The UNIX File System

* UNIX: a time-sharing system.

* UNIX: AT&T version, BSD version (Berkeley Software Distribution).

* File: a uniform logical view of information storage.
  - A file is a collection of related information defined by its creator.

* Why file system ?

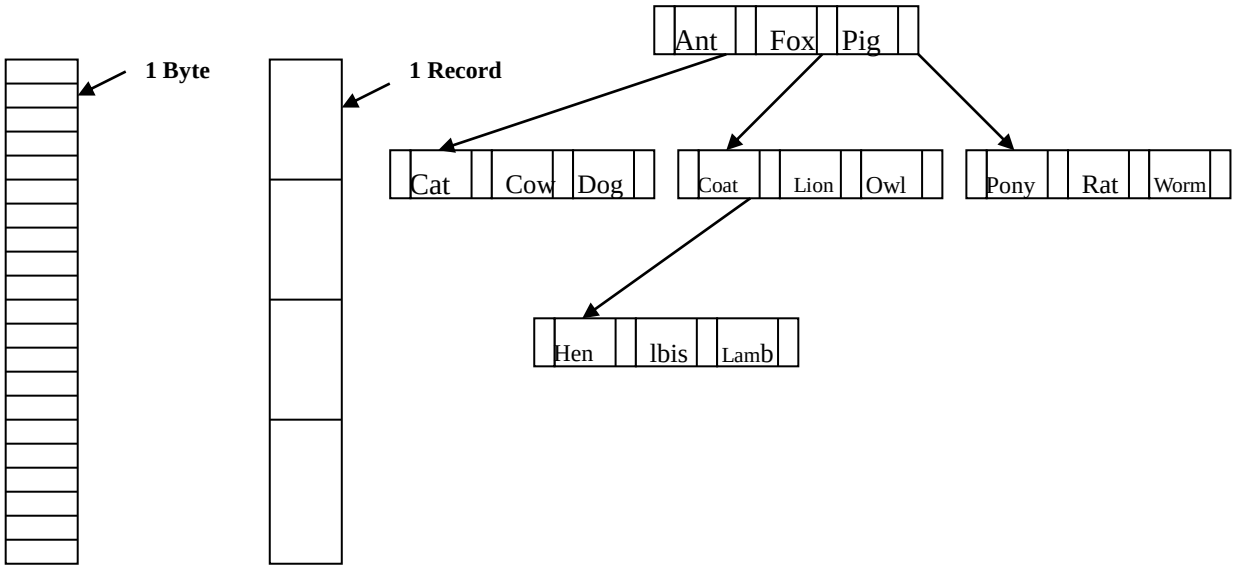
  1) when size of files is large.
  2) for back-up.
  3) sharing.

* We are cared about how files are structed, named, accessed, used, protected, and implemented ==> a file system.

* File System:

  1) the directory structure.
  2) the collection of actual fields.

* File structure: byte sequence, record sequence, tree.


1 Byte
1 Record
Ant
Fox
Pig
Cat
Cow
Dog
Coat
Lion
Owl
Pony
Rat
Worm
Hen
Ibis
Lamb

* File access:

1) sequential access: tape.
2) random access: disk.

* Types of files:

1) test files. (characters)
2) source files.
3) object files. (binary files)
4) database. (record)

or

1) regular files (user files): ASCII files/binary files.

- ASCII files: they can be displayed and printed as is, and they can be edited with an ordinary test editor.

- binary files: consists of a collection of library procedures compiled but not linked.

2) directories (system files).

3) character special files: related to I/O terminal, printer, network.

4) block special files (model disk).

The UNIX File System

* A File system is the organization framework that specifies how a mass storage device is organized.

-File systems usually contain a map, which is called the i-node table on the UNIX system, that is used to locate a file, given its name.

*UNIX File Types

1) ordinary files: text and binary files.

2) directory files: collections of files.

. A directory file stores the names of the files it contains plus information that is used to locate and access the files.

3) device files: special device files.

. A device file is a means of accessing a hardhare device, usually an I/O device. Ex. a pseudo-tty.

4) symbolic link files.

. A symbolic link is a way to make a new name for an existing files. It makes it easy to place “copies” of things whenever they are needed, without the overlead (and duplications) of truly making copies.

5) the named pipes.

* File Access Modes + Protection + Security

* The directory system can be viewed as a symbol table that translates file names into their directory entries.
    ==> the directory itself can be organized in many ways.
    ==> problem: how to search the entry ?
        1) linear search.
        2) hash.

* UNIX directory
    - files are organized in tree-structured directories.
    - absolute/relative path name.
    - working directory.

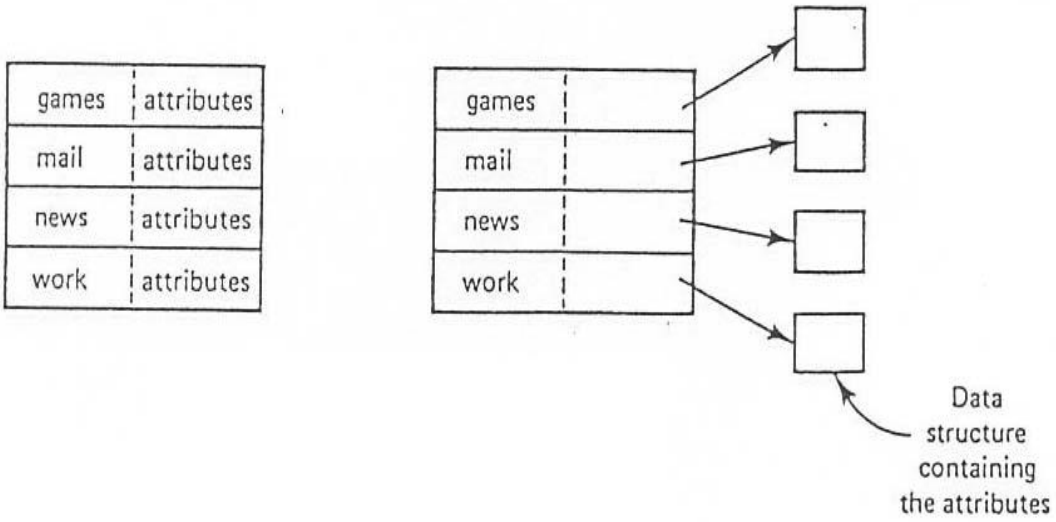
* Implementing Directories

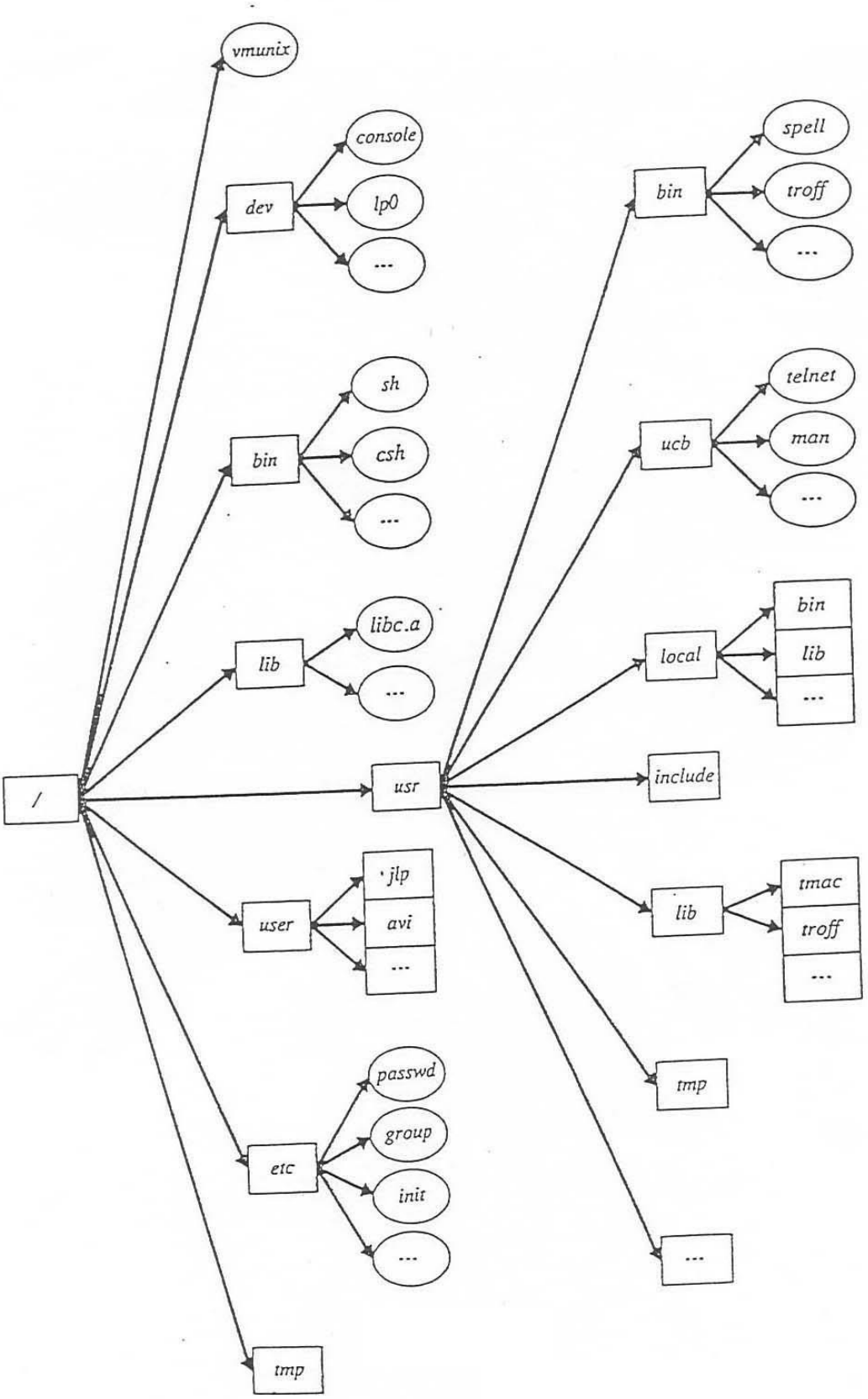
- When a file is opened, OS uses the path name supplied by the user to locate the directory entry.
  The directory entry provides the information needed to find the disk blocks.

- Depending on the system, the information may be the disk address of the entire file (continuous allocation), the number of the first block or the number of the I-node (in UNIX).

- The main function of the directory system is to map the ASCII name of the file to the information needed to locate the data.

- Issue: where the attributes of a file should be stored ?
    1) store them directly in the directory entry.
    2) i-node: store the attributes in the i-node, rather than directory entry.


games attributes
mail attributes
news attributes
work attributes
games
mail
news
work
Data structure containing the attributes


/
vmunix
dev
console
lp0
...
bin
sh
csh
...
lib
libc.a
...
usr
bin
spell
troff
...
ucb
telnet
man
...
local
bin
lib
...
include
lib
tmac
troff
...
tmp
...
user
jlp
avi
...
etc
passwd
group
init
...
tmp

Directory	Description
/bin	Frequently used system binaries
/dev	Special files for I/O devices
/etc	Miscellaneous system administration
/lib	Frequently used libraries
/tmp	Some utilities generate their temporary files here
/usr	All user files are in this part of the tree
/usr/adm	System accounting
/usr/ast	Home directory for the user whose login name is ast
/usr/bin	Other system binaries are kept here
/usr/include	System header files
/usr/lib	Libraries, compiler passes, miscellaneous
/usr/man	Online manuals
/usr/spool	Spooling directories for printer and other daemous
/usr/src	System source code
/usr/tmp	Other utilities put their temporary files here

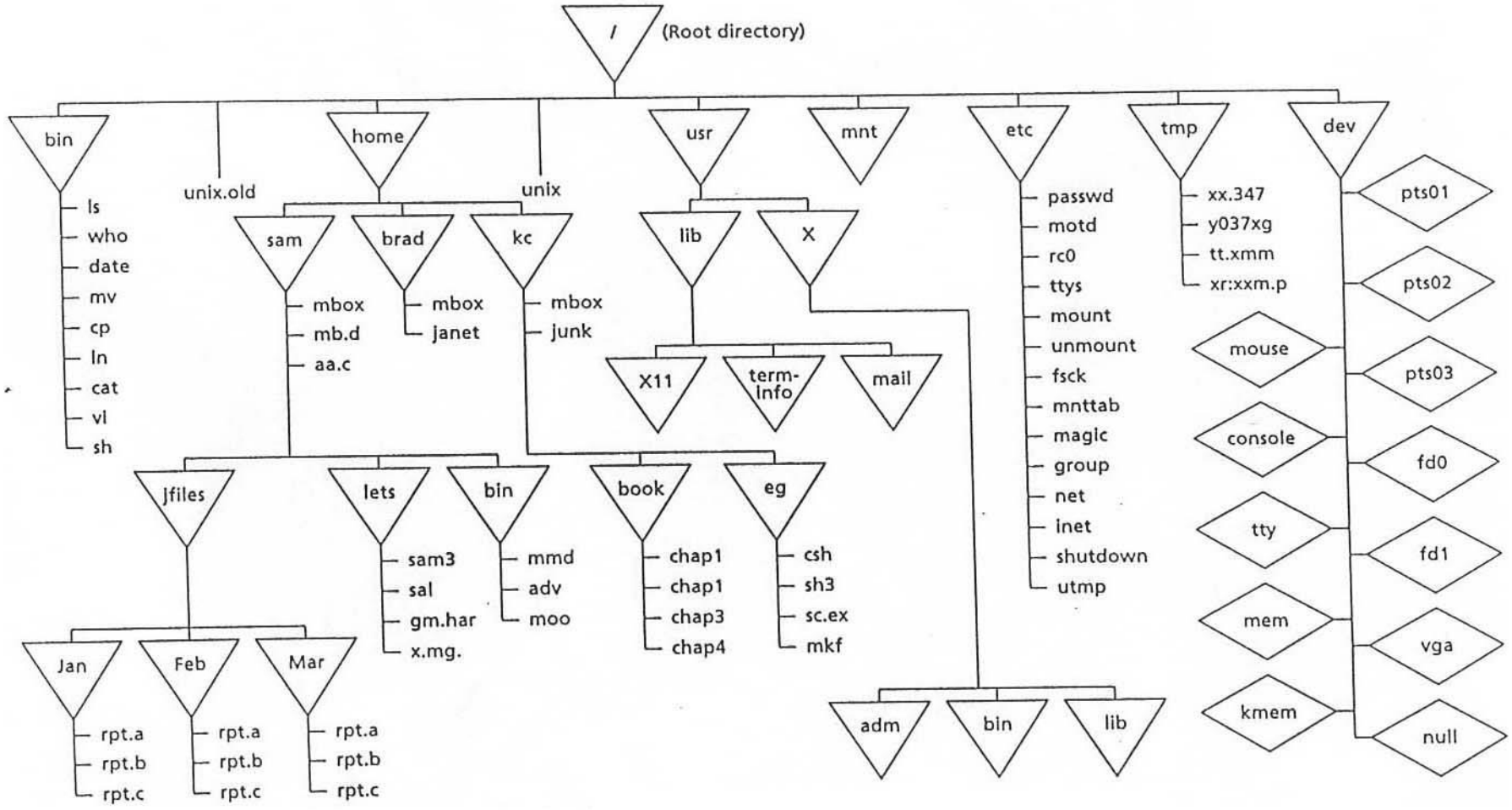


Figure 1 ■ A Simplified Diagram of a Typical UNIX File System
In this diagram, directory files are shown in triangles, special device files are shown in diamonds, and ordinary files are shown without borders.

* File system implementation

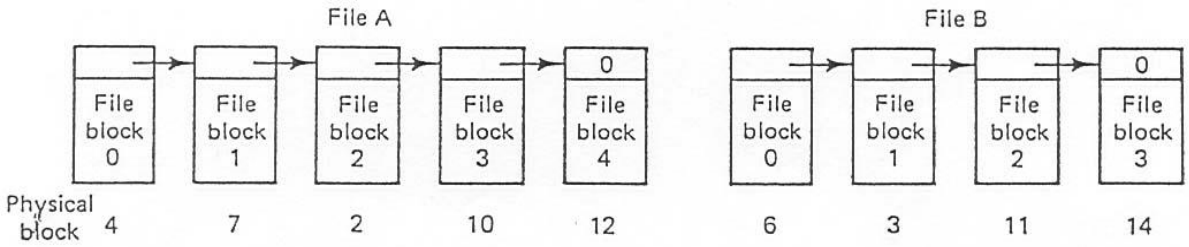
- implement file storage; keep track of which disk blocks go with which files.

1) continuous allocation.

- simple, good performance.
- but not flexible (must know the size of the file in advance), large fragmentation.

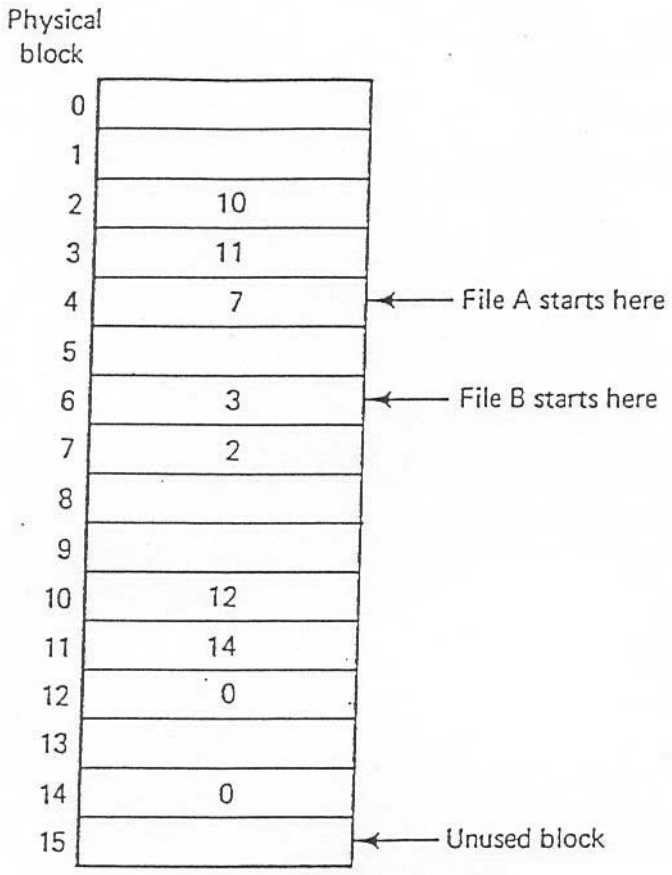
2) Link list allocation: a linked list of disk block.

- no space is wasted to disk fragmentation.
- it is sufficient for the directory entry to merely store the disk address of the first block.
- however, reading a file sequentially is straightforward.
- random access is slow.



Storing a file as a linked list of disk blocks.

3) Linked list allocation using an index.
- the entire block is available for date.
- random access is much easier.
- however, the entire table must be in memory.



Linked list allocation using a table in main memory.

4) I-node in UNIX.

- associate with each file a little table called i-node (index-node), which lists the attributes and disk addresses of the file's blocks.

- for small files: all the necessary information is in i-node.

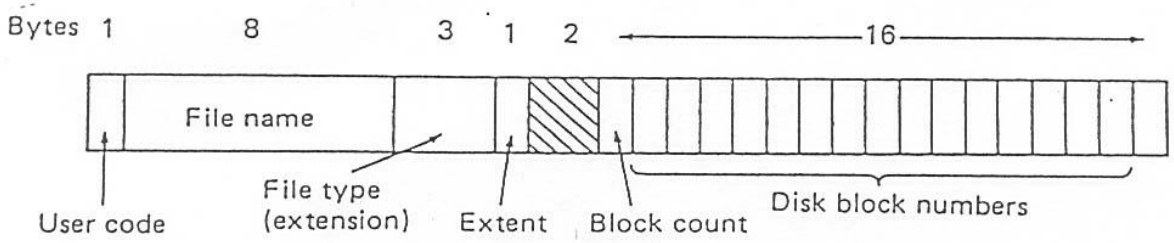
- for large files: one of the addresses in i-node is the address of a disk block called a single direct block.

- file descriptor is an index to a small table of open files for this process.

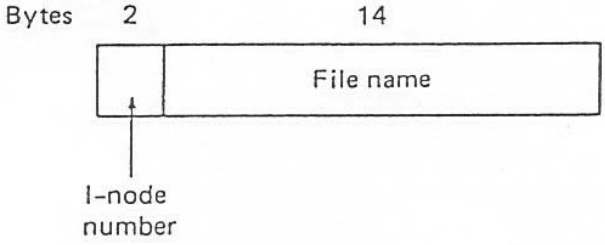
- in this table, each entry contains an pointer to a file structure, which in trun, points to the i-node.

- ls -l
  after get i-node number, copy i-node information from disk to memory (called in-core i-node).

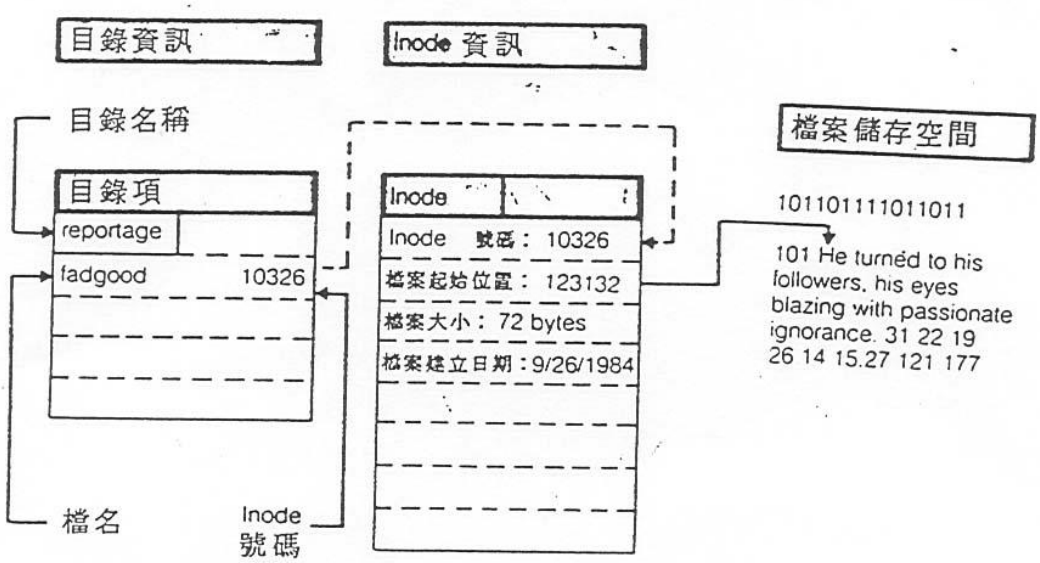
- each i-node contains 15 direct link pointer (each of them pointing to a page <= 12 blocks), 3 inderect block pointers (single, double, triple: indirect block pointer).



(i) A directory that contains the disk block numbers for each file.



(ii) A UNIX directory entry.



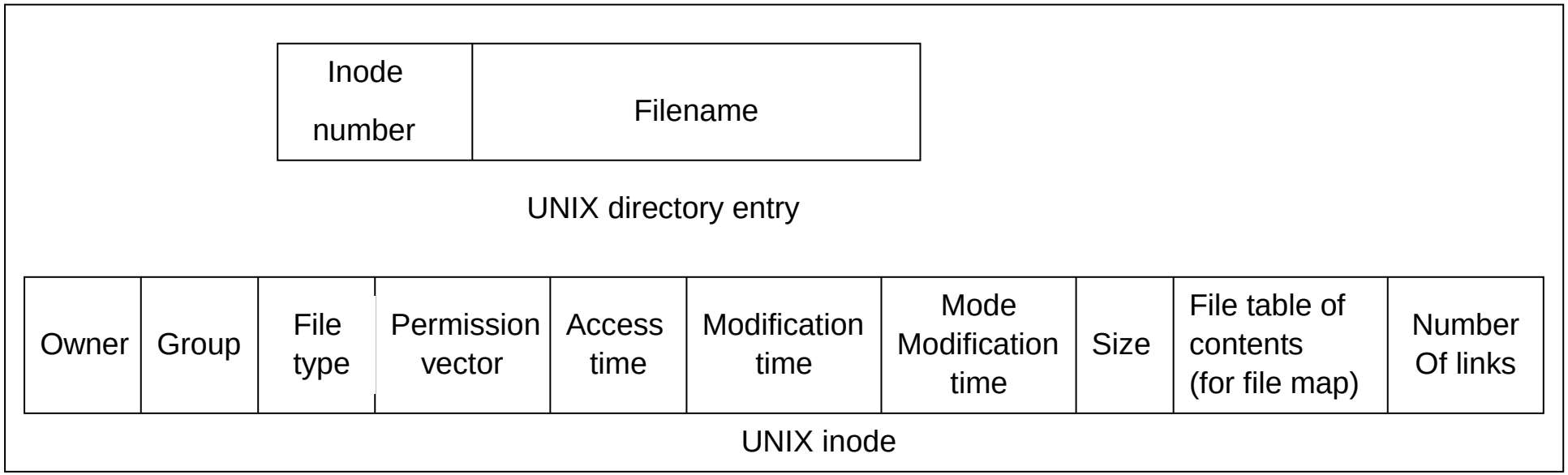


Figure 2

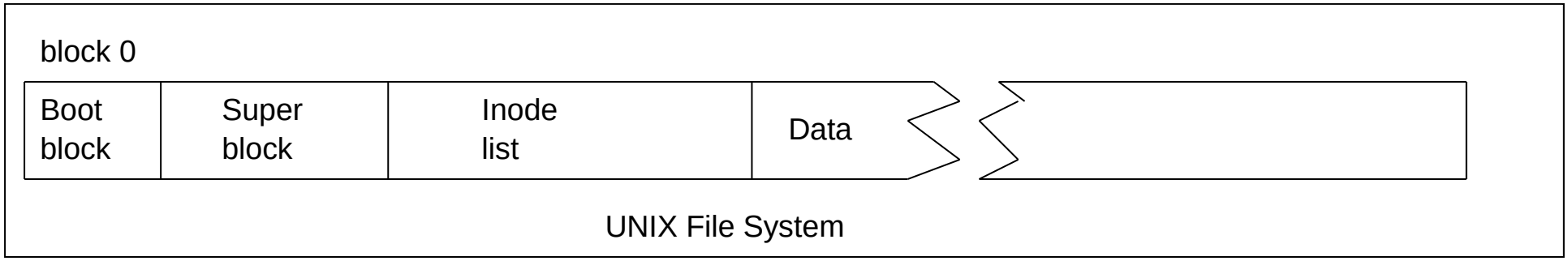


Figure 3

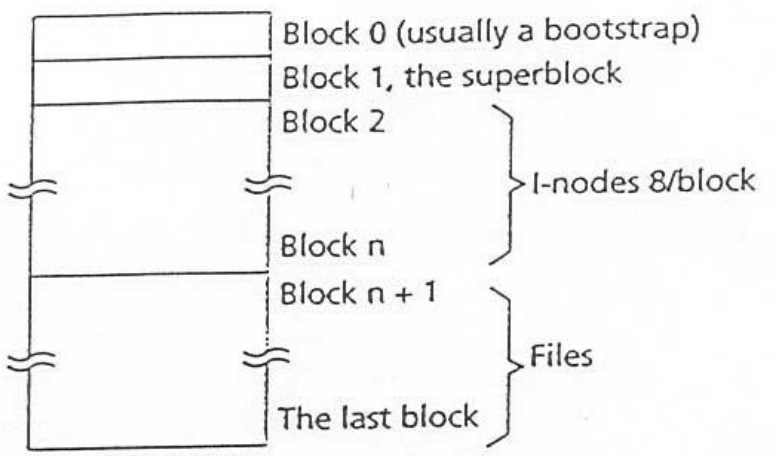


Figure 3 ■ File Systems Traditionally Used This Simple Layout

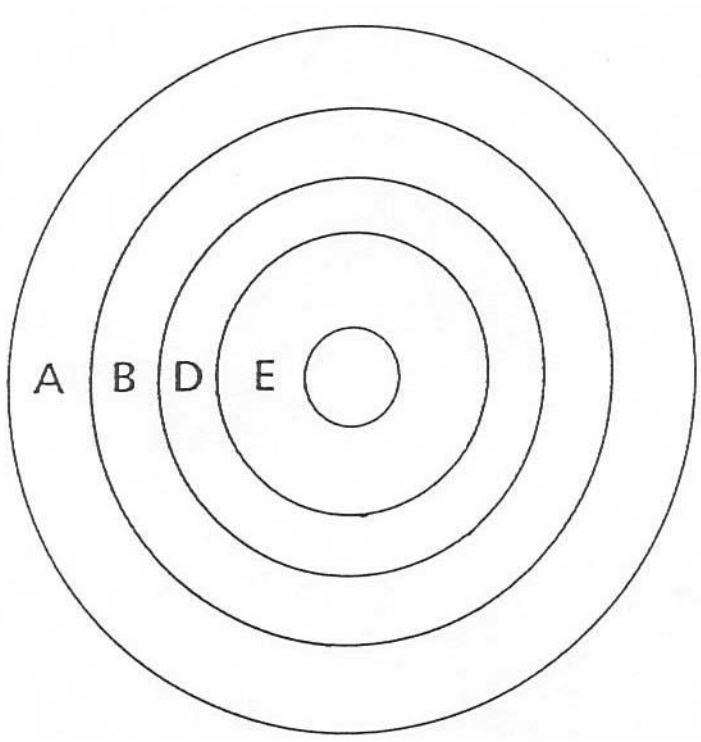


Figure 4 ■ Disks are Usually Divided into Partitions or Slices

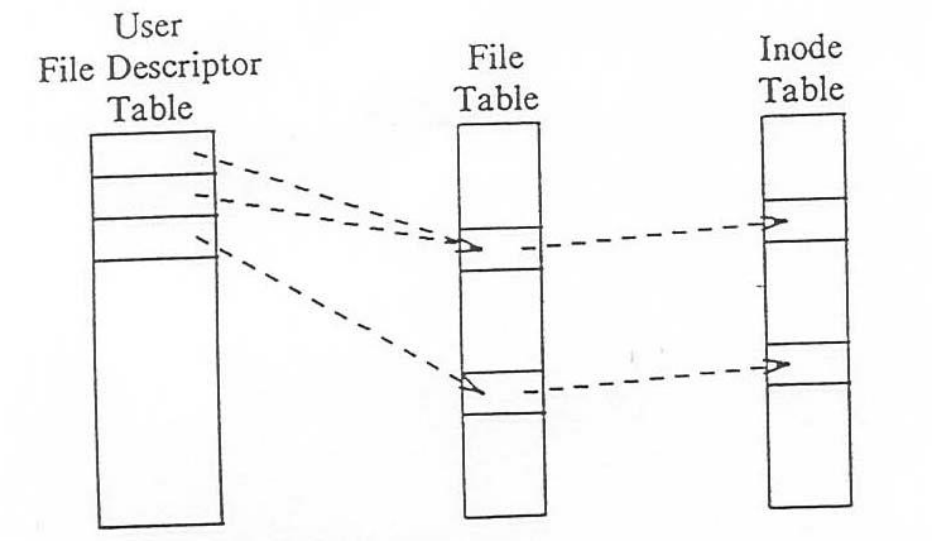


Figure 5 ■ File Descriptors, File Table, and Inode Table

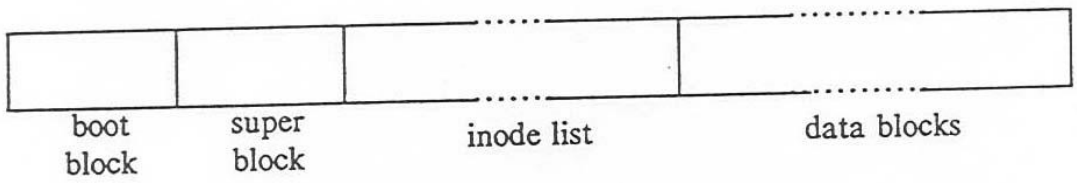
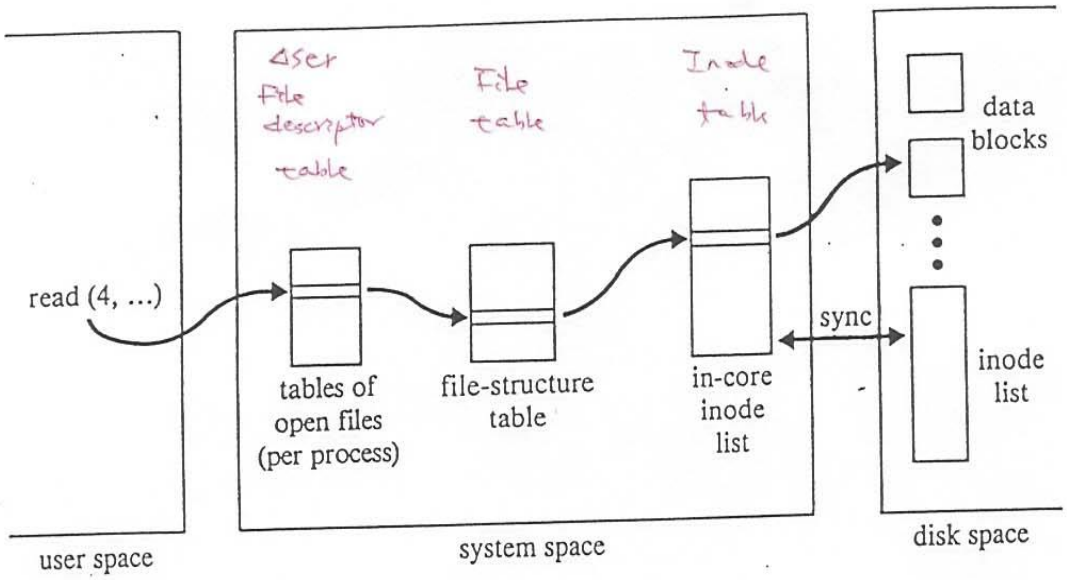


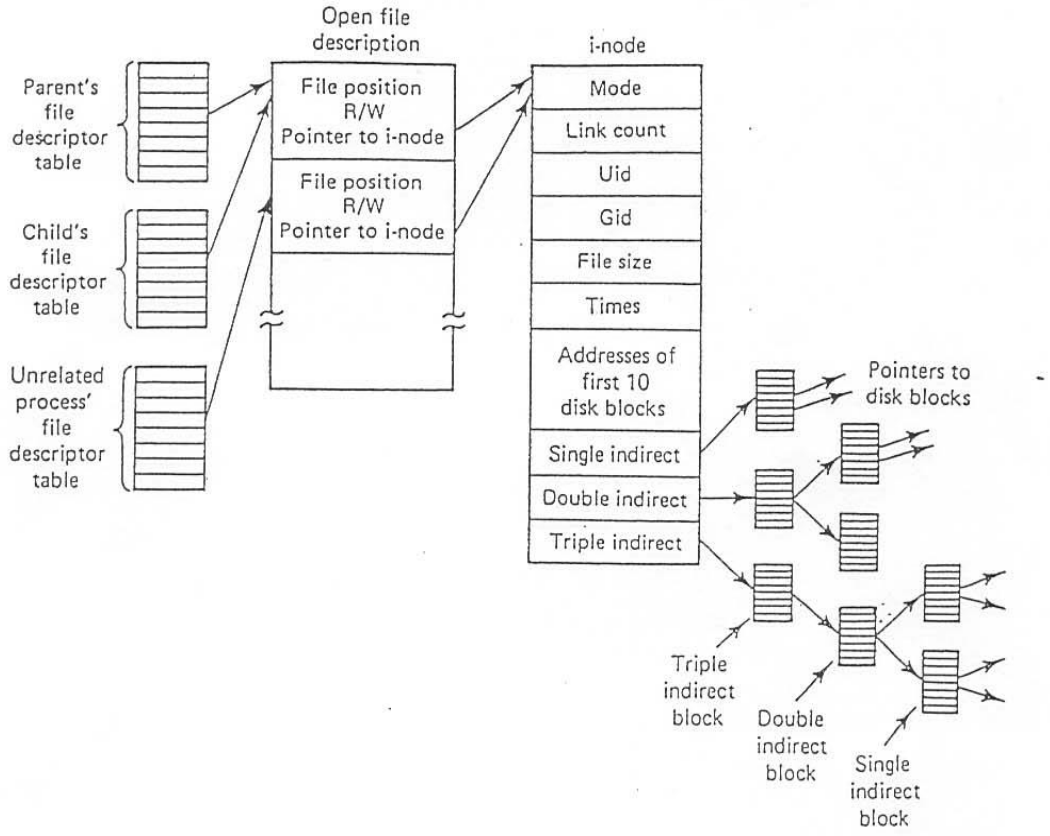
Figure 6 ■ File System Layout

A file system has the following structure (Figure 6).

- The *boot block* occupies the beginning of a file system, typically the first sector, and may contain the *bootstrap* code that is read into the machine to boot, or initialize, the operating system. Although only one boot block is needed to boot the system, every file system has a (possibly empty) boot book.
- The *super block* describes the state of a file system – how large it is, how many files it can store, where to find free space on the file system, and other information.
- The *inode list* is a list of inodes that follows the super block in the file system. Administrators specify the size of the inode list when configuring a file system. The kernel references inodes by index into the inode list. One inode is the *root inode* of the file system: it is the inode by which the directory structure of the file system is accessible after execution of the *mount system* call.
- The data blocks start at the end of the inode list and contain file data and administrative data. An allocated data block can belong to one and only one file in the file system.



File-system control blocks.



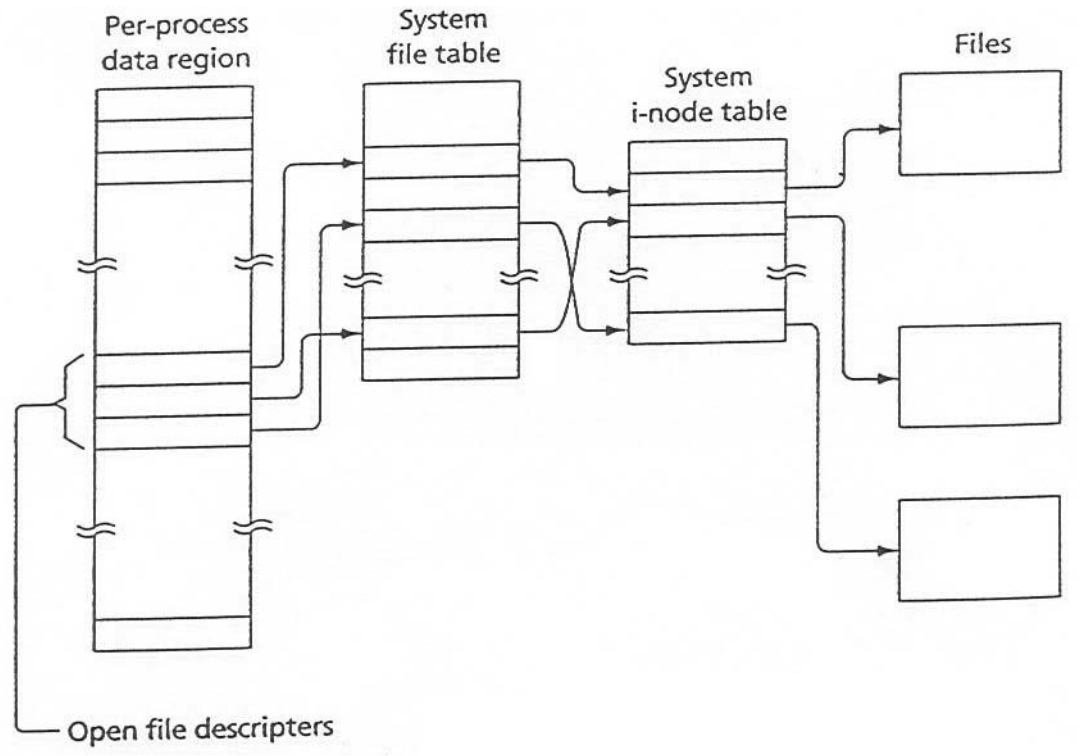
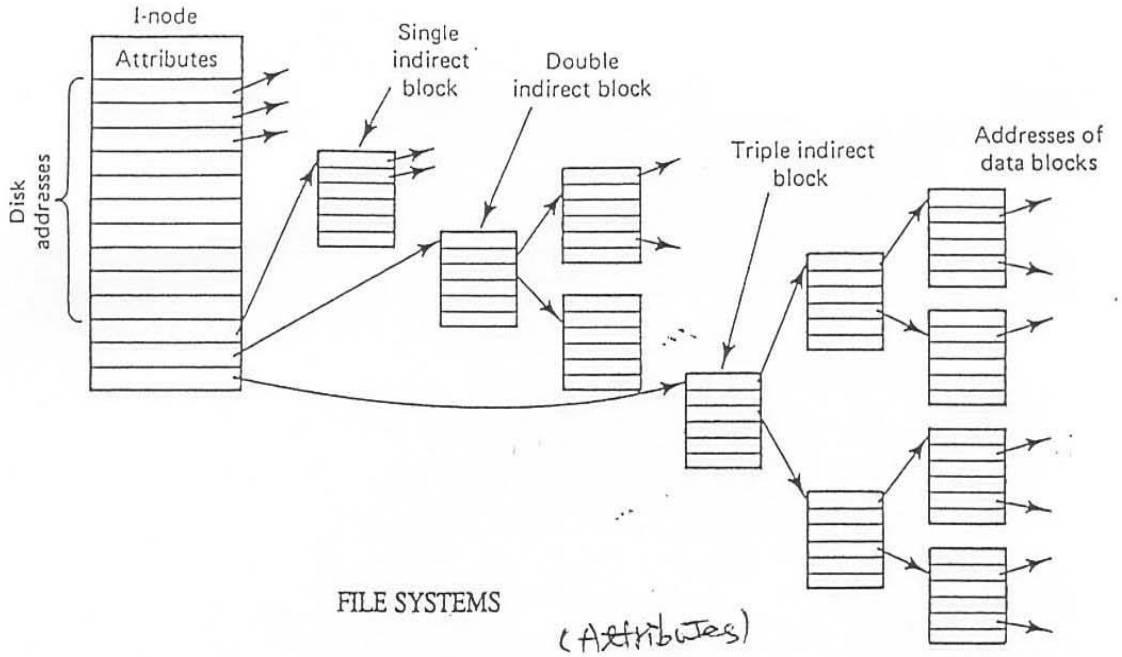


Figure 7 ■ The Kernel Data Structure for Accessing Files



FILE SYSTEMS

(Attributes)

Field	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	Id of person who created the file
Owner	Current owner
Read-only flag	0 for read/write, 1 for read only
Hidden flag	0 for normal, 1 for do not display in listings
System flag	0 for normal file, 1 for system file
Archive flag	0 has been backed up, 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file, 1 for binary file
Random access flag	0 for sequential access only, 1 for random access
Temporary flag	0 for normal, 1 for delete on process exit
Lock flags	0 for unlocked, nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time file was created
Time of last access	Date and time file was last accessed
Time of last change	Date and time file was last changed
Current size	Number of bytes in the file
Maximum size	Maximum size file may grow to

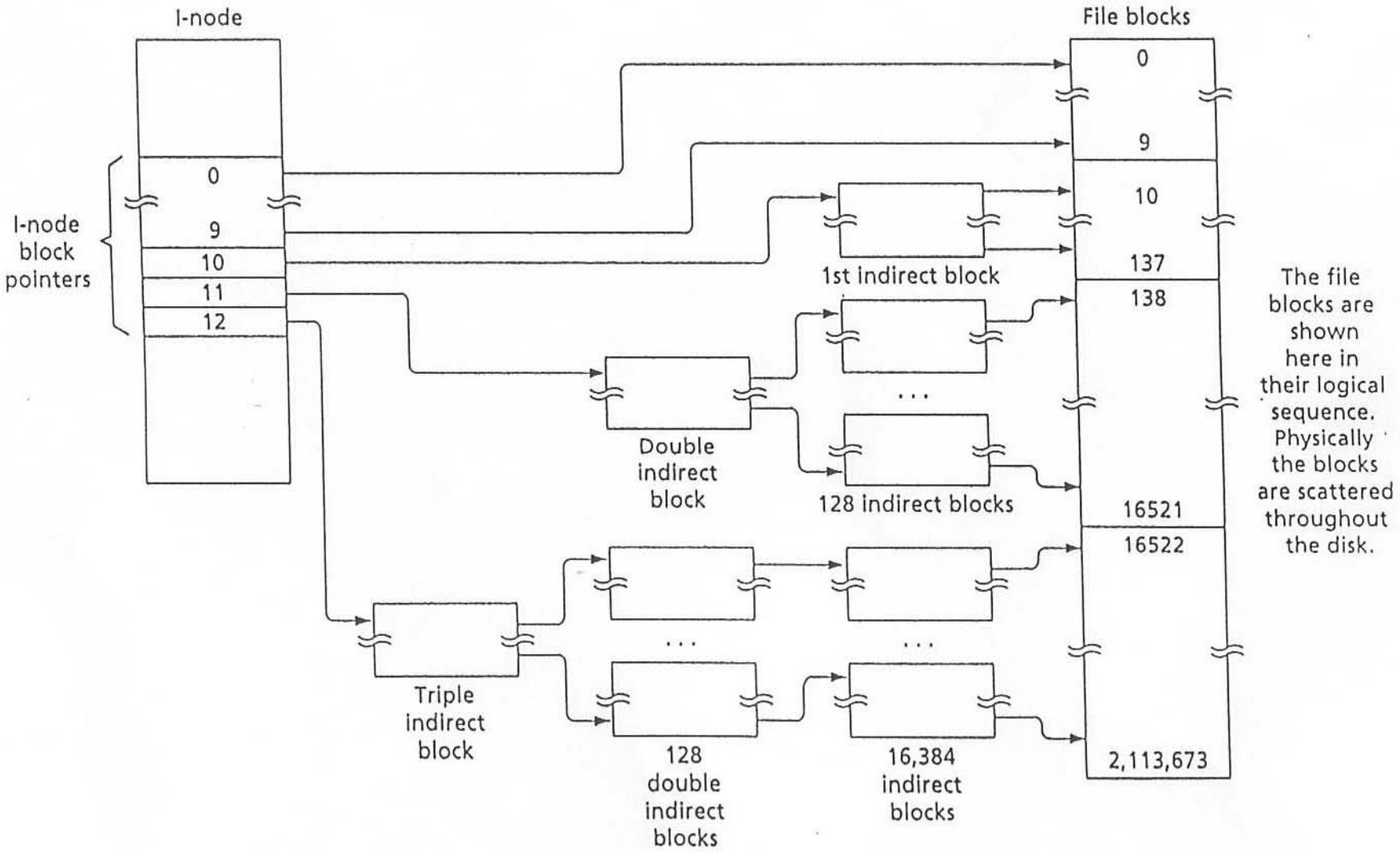
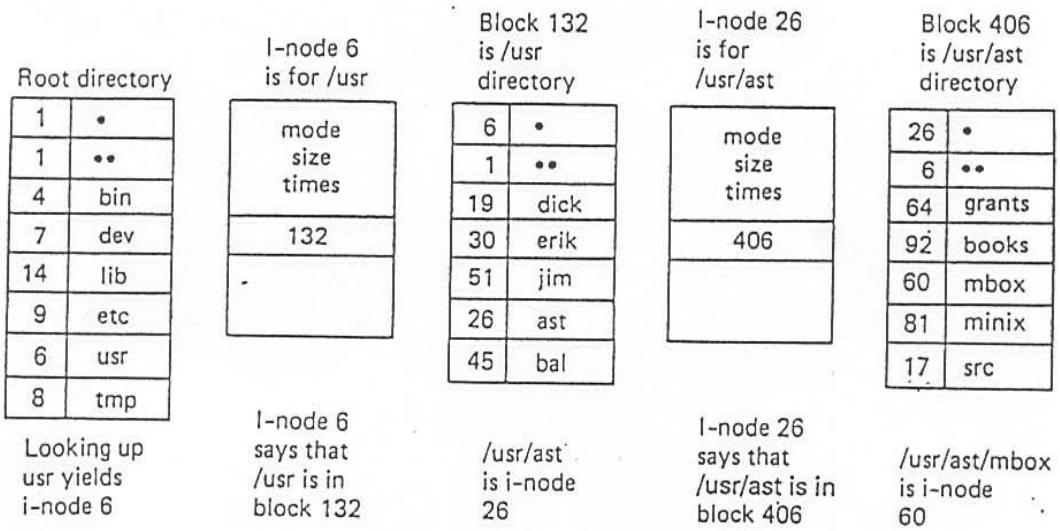


Figure 8 ■ The I-node, showing How I-node Block Pointers Are Used to Locate the Blocks in a File

Field	Description (i-node information)
st_mode	Mode word containing the protection bits
st_ino	I-node number, used to identify the file
st_dev	Device on which the file resides
st_nlink	Number of links to the file
st_uid	User id
st_gid	Group id
st_size	File size in bytes
st_atime	Time of last access
st_mtime	Time of last modification
st_ctime	Time of this information was last changed

The structure used to return information for the STAT system call.



The steps in looking up */usr/ast/mbox*.

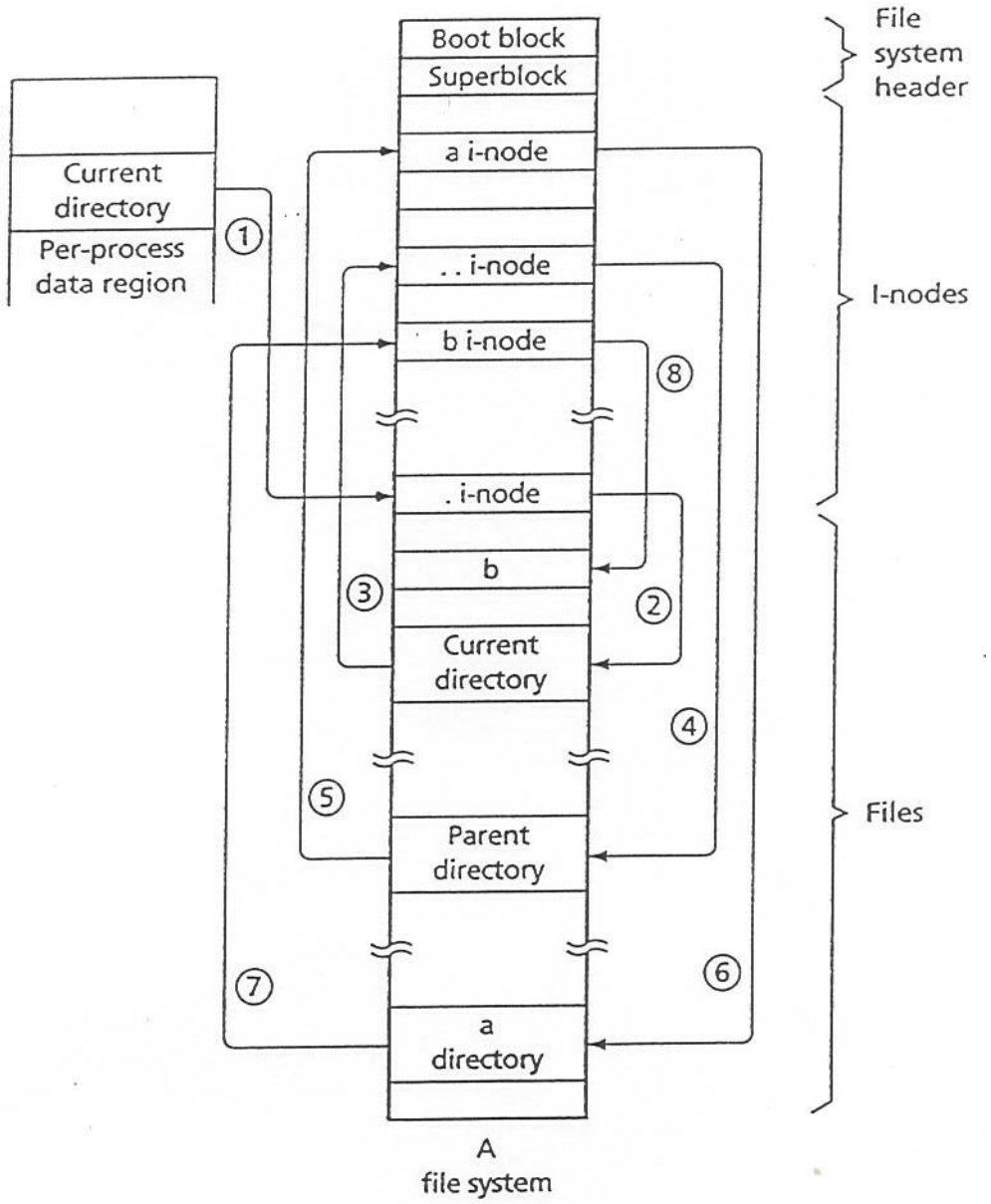


Figure 10 ■ Kernel Operations for Following the Pathname ../a/b

* File Operations:

1) create
- find spce on disk for the file.
- insert the directory entry, records the name of the file and the location in the file system.

2) open
- the open operation takes a file name, searches the directory, copying the directory entry into the table (in memory) of open files; then, OS returns an pointer to the entry in the table of open files, avoiding any further searching.

3) write
- input: the file name + data.
- according to the file name, find the directory entry.
- the directory entry will need to store a pointer to the current block of the file. (write pointer)

4) read
5) reset
6) delete
7) link
8) unlink. (only when the number of linked file names = 0)

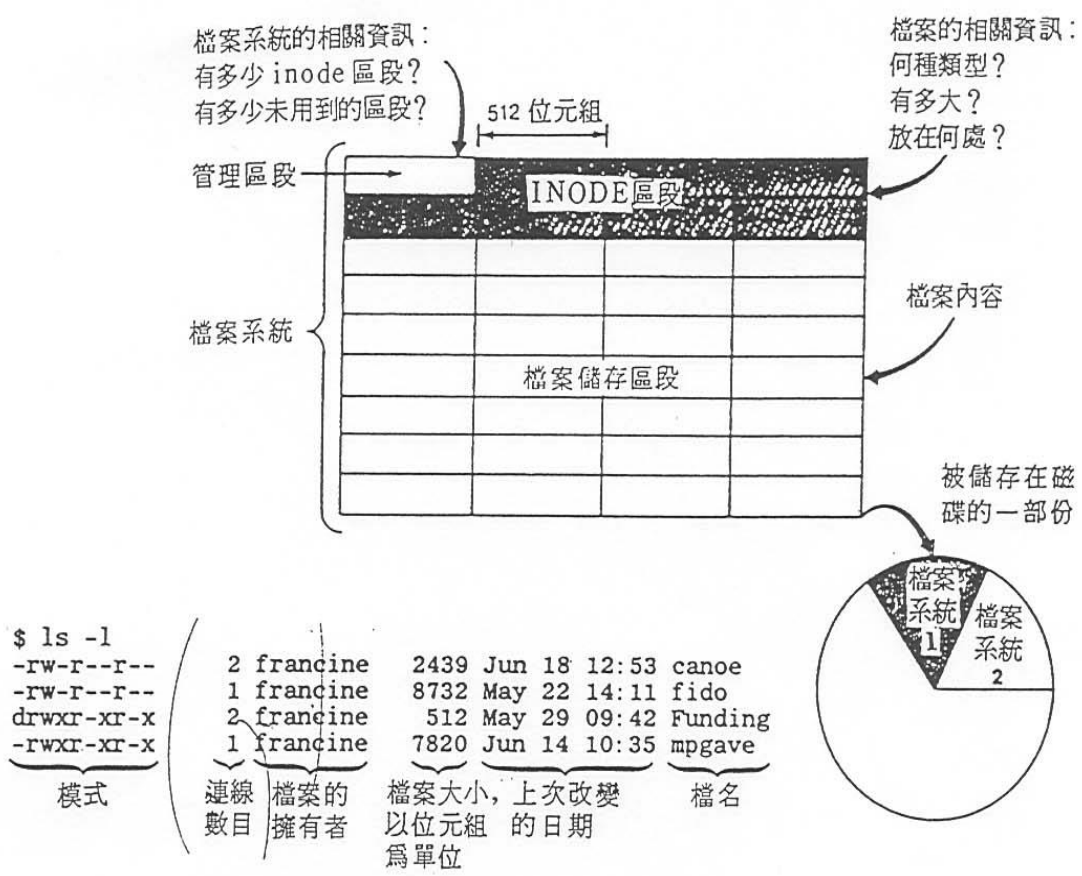
* To avoid to search the directory entry again, when a file operation is requested, an index into to this table is used (returned), so no searching is required.

* Disk space management: page/segment
    - block size.
    - keep track of free blocks.
    - disk quotas of each user.

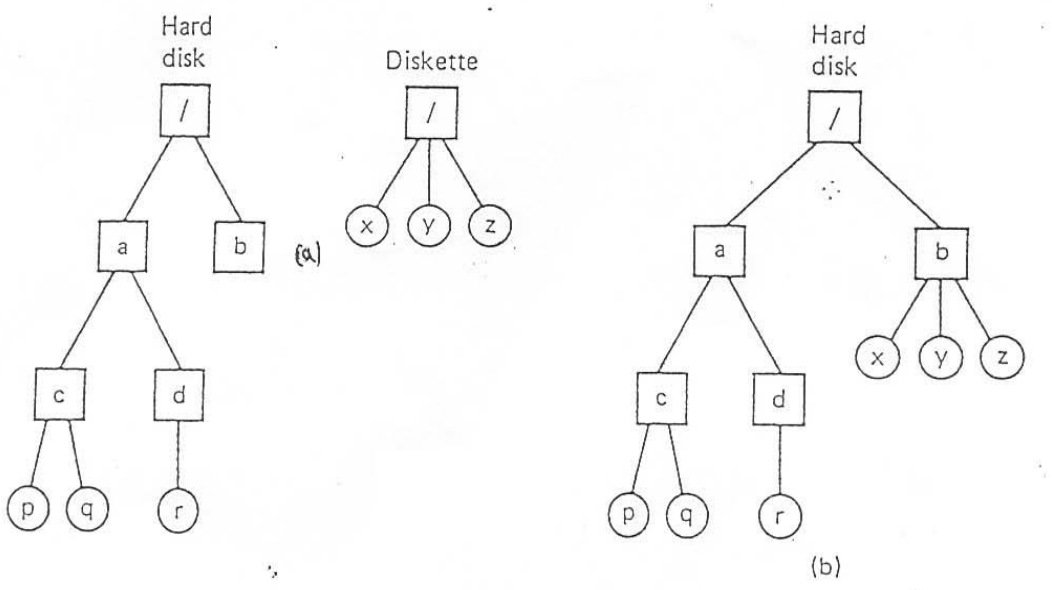
* File system reliability: backup, recovery, consistency (concurrency control).

* Security and Protection
    - (owner, group, universal) + (rwx)
    -rwxrwxrwx (777)

* Mount



ls -l



(b) After mounting.

* Managing your Files (utilities for file management)

-pwd, cd, ls, rm, mv, cp.

-ln : create links.

. It creates a new name which references the original file. Only one copy exists although it has two names.

a) hard links (traditional forms).

-Ex. In old new.
Those two files have the same I-node number.

-Two problems:

(1) They only work within a file system because they are based on I-node.

(2) They can be very transitory because they operate at such a low level in the UNIX file system.

b) symbolic links.

-Ex. Ln –s old new.

-A symbolic link operates at a higher level then a hard link because a symbolic link refers to a file by name, not through the I-node table. These two files (old, new) have different I-node numbers.

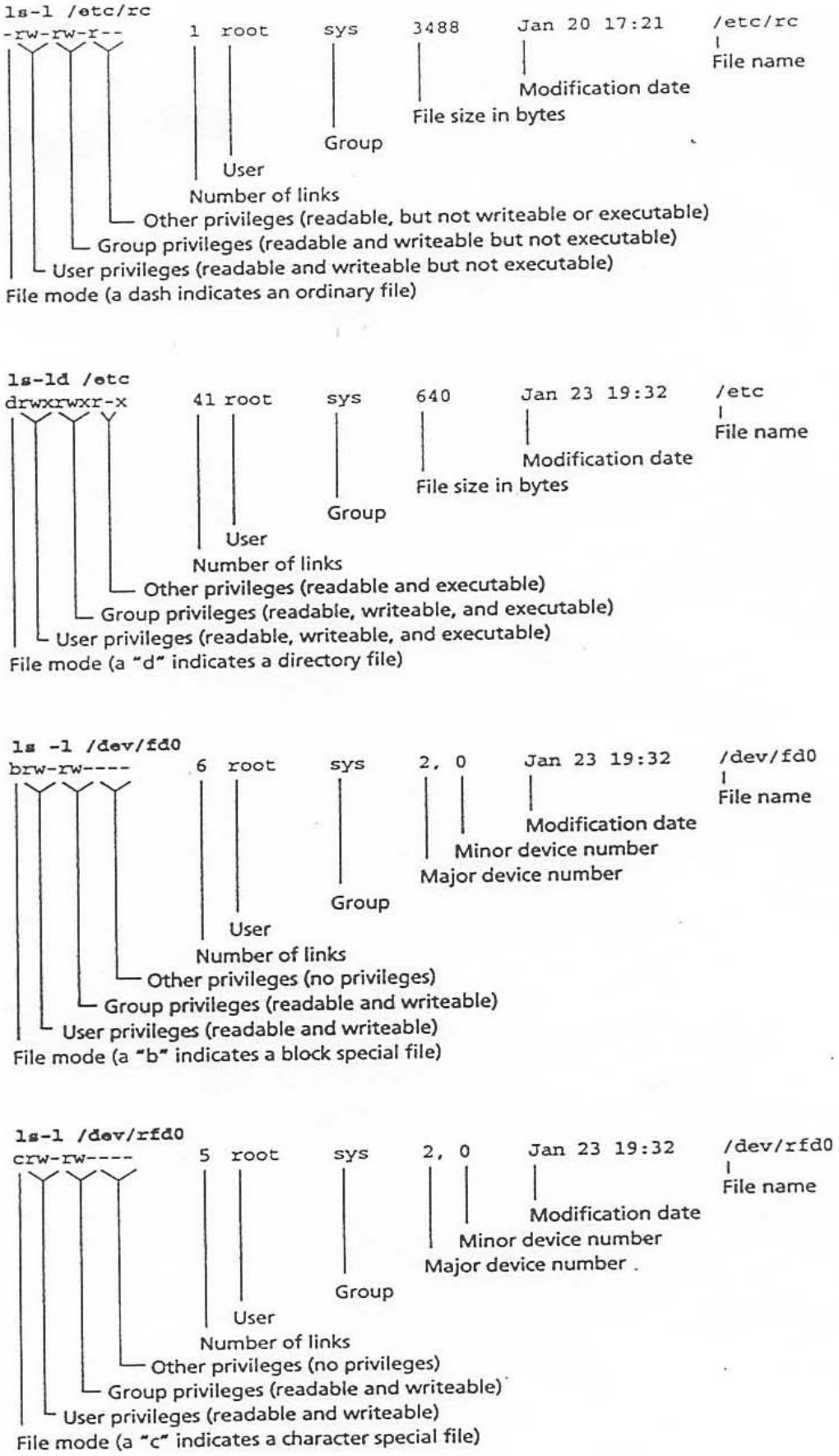


Figure 11 ■ The Long Format Output of the Ls Command

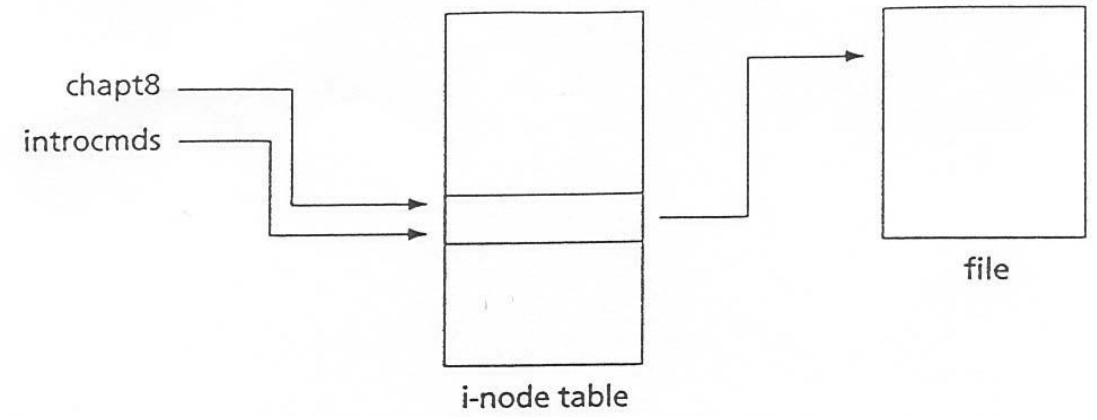


Figure 12 ■ A Hard Link Between introcmds and chapt8
A hard link operates at the level of the I-node table. In this example, the file names introcmds and chapt8 both refer to the same I-node, hence both are links to the same file.

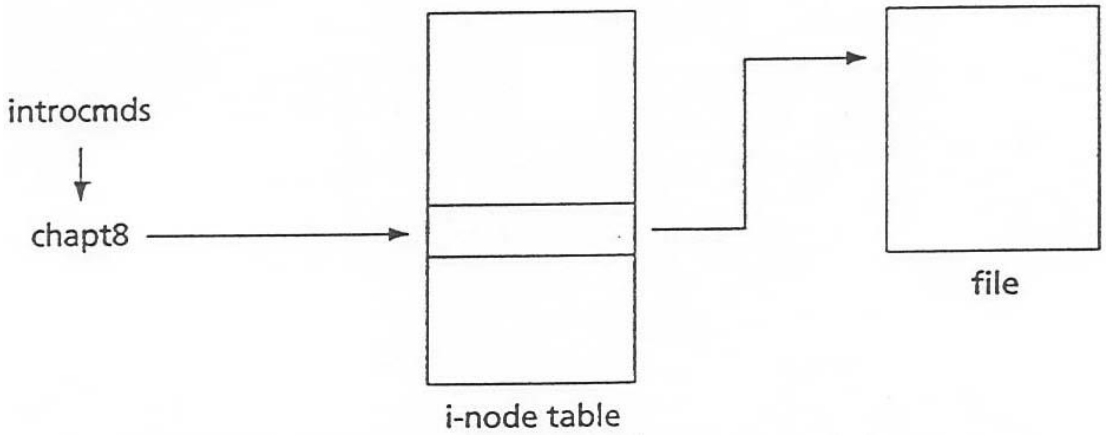


Figure 13 ■ A Symbolic Link Between introcmds and chapt8
Symbolic links make one file name refer to another. In this example, the file name introcmds is a symbolic link to the file chapt8. The file chapt8 may be modified in any way, but so long as the name chapt8 exists, the symbolic link will remain valid.

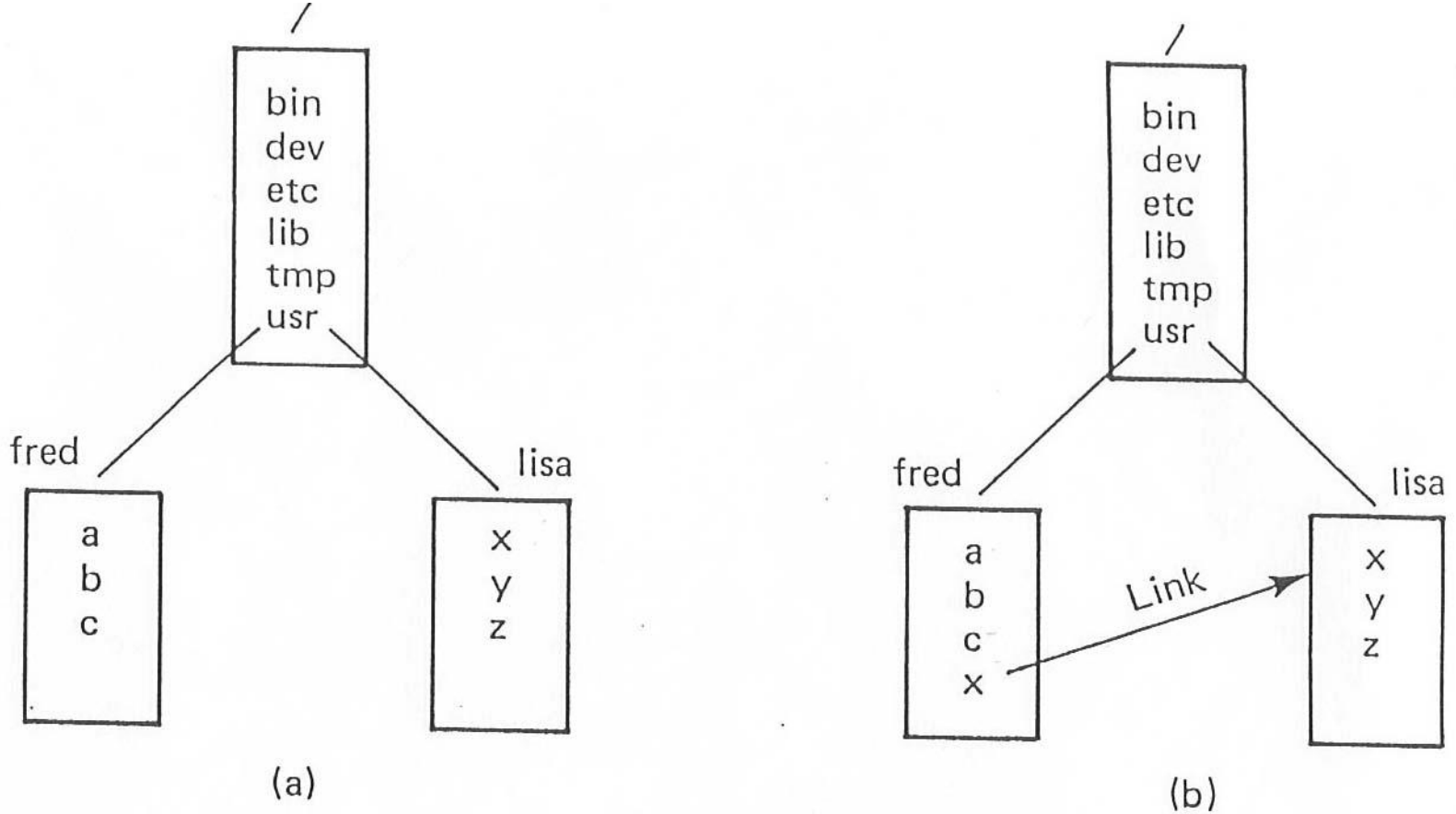


Figure 14 ■ (a) Before linking. (b) After linking.

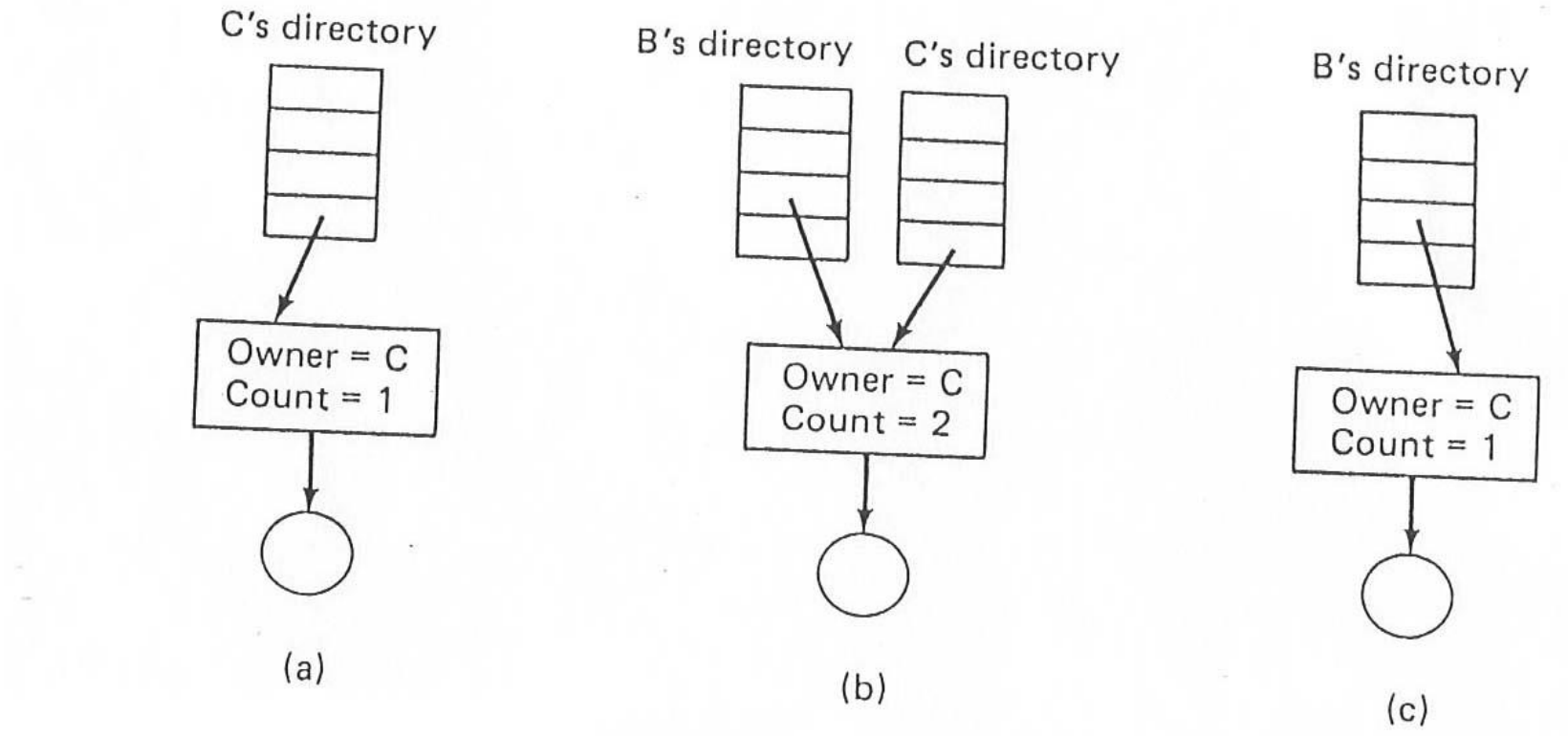


Figure 15 ■ (a) Situation prior to linking. (b) After the link is created.

(c) After the original owner removes the file.

-chomd (change file modes).

-chown (change file owners).

-chgrp (change file groups).

-mkdir, rmdir.

-find : search for files.

. Find examines a file system subtree, not just a single directory, looking for files that match a set of criteria.

Ex. find . –name checklist -print
find /usr –name ‘v*[0-9]’ -print
find /usr –size + 1000 -print
find /usr –mtime –1 -print
find / -name core –exec rm {};

-pack and compress : save space.

. When you compress a file, the programs create a new program of the same name but with a .z(pack) or a .Z(compress) suffix, and then delete the original.

Table 1 ■ Find Command-line Options

Searching

-atime n	Find files accessed n days ago. –n means less than n days, n means exactly n days, while +n means more than n days.
-ctime n	Find files whose i-node was modified n days ago. –n means less than n days, n means exactly n days, while +n means more than n days.
-depth	Perform a depth first search, which means examine a directory's subdirectories before examining the files in the directory. Always true.
-follow	Follow symbolic links. Always true.
-fstype type	Confine the search to file systems of the specified type. Common file system types are rfs,nfs,and s5. Always true.
-group groupnm	Search for files belonging to the specified group.
-inum num	Search for files that have the specified inum.
-links n	Search for files with n links.
-local	Search only for files on local file systems.
-mount	Search for files within a single file system.
-mtime n	Search for files that have been modified in n days. –n means less than n days, n means exactly n days, while +n means more than n days.
-name file	Search for files with the specified name.The name may contain shell metacharacters,but they must usually be quoted.
-newer file	Search for files newer than the specified file.
-nogroup	Search for files within a group, meaning files whose group ID number doesn't appear in /etc/group.
-nouser	Search for files without an owner, meaning files whose owner ID number doesn't appear in /etc/passwd.
-perm perms	Search for files whose permissions exactly match the specified perms, which are the files access modes in octal. If perms is preceded by a -, all permission bits (sticky, set UID, set GID) are included.

-prune	Stops a search from proceeding beyond the specified point.
-size n	Search for files that are n blocks large, -n means less than n blocks, n means exactly n blocks, while +n means more than n blocks.
-type x	Search for files whose type is x: f for regular files, b for block special files, c for character special files, d for directory files, p for fifo (pipe) files, l for symbolic link files.
-user name	Search for files owned by the specified user.
(expr)	Parentheses for grouping.
!	Not
-o	OR
-a	AND (Not needed, because search criteria are ANDed left to right by default.)

Actions

-exec cmds	Execute the specified command, which may be many words. The command must be terminated by an escaped semicolon. In the command, the word {} will be replaced by the name of the matched file.
-ok cmds	Same as –exec, expect that it will operate interactively. For each found file, find will print the first word of the command, the name of the file, and then a question mark. You can type y to execute the command or anything else to skip the command.
-print	Print the name of each found file.

-tar : collect files.

. Pack and compress are great for squeezing space from large files, but they don’t do much for small files.

. Tar was originally designed as a program for working files on tap; its name stands for type archive. Tapes are usually treated as a single large file, and tar was designed to be adept at packing sets of files into a single file that could be stored on a type.

. Tar doesn’t remove the original files, since tar was designed to perform backups to type.

-file : deduce file types.

-du : disk usage.

-od : dump files.

11.3.1 Remote File System Organization

A remote file system differs from a remote disk system in that some of the semantics of the file and directory system are implemented within the server as well as the client machine. The server provides shared files, accessible from each of the client machines. As part of the file service, the server may provide concurrency control and file protection.

Many contemporary network file systems are implemented in the context of UNIX file systems, for example, AT&T Remote File System [24] and Sun's Network File Systems [25,28]. Thus, our discussion is based around many UNIX file system concepts. In particular, we assume that the file system is hierarchical --- tree-structured except for the case when a file is linked into more than one directory --- and that files are named according to path names in a tree or graph structure.

Files may be referenced on remote servers in two general ways: superpath names and remote mounting. Superpath names expand the normal UNIX absolute path names to include a level above root. Names in the high level are machine names.Two forms of superpath names are used:

pawnee:/usr/gjn/book/chap11

and

/../pawnee/usr/gjn/book/chap11

(The latter name is intended to mean "start at this machine's root, go up one level, and then choose the machine name and the absolute path name on the machine.) This technique causes the application software to distinguish between local files and remote files, since remote file names have the superlevel.

UNIX file systems include a mechanism for

incorporating subfile systems within the root file system. By performing a **mount** operation, a subtree can be appended to an existing hierarchical directory system.(see figure 16)

The **remote mount** approach allows the **mount** operation to extend across the network, interconnecting the logical tree in one machine into the tree structure on another machine. The **remote mount** command shown in figure 17 mounts directory b in machine A at mount point (directory) x in machine B. Thus /a/b/c in machine A refers to

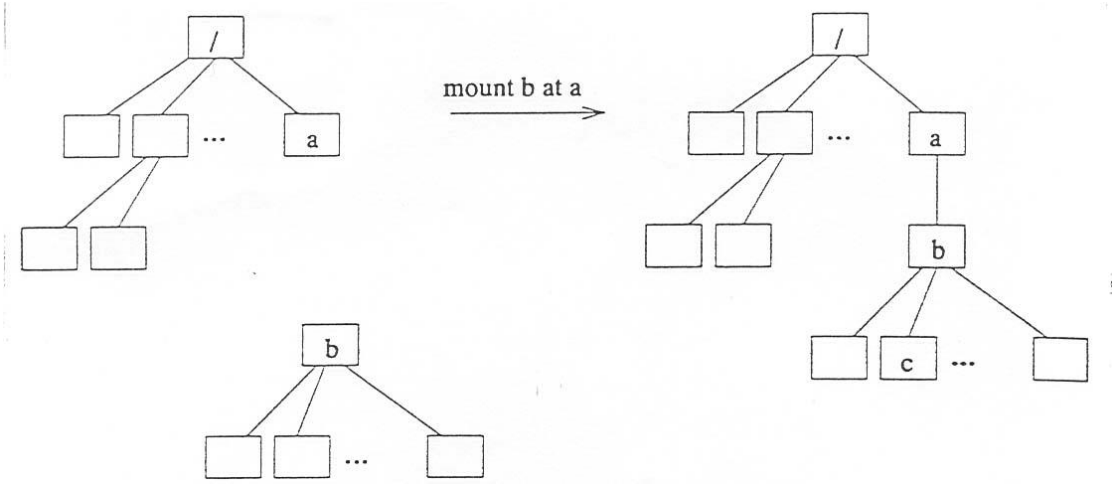


Figure 16 ■ The UNIX mount command

the same file as /x/b/c when referenced in machine B. Notice that this approach causes processes on each machine to see a different topology of the network file system, although local and remote files have the same form.

The issue of determining remote names also arises in remote file systems. For a file system to be remotely mounted or otherwise referenced from a client machine, it may be necessary for the name to be advertised in a global name space (see the discussion of naming in chapter 10). Remote file systems ordinarily operate in a network configuration that includes a name server that implements

the global name space.

Opening a file system on a remote file server may be a relatively complex and time-consuming operation. Suppose that a file name is specified as a network reference,

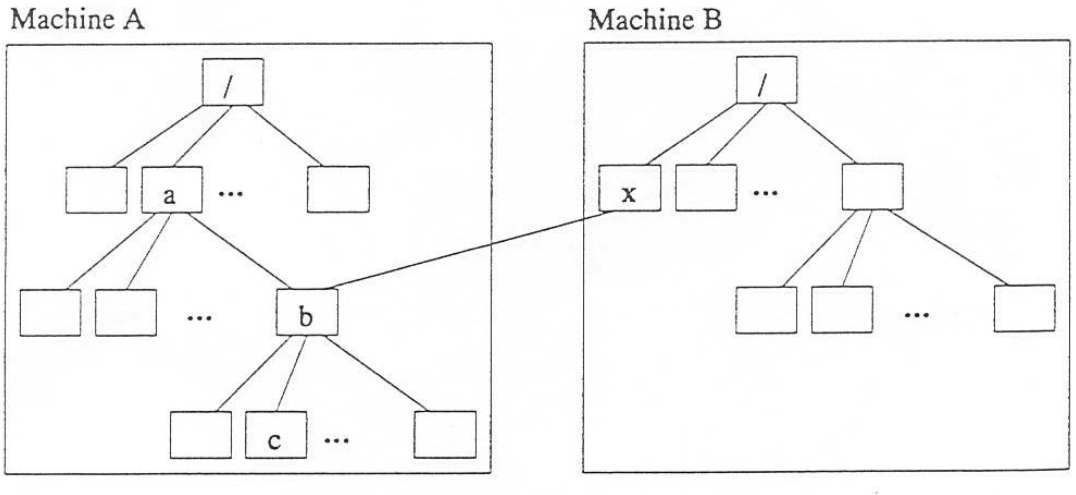


Figure 17 ■ A Shared Remote Disk Server

using remotely mounted file systems. In general, the open command causes a serial search of each directory in the path name (see chapter 9). At each level of the search, there is the possibility of encountering a remote mount point(see figure 18). Each subsequent directory search may cause the remainder of the path name to be passed to a new file server to complete the command. For example, if a process in machine B attempted to open /x/b/c/d/e, the open request would have to be processed in all three machines in order to locate the file descriptor for the leaf node file.

In UNIX file systems, a successful file open operation will result in the file descriptor being loaded into the client's memory (refer to Chapter 9). This is likely to also be required in most systems, since the current state of the file is saved in the descriptor. If two different client processes

open the same file, then each will have a cached version of the open file descriptor. Depending on the operating system policy, it may be acceptable to allow both systems to have the file open for writing at the same time (this is acceptable in UNIX). This situation illustrates the necessity for storage locks to control concurrent access to a file that has multiple open operations, at least one of which allows writing. For example, the AT&T RFS system provides a mechanism for locking, while the Sun NFS does not.

There are many strategies for implementing the file read and write operations once a file has been opened.The AT&T RFS approach and the Sun NFS approach illustrate contrasting strategies.

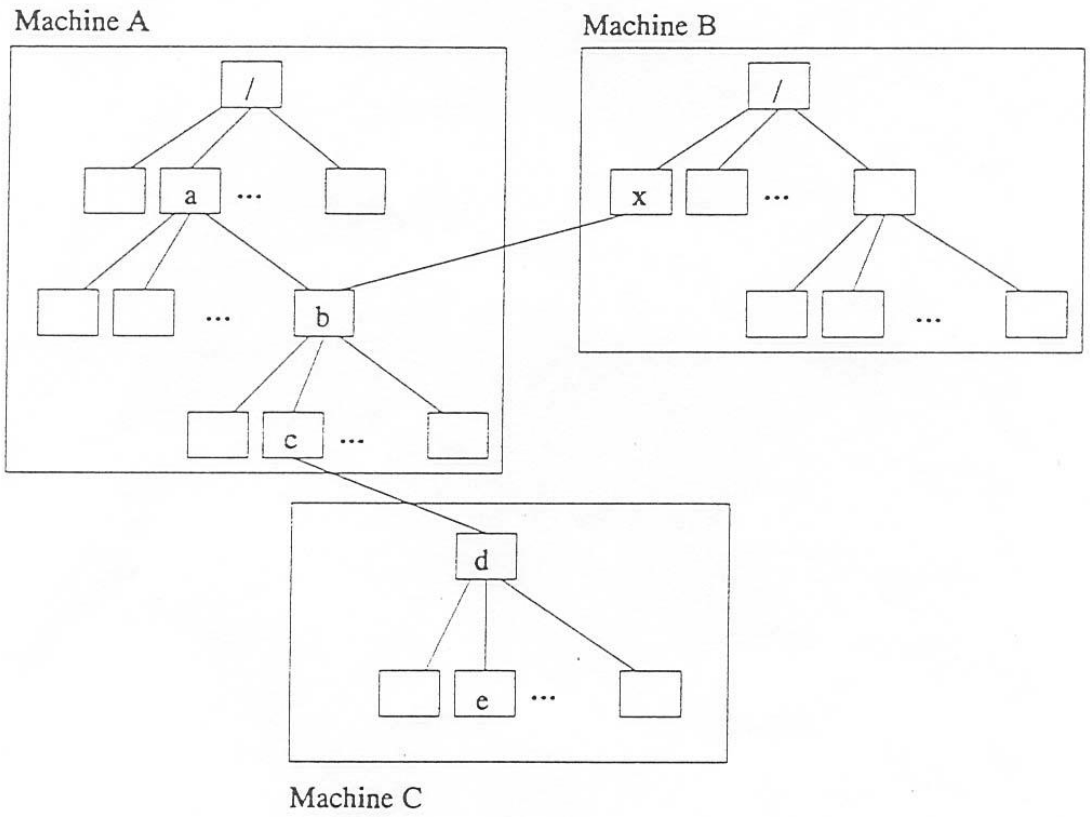


Figure 18 ■ Opening Remote Files

* More about UNIX

- CPU scheduling: priority (multi-level queue).

- Memory management: demand-paged virtual-memory system.
  - page replacement algorithm: LRU (global).

- Interprocess communication: pipe.

  - a pipe is essentially a queue of bytes between two processes.

  - the pipe is accessed by a file descriptor.

  - one process writes into the pipe and the other reads from the pipe.

  - in pipe, synchronization is needed because when a procedure tries to read from an empty pipe, it is blocked until data are available.

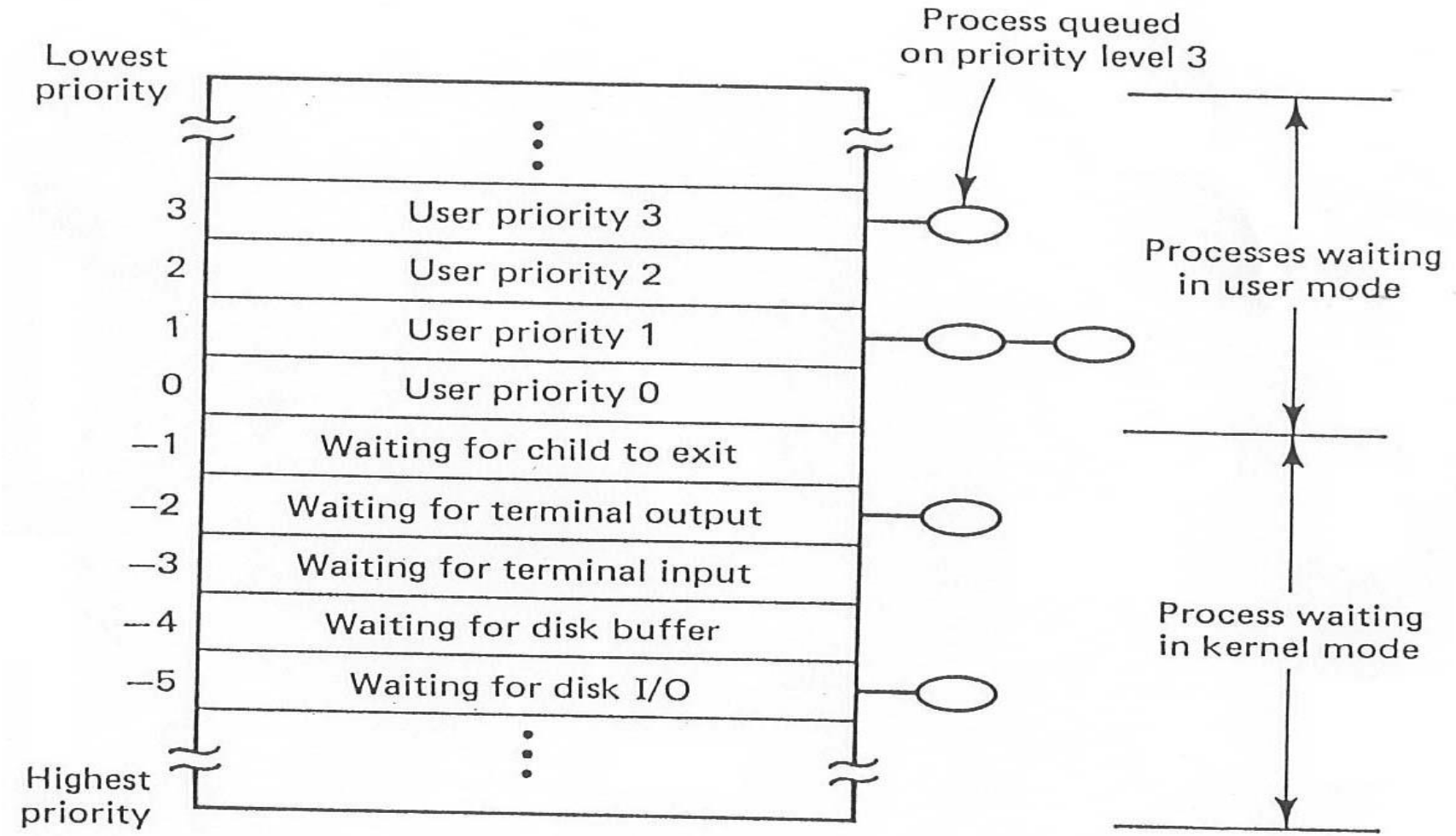


Figure 19 ■ The UNIX scheduler is based on a multilevel queue structure.