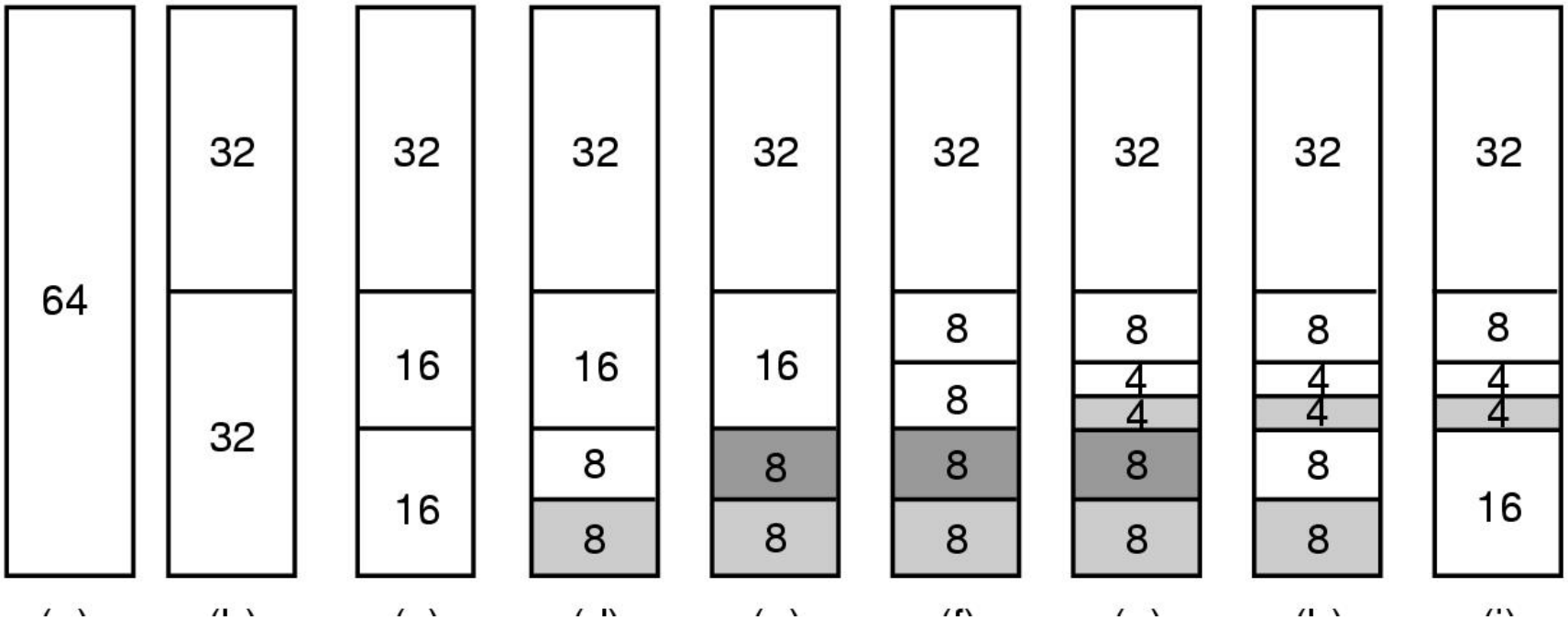

Figure 10-17. Operation of the buddy algorithm.

The Page Replacement Algorithm

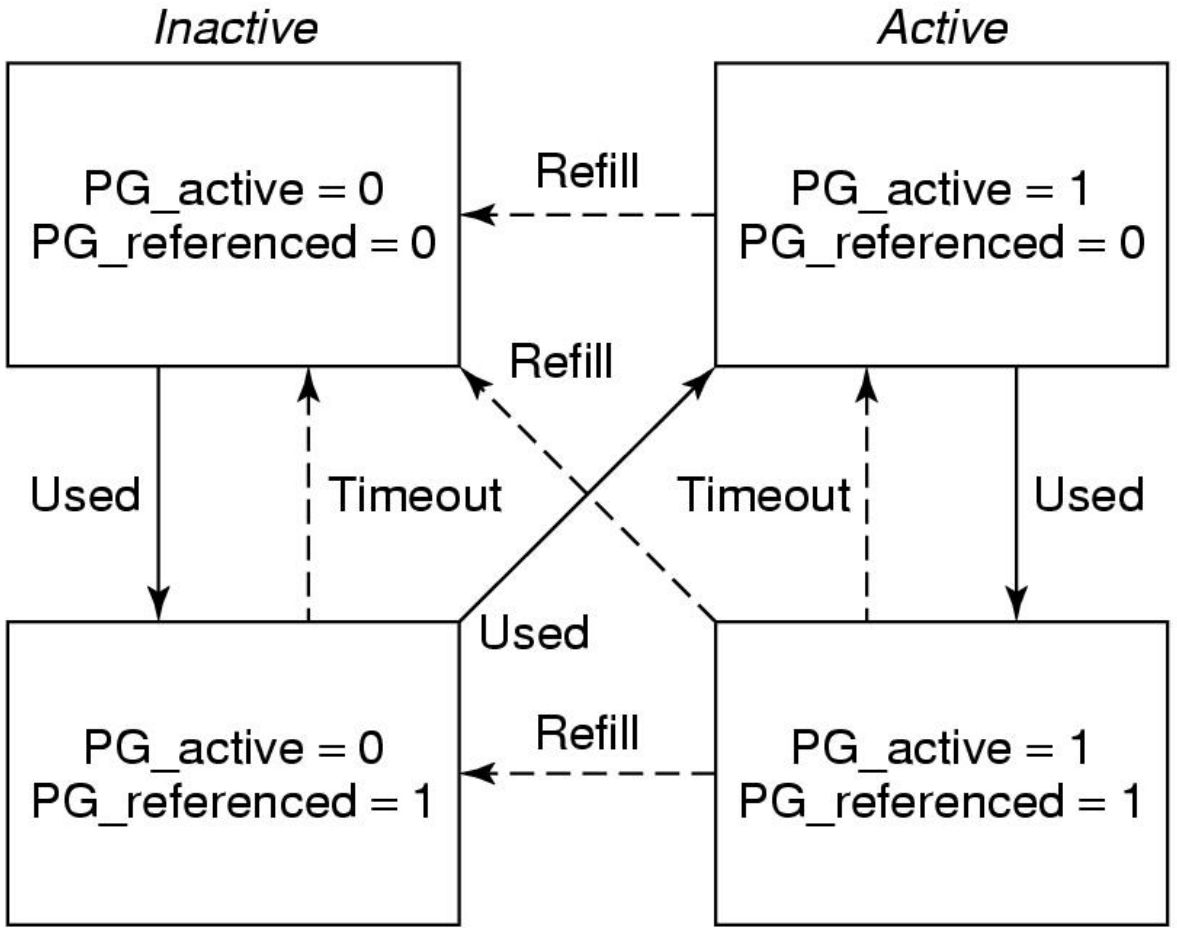

Figure 10-18. Page states considered in the page frame replacement algorithm.

Networking (1)

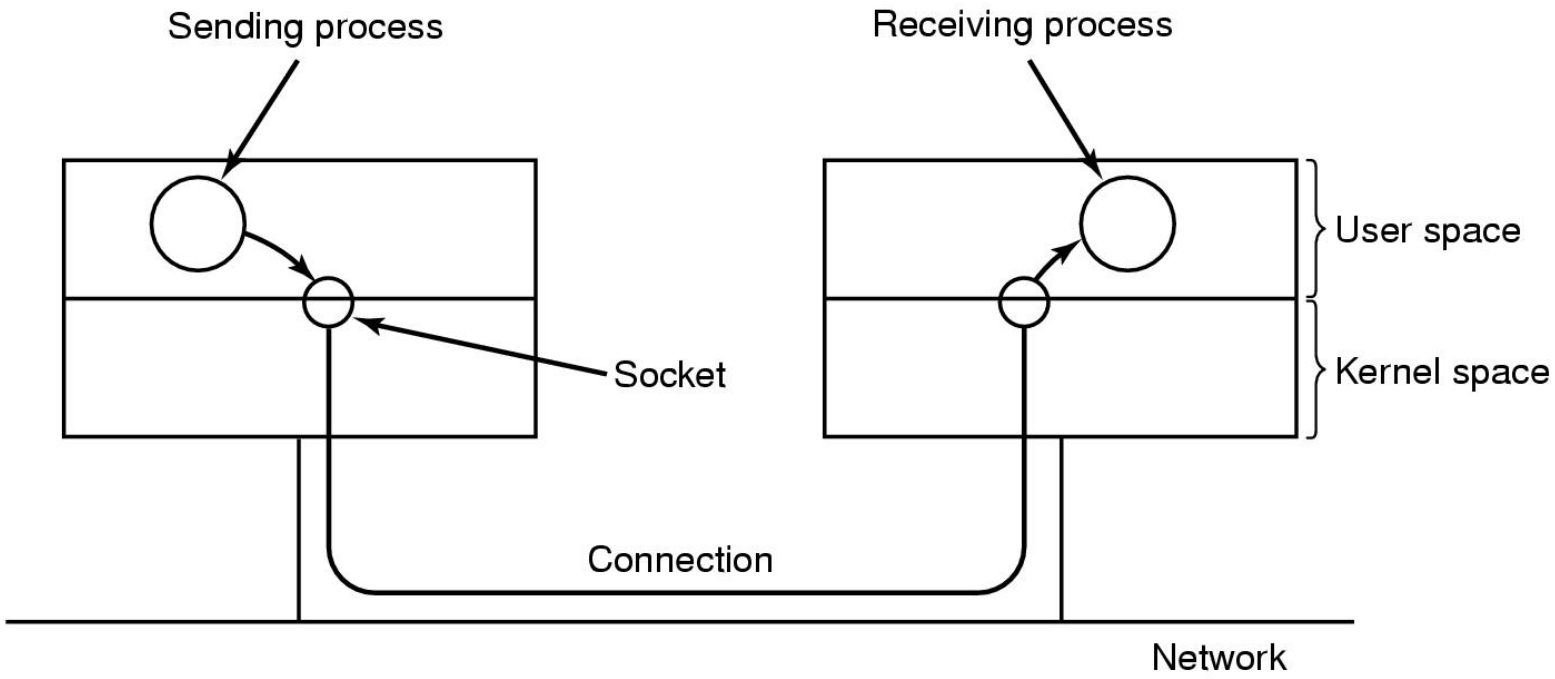

Figure 10-19. The uses of sockets for networking.

Networking (2)

Types of networking:

- Reliable connection-oriented byte stream.
- Reliable connection-oriented packet stream.
- Unreliable packet transmission.

Input/Output System Calls in Linux

Function call	Description
s = cfsetospeed(&termios, speed)	Set the output speed
s = cfsetispeed(&termios, speed)	Set the input speed
s = cfgetospeed(&termios, speed)	Get the output speed
s = cfgtetispeed(&termios, speed)	Get the input speed
s = tcsetattr(fd, opt, &termios)	Set the attributes
s = tcgetattr(fd, &termios)	Get the attributes

Figure 10-20. The main POSIX calls for managing the terminal.

The Major Device Table

Device	Open	Close	Read	Write	Ioctl	Other
Null	null	null	null	null	null	...
Memory	null	null	mem_read	mem_write	null	...
Keyboard	k_open	k_close	k_read	error	k_ioctl	...
Tty	tty_open	tty_close	tty_read	tty_write	tty_ioctl	...
Printer	lp_open	lp_close	error	lp_write	lp_ioctl	...

Figure 10-21. Some of the file operations supported for typical character devices.

Implementation of Input/Output in Linux (2)

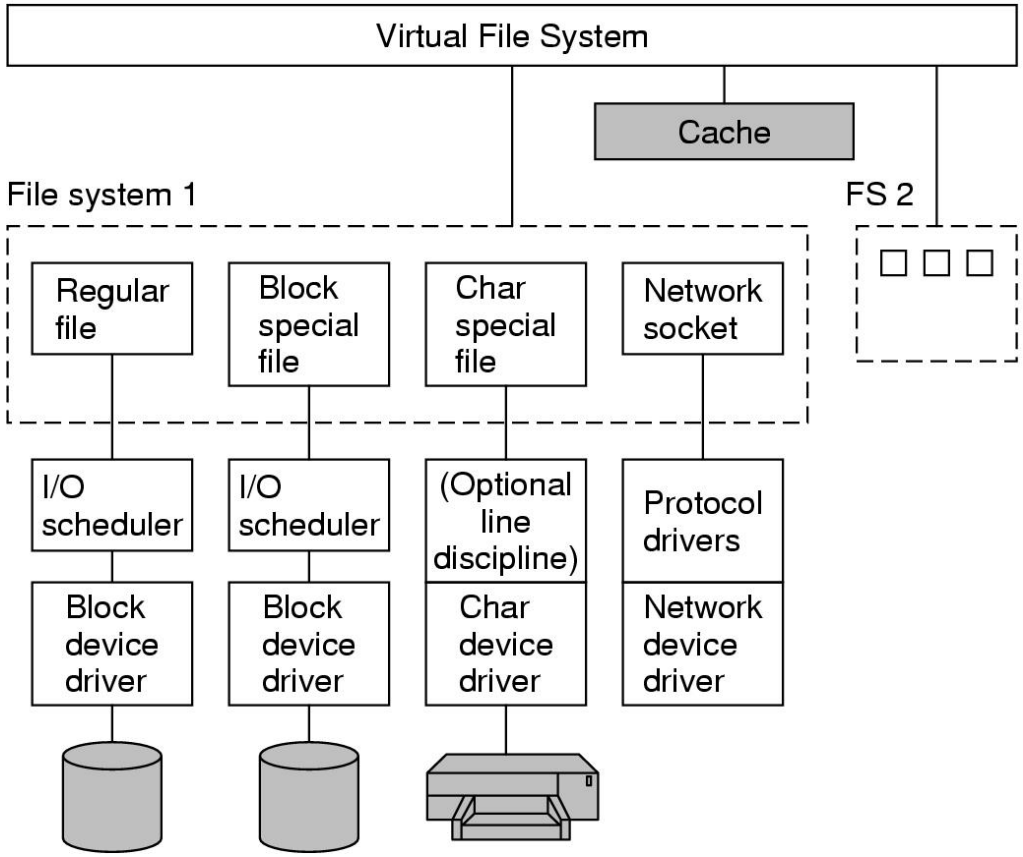

Figure 10-22. The Linux I/O system showing one file system in detail.

The Linux File System (1)

Directory	Contents
bin	Binary (executable) programs
dev	Special files for I/O devices
etc	Miscellaneous system files
lib	Libraries
usr	User directories

Figure 10-23. Some important directories found in most Linux systems.

The Linux File System (2)

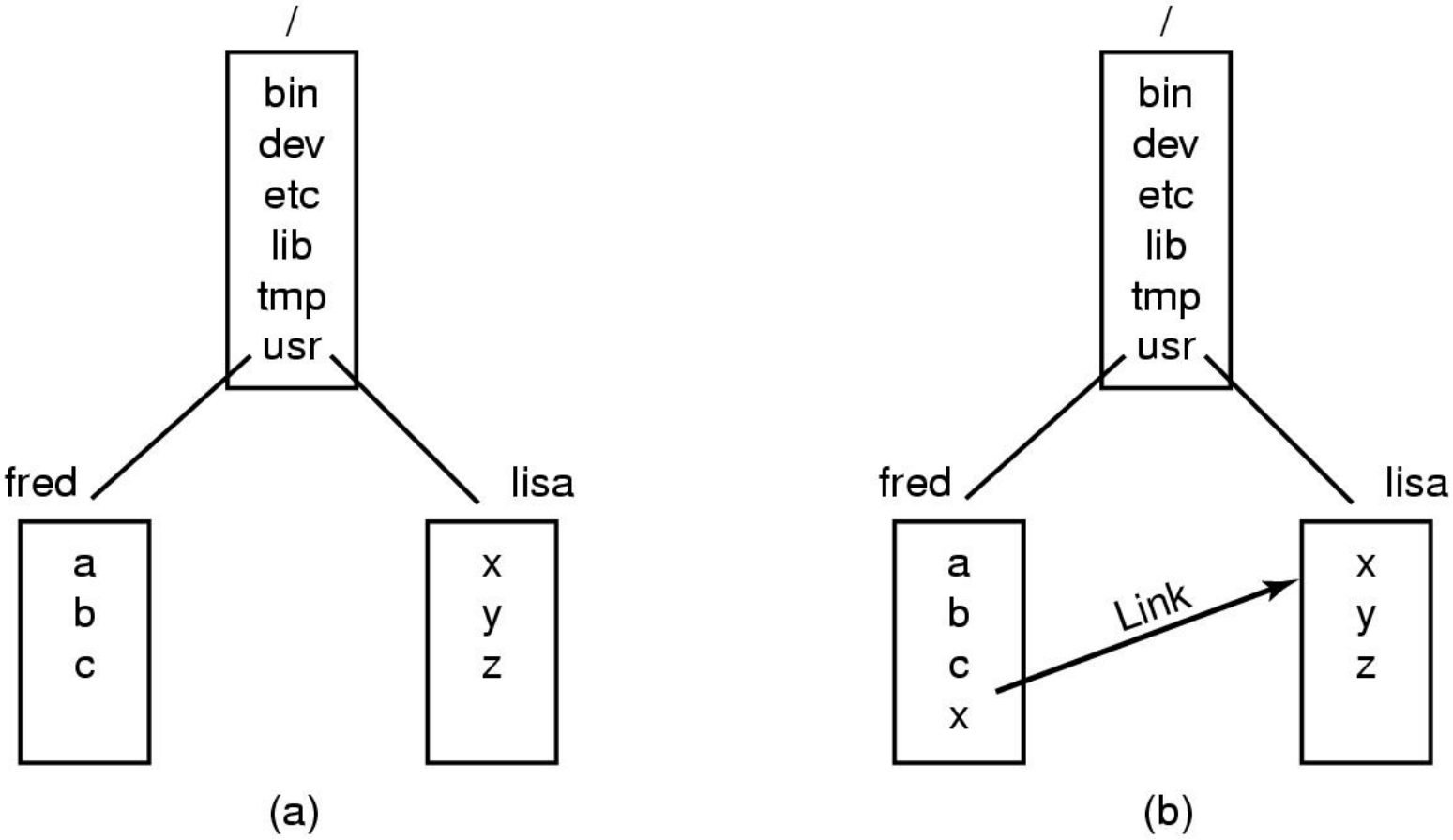

Figure 10-24. (a) Before linking. (b) After linking.

The Linux File System (3)

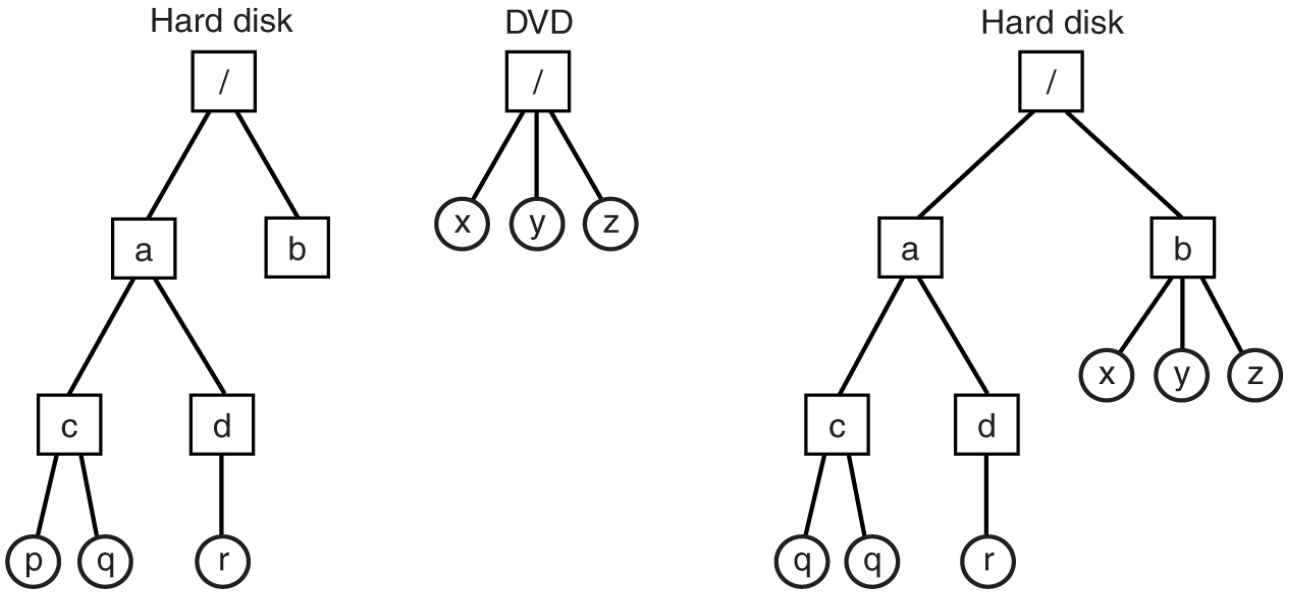

Figure 10-25. (a) Separate file systems. (b) After mounting.

The Linux File System (4)

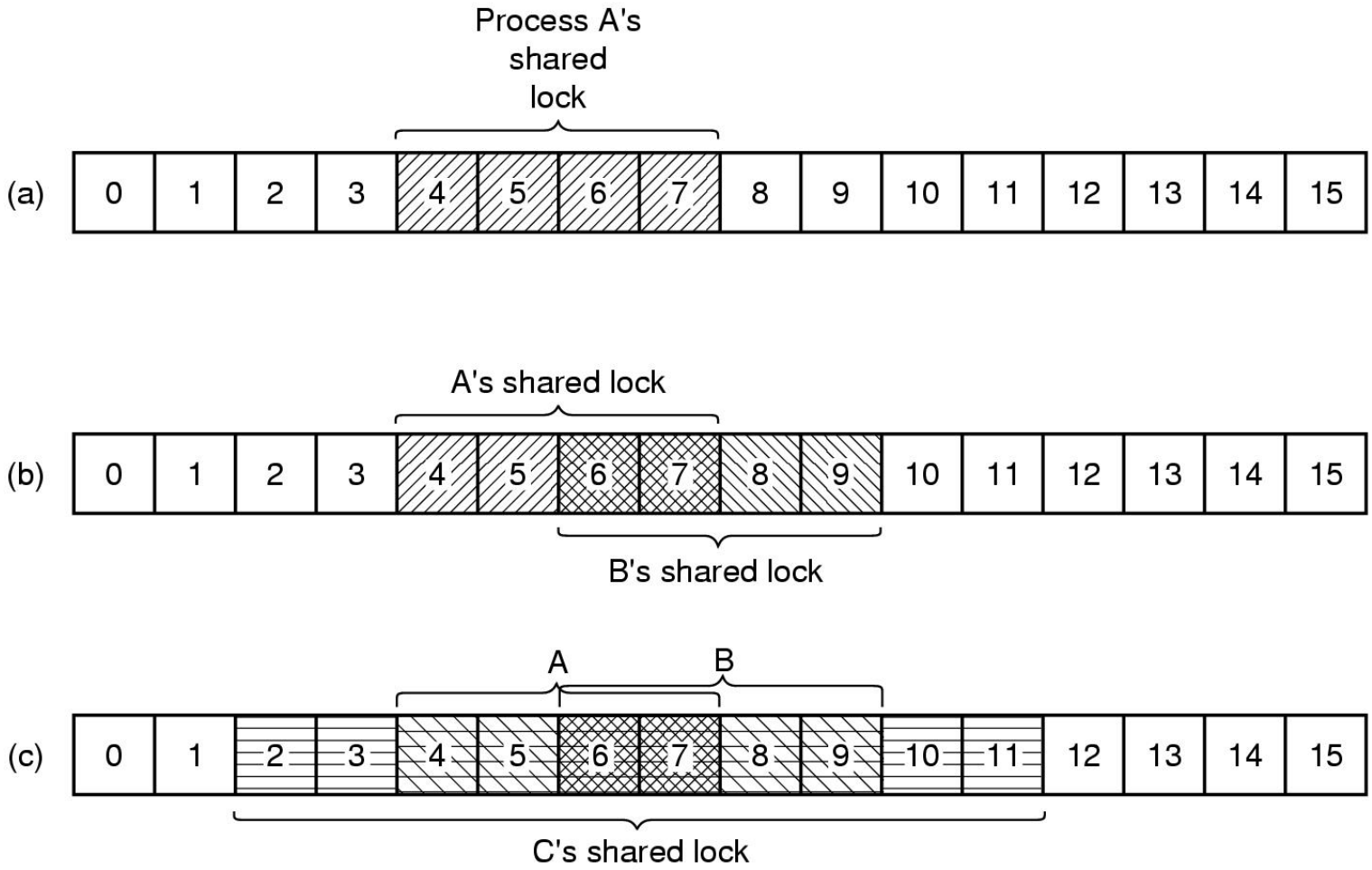

Figure 10-26. (a) A file with one lock. (b) Addition of a second lock. (c) A third lock.

File System Calls in Linux (1)

System call	Description
fd = creat(name, mode)	One way to create a new file
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
s = fstat(fd, &buf)	Get a file's status information
s = pipe(&fd[0])	Create a pipe
s = fcntl(fd, cmd, ...)	File locking and other operations

Figure 10-27. System calls relating to files.

File System Calls in Linux (2)

Device the file is on
I-node number (which file on the device)
File mode (includes protection information)
Number of links to the file
Identity of the file's owner
Group the file belongs to
File size (in bytes)
Creation time
Time of last access
Time of last modification

Figure 10-28. The fields returned by the **stat** system call.

File System Calls in Linux (3)

System call	Description
s = mkdir(path, mode)	Create a new directory
s = rmdir(path)	Remove a directory
s = link(oldpath, newpath)	Create a link to an existing file
s = unlink(path)	Unlink a file
s = chdir(path)	Change the working directory
dir = opendir(path)	Open a directory for reading
s = closedir(dir)	Close a directory
dirent = readdir(dir)	Read one directory entry
rewinddir(dir)	Rewind a directory so it can be reread

Figure 10-29. System calls relating to directories.

The Linux Virtual File System

Object	Description	Operations
Superblock	specific filesystem	read_inode, sync_fs
Dentry	directory entry, single component of a path	create, link
I-node	specific file	d_compare, d_delete
File	open file associated with a process	read, write

Figure 10-30. File system abstractions supported by the VFS.

The Linux Ext2 File System (1)

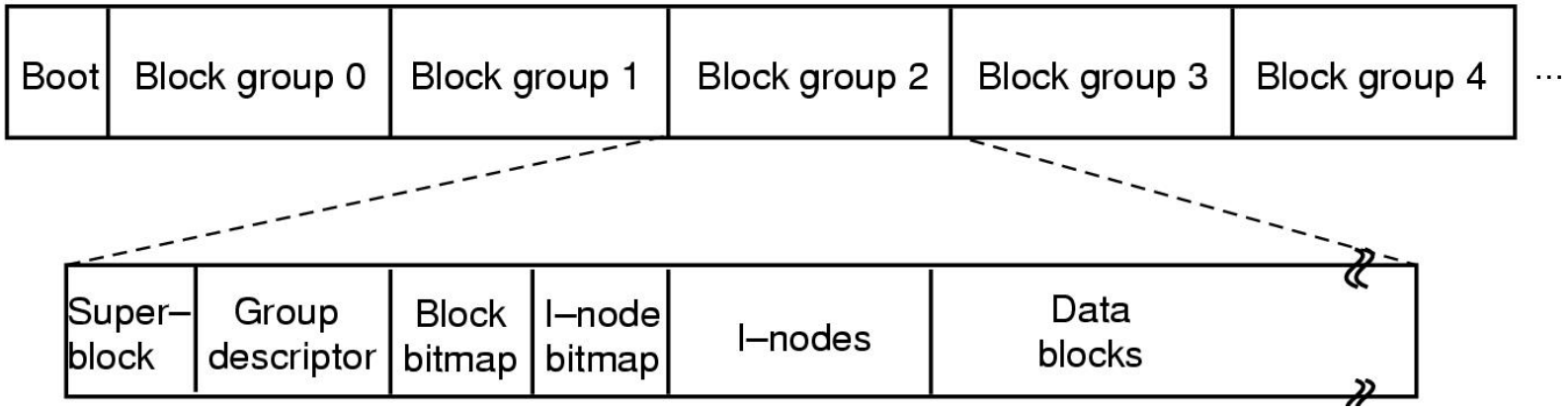

Figure 10-31. Disk layout of the Linux ext2 file system.

The Linux Ext2 File System (2)

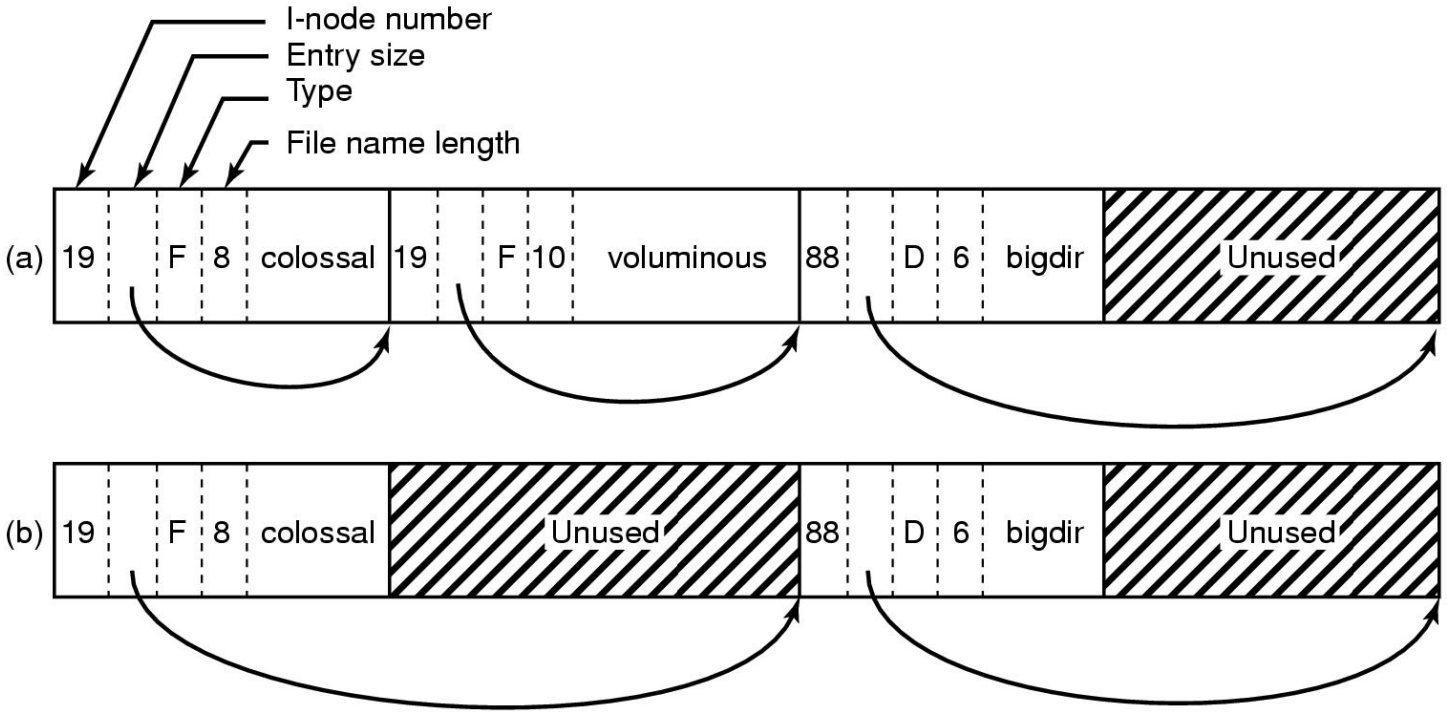

Figure 10-32. (a) A Linux directory with three files. (b) The same directory after the file voluminous has been removed.

The Linux Ext2 File System (3)

Field	Bytes	Description
Mode	2	File type, protection bits, setuid, setgid bits
Nlinks	2	Number of directory entries pointing to this i-node
Uid	2	UID of the file owner
Gid	2	GID of the file owner
Size	4	File size in bytes
Addr	60	Address of first 12 disk blocks, then 3 indirect blocks
Gen	1	Generation number (incremented every time i-node is reused)
Atime	4	Time the file was last accessed
Mtime	4	Time the file was last modified
Ctime	4	Time the i-node was last changed (except the other times)

Figure 10-33. Some fields in the i-node structure in Linux

The Linux Ext2 File System (4)

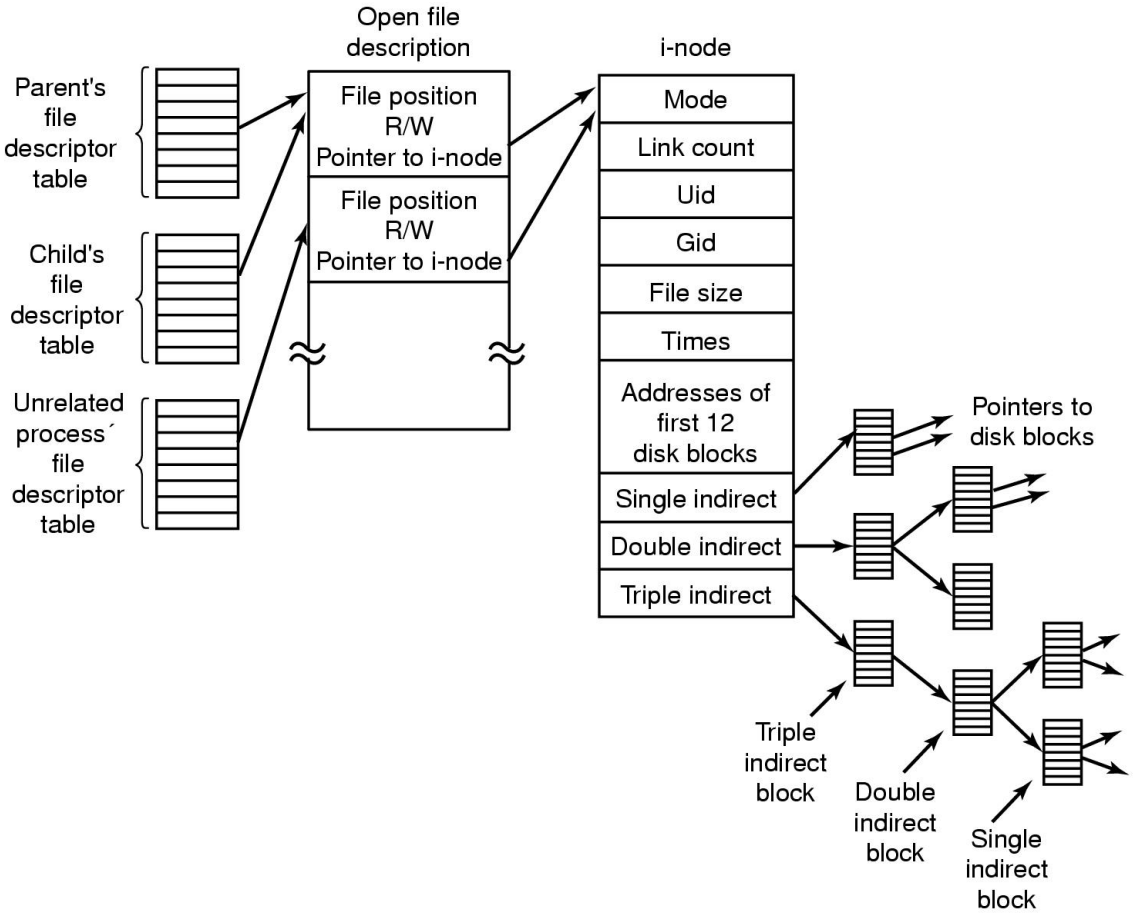

Figure 10-34. The relation between the file descriptor table, the open file description table, and the i-node table.

NFS Protocols

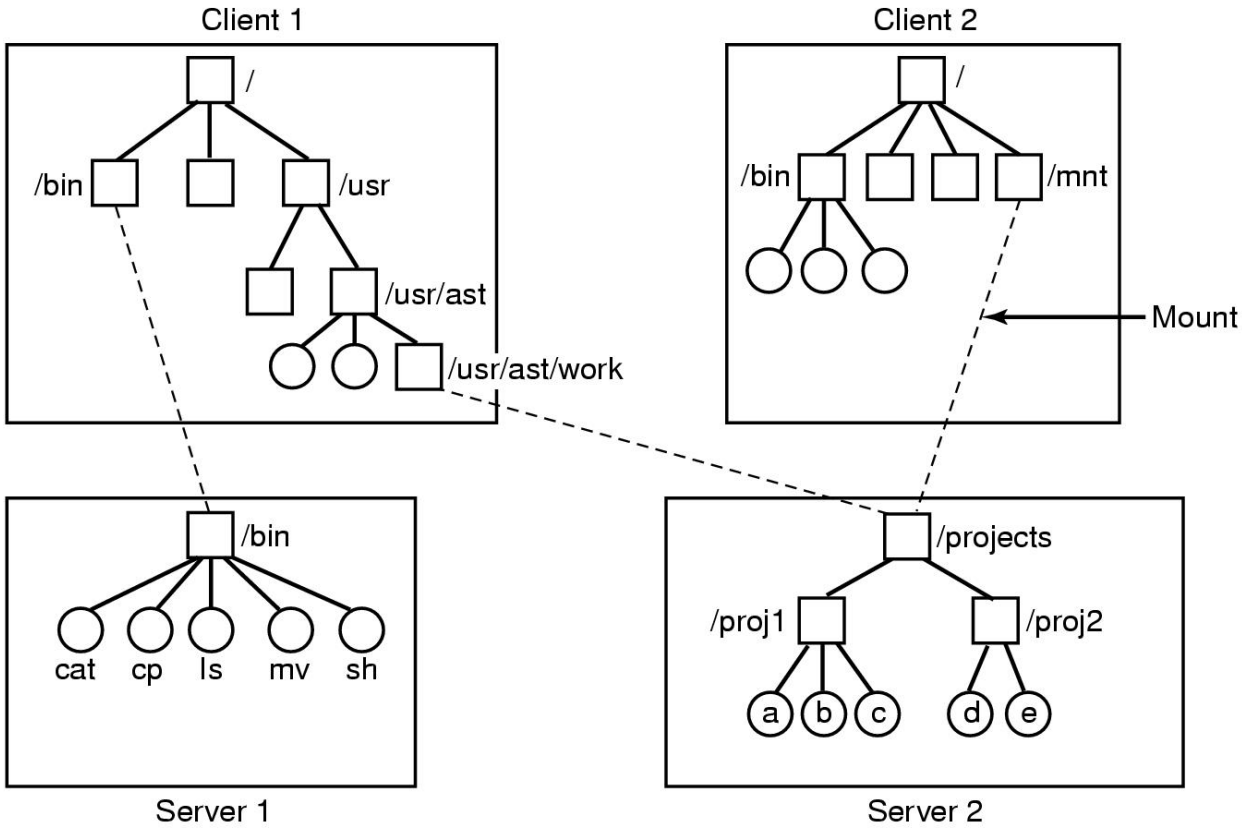

Figure 10-35. Examples of remote mounted file systems. Directories shown as squares, files shown as circles.

NFS Implementation

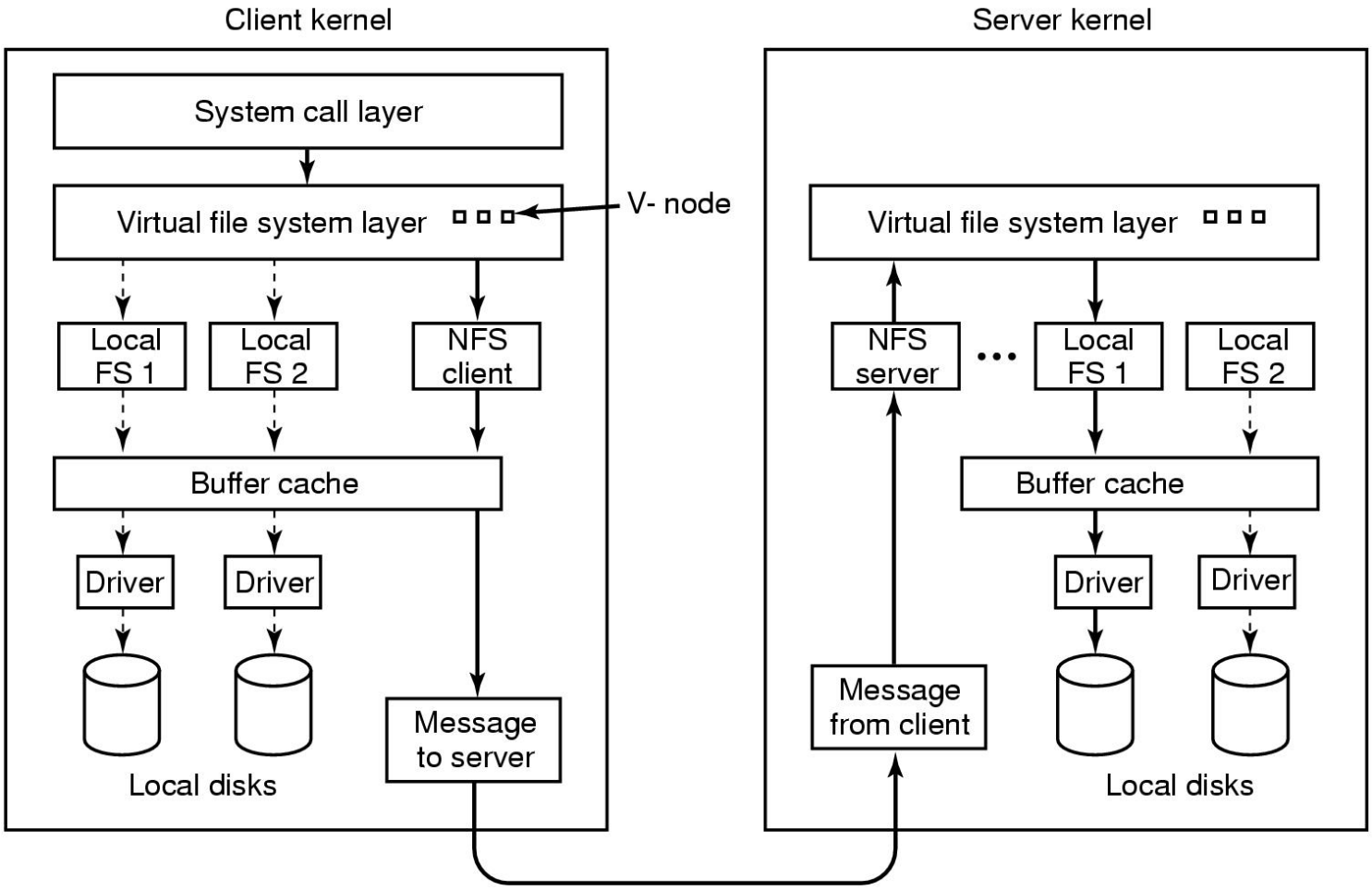

Figure 10-36. The NFS layer structure

Security In Linux

Binary	Symbolic	Allowed file accesses
111000000	rwx------	Owner can read, write, and execute
111111000	rwxrwx---	Owner and group can read, write, and execute
110100000	rw-r-----	Owner can read and write; group can read
110100100	rw-r--r--	Owner can read and write; all others can read
111101101	rwxr-xr-x	Owner can do everything, rest can read and execute
000000000	---------	Nobody has any access
000000111	------rwx	Only outsiders have access (strange, but legal)

Figure 10-37. Some example file protection modes.

Security System Calls in Linux

System call	Description
s = chmod(path, mode)	Change a file's protection mode
s = access(path, mode)	Check access using the real UID and GID
uid = getuid( )	Get the real UID
uid = geteuid( )	Get the effective UID
gid = getgid( )	Get the real GID
gid = getegid( )	Get the effective GID
s = chown(path, owner, group)	Change owner and group
s = setuid(uid)	Set the UID
s = setgid(gid)	Set the GID

Figure 10-38. system calls relating to security.