File Systems

Chapter 4

File Systems (1)

Essential requirements for long-term information storage:

1. It must be possible to store a very large amount of information.
2. Information must survive termination of process using it.
3. Multiple processes must be able to access information concurrently.

File Systems (2)

Think of a disk as a linear sequence of fixed-size blocks and supporting two operations:

1. Read block *k*.
2. Write block *k*

File Systems (3)

Questions that quickly arise:

1. How do you find information?
2. How do you keep one user from reading another user's data?
3. How do you know which blocks are free?

File Systems (4)

- Files are logical units of information created by processes.
- It provides a way to store information on the disk and read it later.
- A disk usually contains thousands, even millions of them, each one independent of the others.
- Process can read existing files and create new ones when needed.
- Information stored in files must be persistent, i.e., not be affected by process creation and termination.
- A file should disappear only when its owner explicitly removes it.

File Systems (5)

- Files are managed by the operating system.
- How files are structured, named, accessed, used, protected, implemented, and managed are important topics in operating system design.
- The part of the operating system dealing with the files is known as the file system.

File Naming (1)

- When a process creates a file, it gives the file a name.
- When the process terminates, the file continues to exist and can be accessed by other processes using its name.
- The exact rules for file naming vary from system to system.
- All current operating systems allow strings of one to eight letters as legal file names.
- Often digits and special characters are permitted.
- Some systems support as long as 255 characters.
- UNIX distinguishes between upper case and lower case, where as MS-DOS does not.
- Windows 95 and Windows 98 both used MS-DOS file system, called FAT-16. Windows 98 introduced some extension to FAT-16, leading to FAT-32.

File Naming (2)

- In UNIX, file extensions are just conventions and are not enforced by the operating system.
- However, Windows is aware of the extensions and assigns meaning to them. Users can register extensions with the operating system and specify for each one which program owns that extension. For example, double clicking on file.docx starts Microsoft Word with file.docx as the initial file to edit.

File Naming (3)

Extension	Meaning
.bak	Backup file
.c	C source program
.gif	Compuserve Graphical Interchange Format image
.hlp	Help file
.html	World Wide Web HyperText Markup Language document
.jpg	Still picture encoded with the JPEG standard
.mp3	Music encoded in MPEG layer 3 audio format
.mpg	Movie encoded with the MPEG standard
.o	Object file (compiler output, not yet linked)
.pdf	Portable Document Format file
.ps	PostScript file
.tex	Input for the TEX formatting program
.txt	General text file
.zip	Compressed archive

Figure 4-1. Some typical file extensions.

File Structure (1)

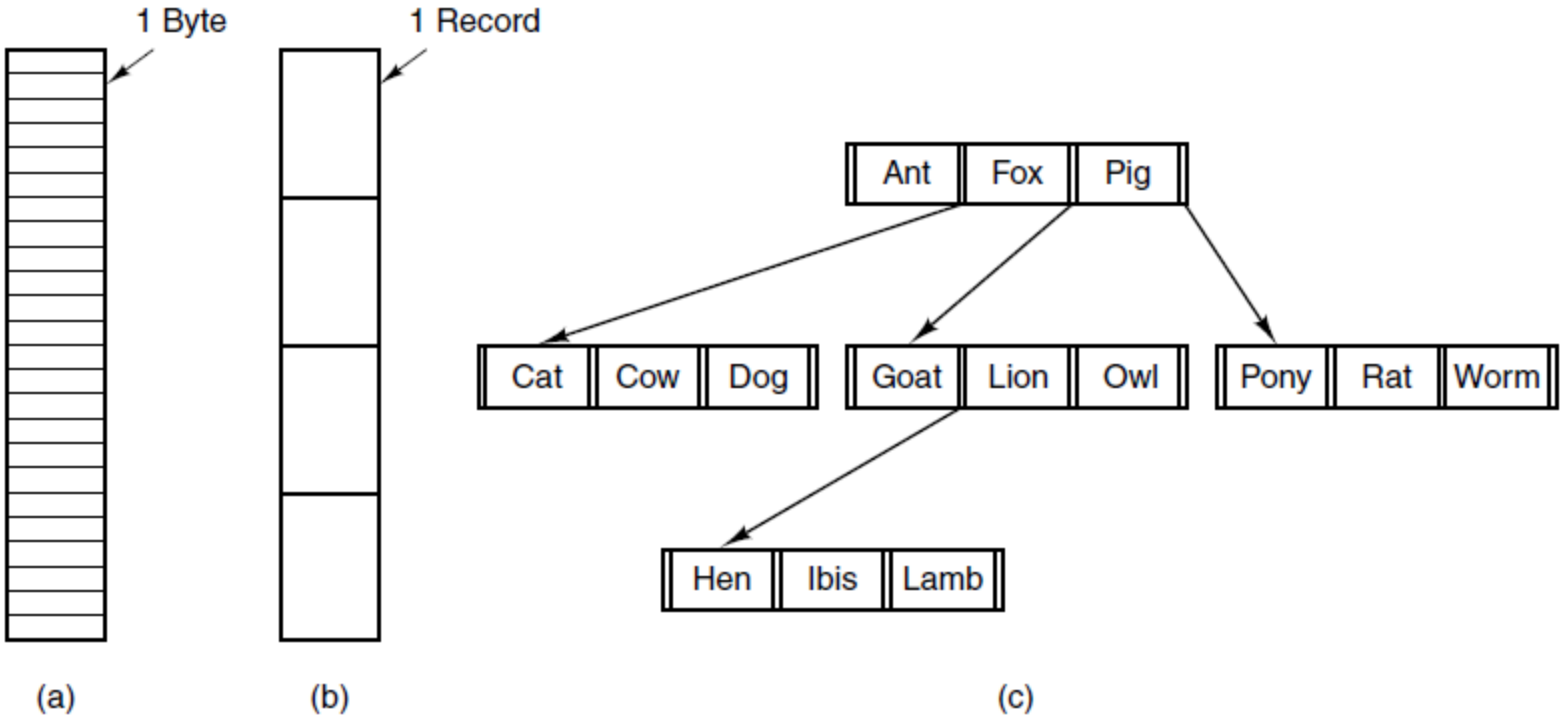


Figure 4-2. Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.

File Structure (2)

- Byte sequence: An unstructured sequence of bytes. OS does not care what is inside it. Any meaning must be imposed by the user-level programs. It provides the maximum flexibility. All versions of UNIX (LINUX and OS X) and windows use this file model.
- Record sequence: A file is a sequence of fixed-length records, each with some internal structures. No current general-purpose system uses this model as its primary file system any more (once it was popular model for mainframe during 80-column punched card!).
- Tree: A file consists of a tree of records of varying lengths, each containing a key field in a fixed position in a record. The tree is sorted on the key field, allowing quick searching for a key. Used in large mainframe computer for commercial data processing.

File Types (1)

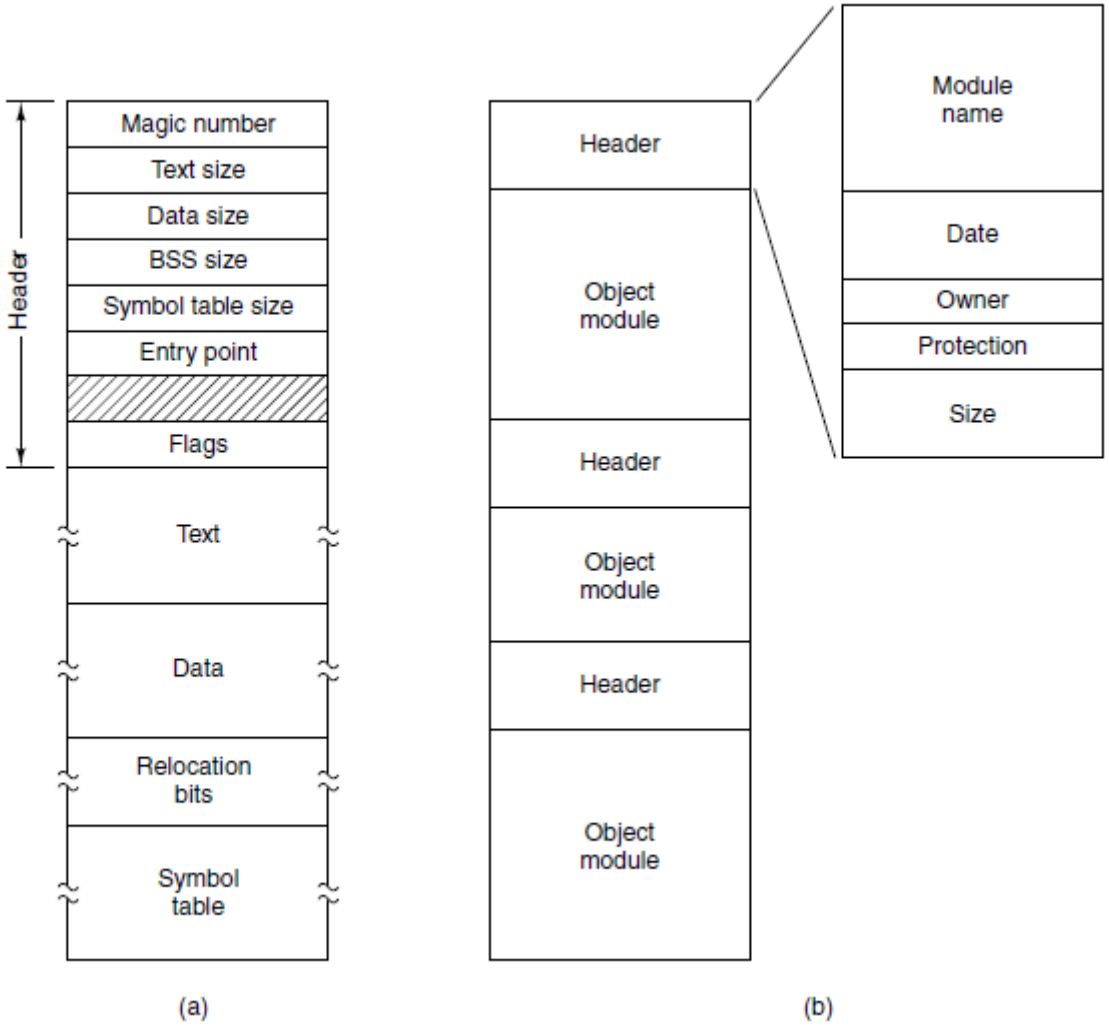


Figure 4-3. (a) An executable file. (b) An archive

File Types (2)

- Several types of file:
- Regular files contain user information. Regular files are either ASCII or binary files.
- Directories are system files for maintaining the structure of the file system.
- Character special files are related to input/output and used in model serial I/O devices (e.g. terminals, printers, networks).
- Block special files are used to model disks.

File Types (3)

- First Example: simple executable binary file from early version of UNIX. OS will execute a file only if it has the proper format. It has five sections: header, text, data, relocation bits, and symbol table.
- The header starts with a magic number, identifying the file as an executable file to prevent accidental execution of a file not in this format. Then the sizes of the various pieces of the file, the address at which execution starts, and some flag bits (to indicate hidden files or backed up recently)
- Following the header are the text and data of the program. These are loaded into memory and relocated using the relocation bits.
- The symbol table is used for debugging.
- Second example: binary file in an archive from UNIX. It consists of a collection of library procedures (modules) compiled but not linked. Header provides its name, creation date, owner, protection code, and size.

File Access

- Early OS only provided sequential file access. Sequential access were convenient when the storage medium was magnetic tape, not magnetic disk. Read operation gives the position in the file to start reading.
- Random access files can be read in order by keys rather than by the position. Application: Database systems. A special operation, seek, is provided to reposition the file pointer to a specific places in the file. Once the position is found, the file can be read sequentially from that point. Used in UNIX and Windows.

File Attributes

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the 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 files; 1 for system file
Archive flag	0 for 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 file 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 the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file was last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Figure 4-4. Some possible file attributes.

File Operations

1. Create
2. Delete
3. Open
4. Close
5. Read
6. Write
7. Append
8. Seek
9. Get attributes
10. Set attributes
11. Rename

Example Program Using File System Calls (1)

```
/* File copy program. Error checking and reporting is minimal. */

#include <sys/types.h>                  /* include necessary header files */
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]);       /* ANSI prototype */

#define BUF_SIZE 4096                   /* use a buffer size of 4096 bytes */
#define OUTPUT_MODE 0700                /* protection bits for output file */

int main(int argc, char *argv[])
{
    int in_fd, out_fd, rd_count, wt_count;
    char buffer[BUF_SIZE];

    if (argc != 3) exit(1);             /* syntax error if argc is not 3 */

    /* Open the input file and create the output file */
```

Figure 4-5. A simple program to copy a file.

Example Program Using File System Calls (2)

```
if (argc != 3) exit(1);                          /* syntax error if argc is not 3 */

/* Open the input file and create the output file */
in_fd = open(argv[1], O_RDONLY);                 /* open the source file */
if (in_fd < 0) exit(2);                          /* if it cannot be opened, exit */
out_fd = creat(argv[2], OUTPUT_MODE);            /* create the destination file */
if (out_fd < 0) exit(3);                         /* if it cannot be created, exit */

/* Copy loop */
while (TRUE) {
        rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
if (rd_count <= 0) break;                        /* if end of file or error, exit loop */
        wt_count = write(out_fd, buffer, rd_count); /* write data */
```

Figure 4-5. A simple program to copy a file.

Example Program Using File System Calls (3)

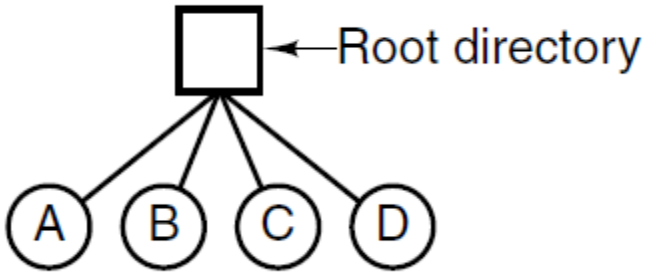
```
        /* Copy loop */
        while (TRUE) {
                rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
        if (rd_count <= 0) break;                         /* if end of file or error, exit loop */
                wt_count = write(out_fd, buffer, rd_count); /* write data */
                if (wt_count <= 0) exit(4);               /* wt_count <= 0 is an error */
        }

        /* Close the files */
        close(in_fd);
        close(out_fd);
        if (rd_count == 0)                                /* no error on last read */
                exit(0);
        else
                exit(5);                                  /* error on last read */
}
```

Figure 4-5. A simple program to copy a file.

Directories

- To keep track of files, file systems normally have directories or folders, which are themselves files.
- In following slides, we will discuss about directories, their organization, their properties, and operations that can be performed on them.

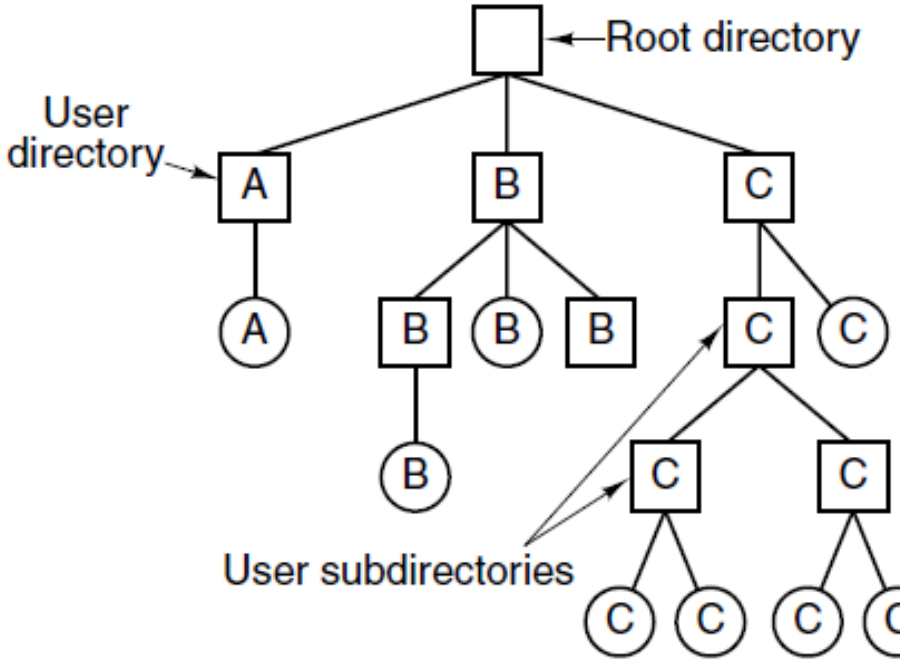
Single-Level Directory Systems



One directory (root directory)containing all the files. Still used in embedded devices such as digital cameras and some portable music players.

Figure 4-6. A single-level directory system containing four files.

Hierarchical Directory Systems



Idea: group related files together.
We need a hierarchy (i.e. a tree of directories).
Absolute path: starts from root
Relative path starts from working directory
Dot (.) refers to current directory
Dot dot (..) refers to parent directory

Figure 4-7. A hierarchical directory system.

Path Names

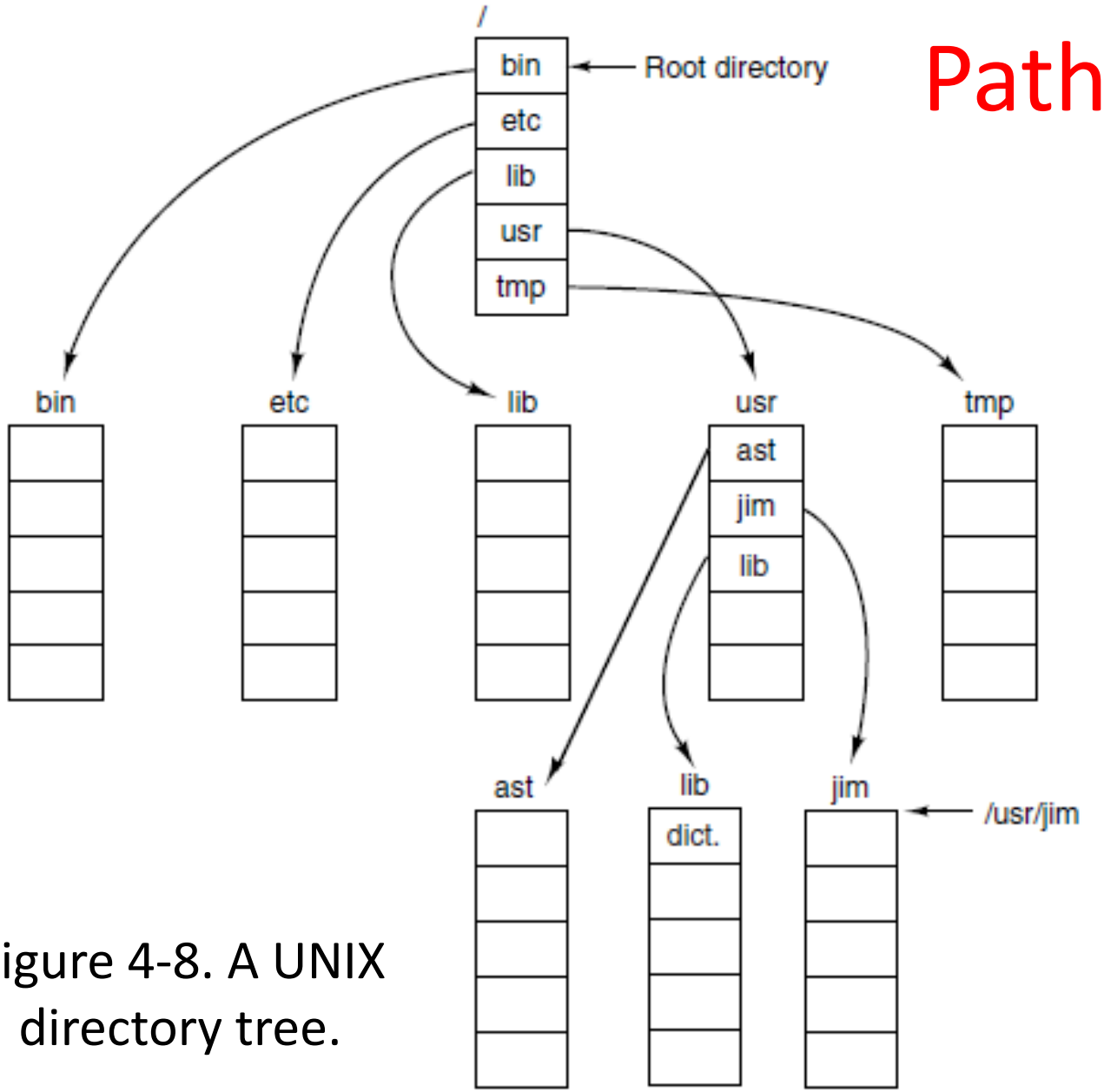


Figure 4-8. A UNIX directory tree.

Directory Operations

1. Create
2. Delete
3. Opendir
4. Closedir
5. Readdir
6. Rename
7. Link
8. Unlink

File System Layout

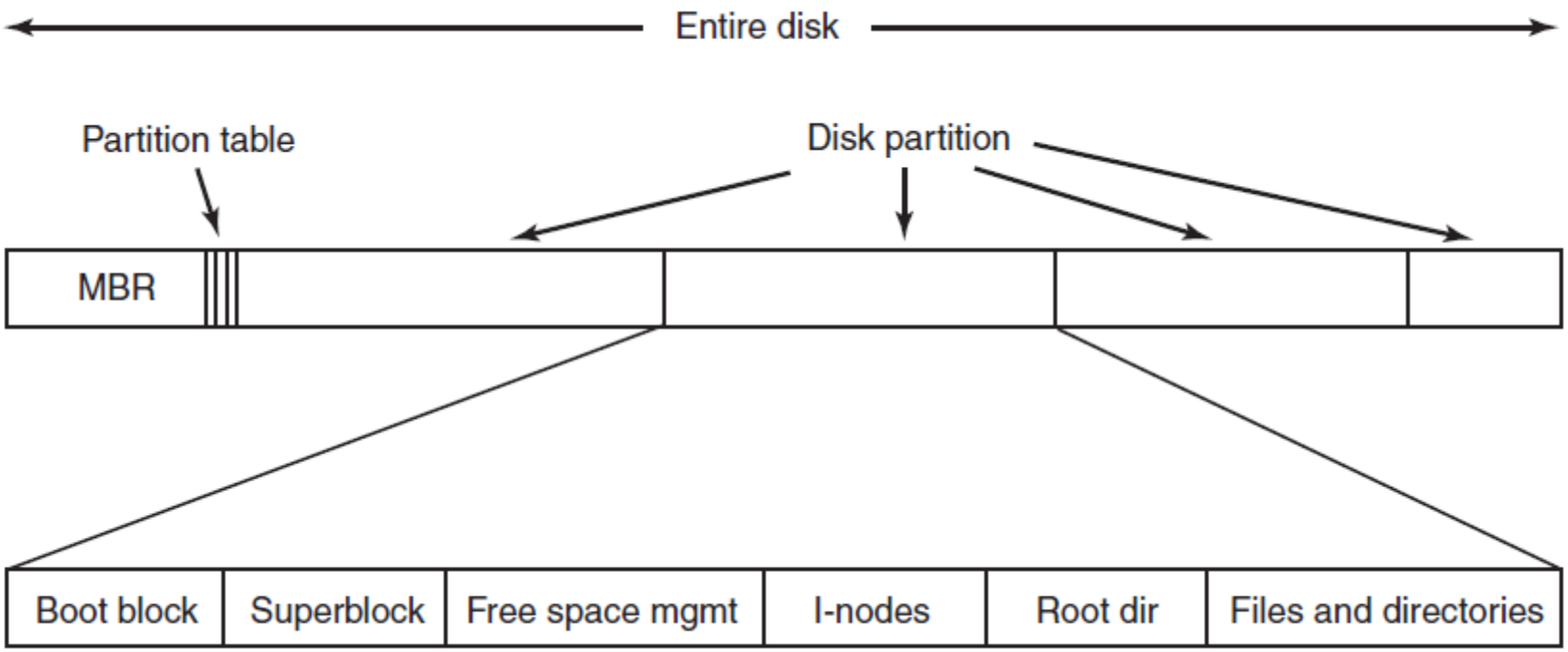


Figure 4-9. A possible file system layout.

Implementing Files
Contiguous Layout

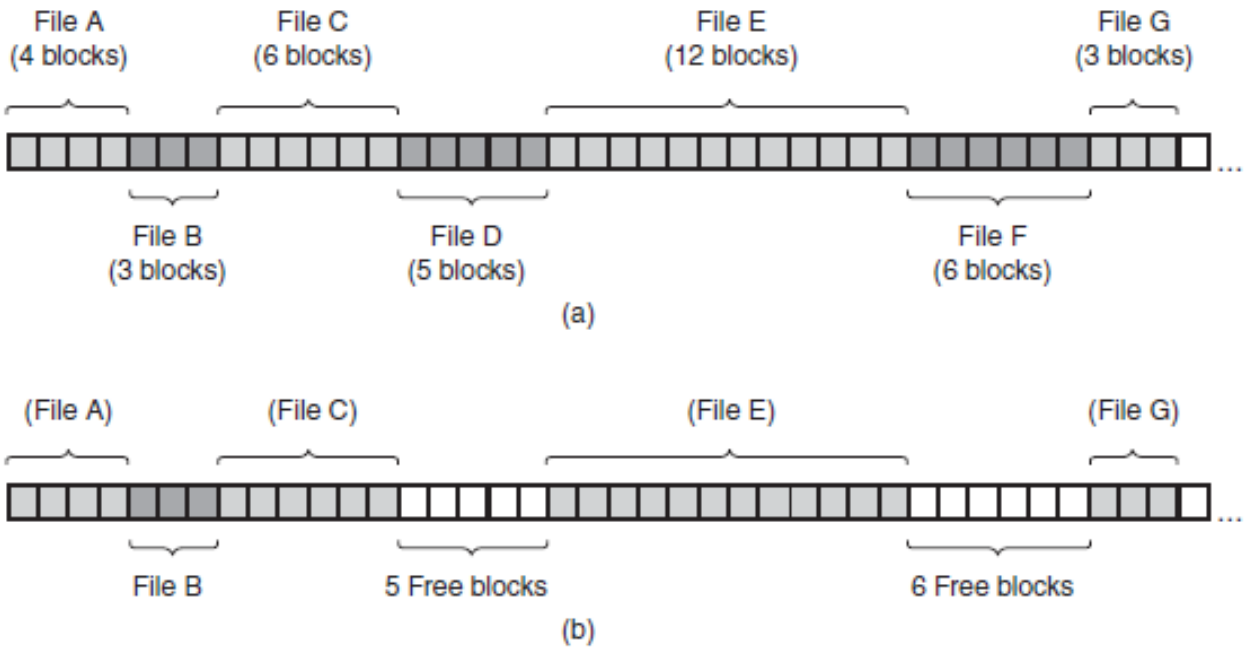


Figure 4-10. (a) Contiguous allocation of disk space for seven files. (b) The state of the disk after files *D* and *F* have been removed.

Implementing Files
Linked List Allocation

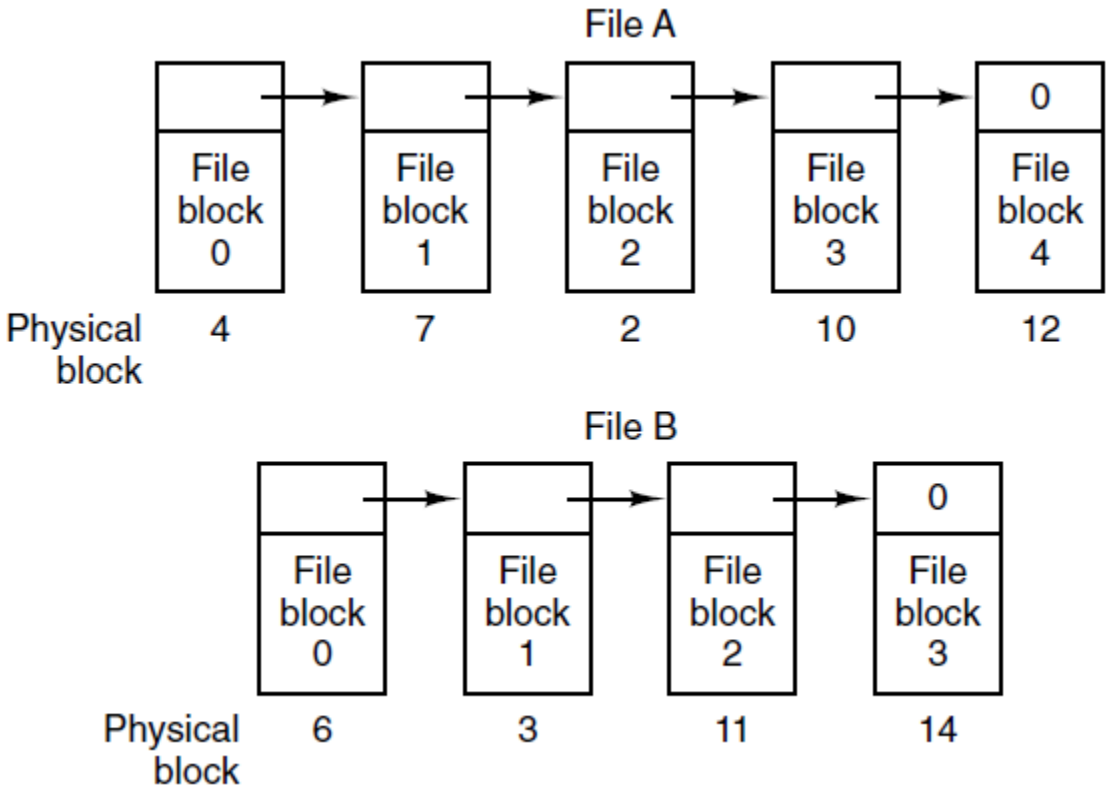


Figure 4-11. Storing a file as a linked list of disk blocks.

Implementing Files
Linked List – Table in Memory

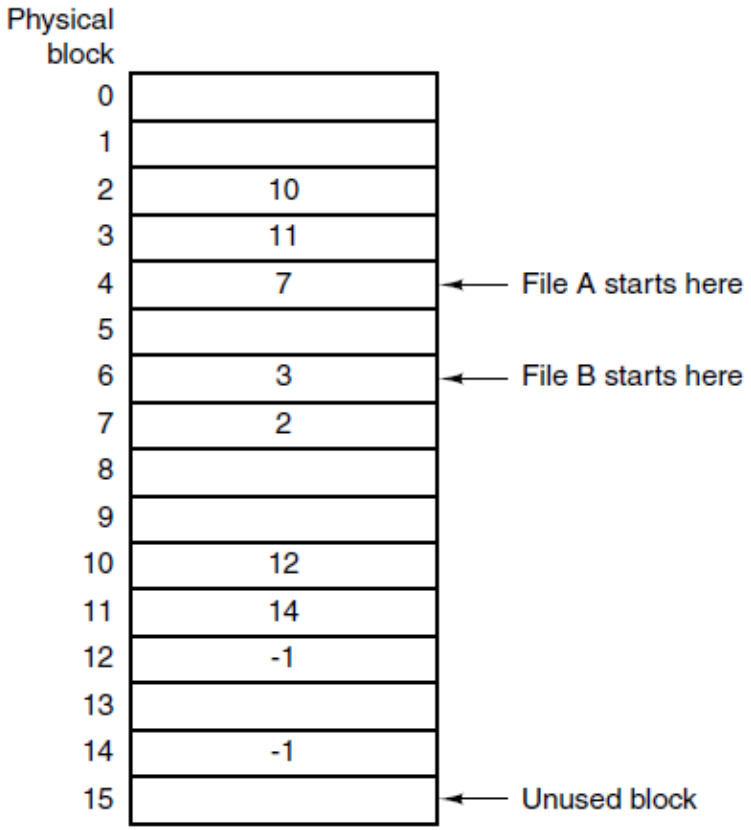


Figure 4-12. Linked list allocation using a file allocation table in main memory.

Implementing Files I-nodes

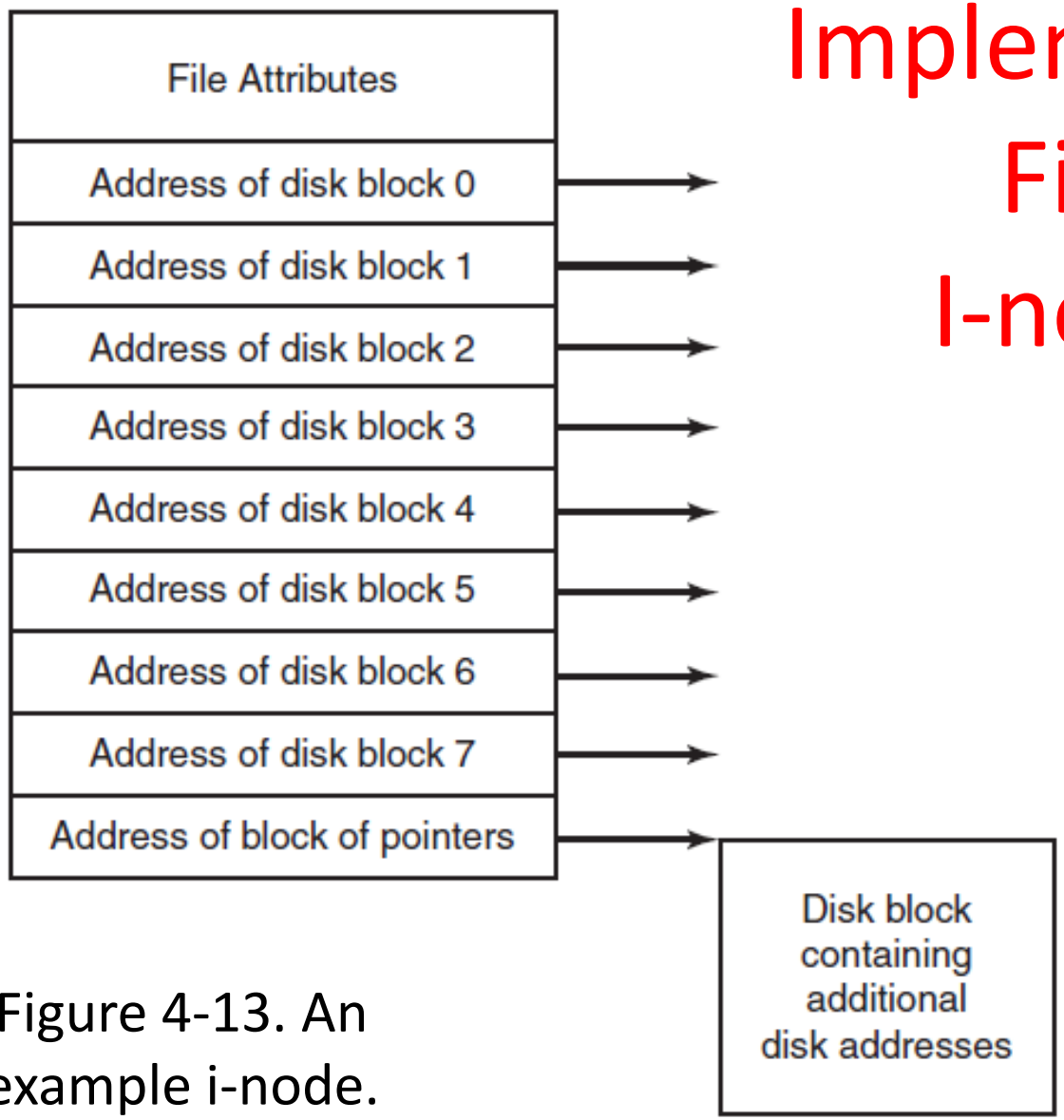


Figure 4-13. An example i-node.

Implementing Directories (1)

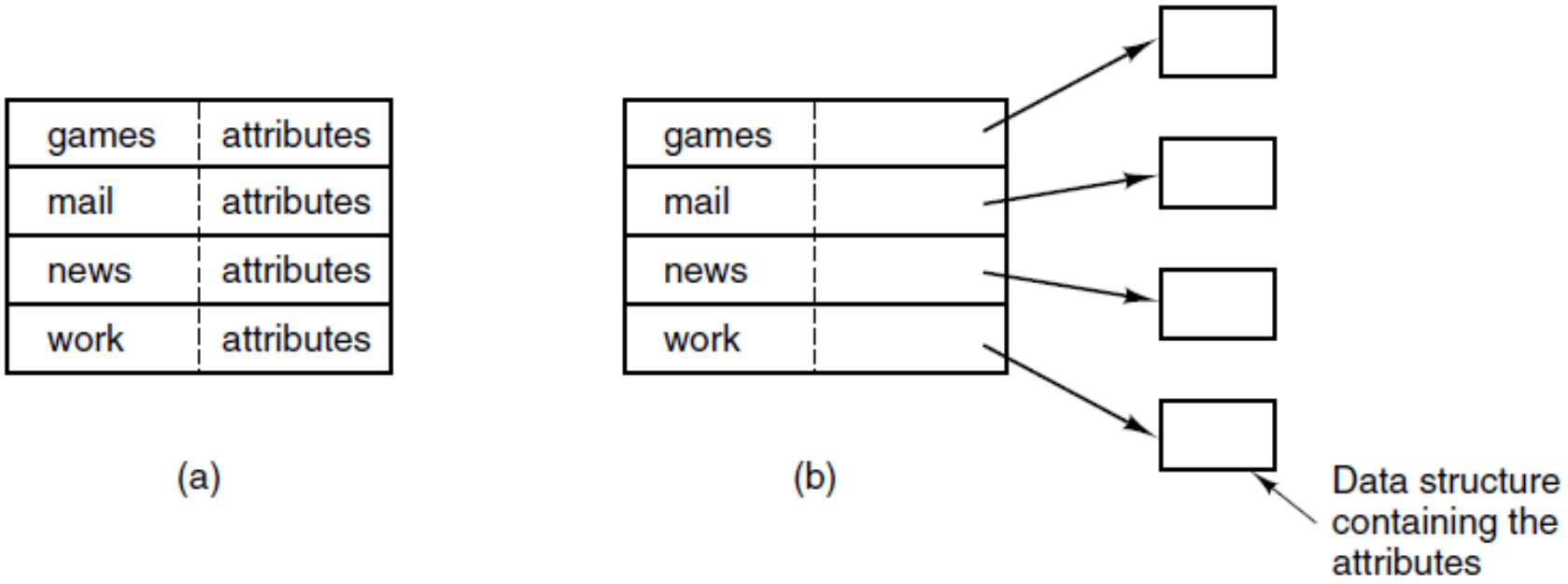


Figure 4-14. (a) A simple directory containing fixed-size entries with the disk addresses and attributes in the directory entry. (b) A directory in which each entry just refers to an i-node.

Implementing Directories (2)

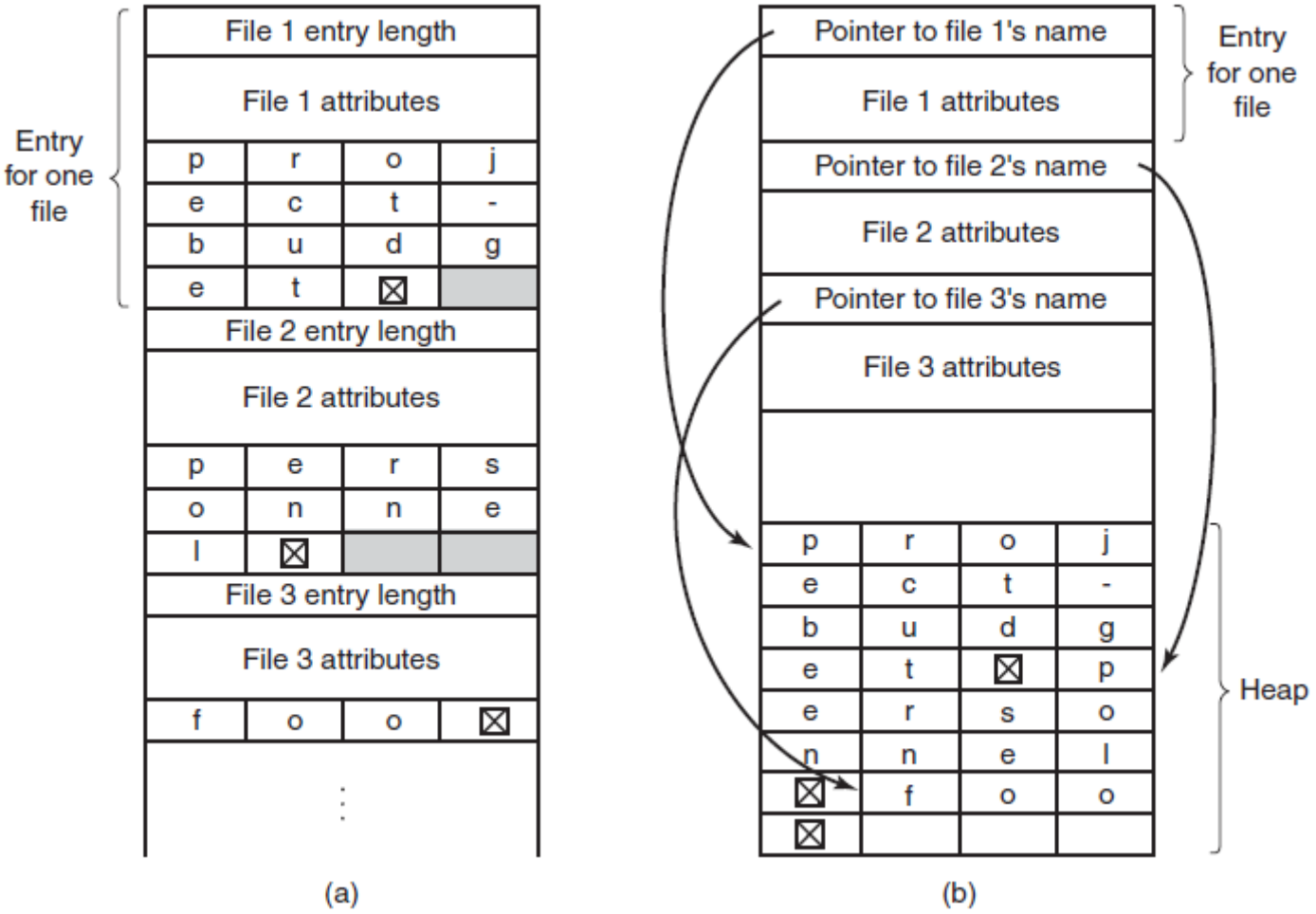


Figure 4-15. Two ways of handling long file names in a directory. (a) In-line. (b) In a heap.

Shared Files (1)

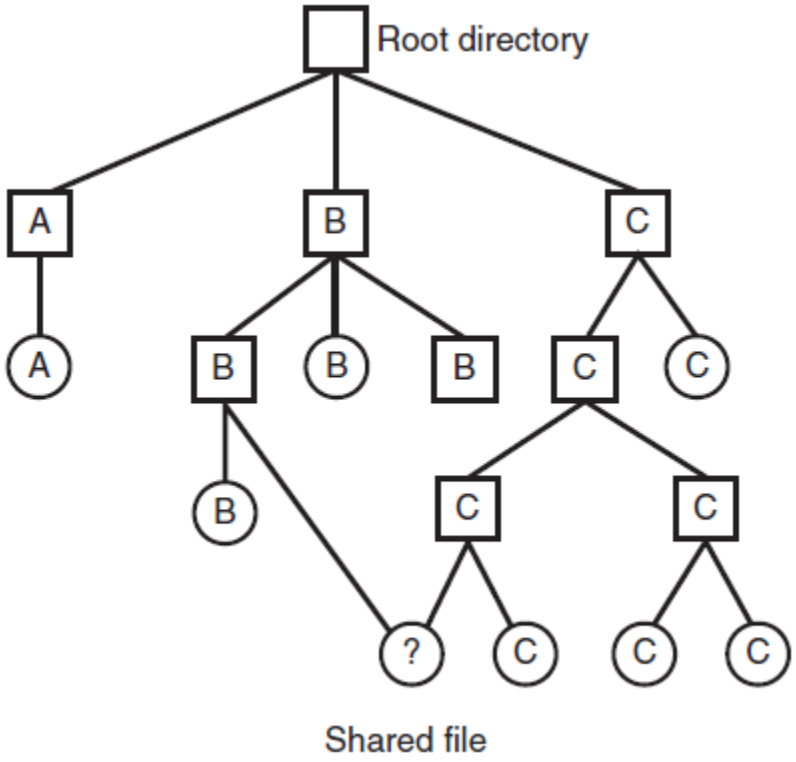


Figure 4-16. File system containing a shared file.

Shared Files (2)

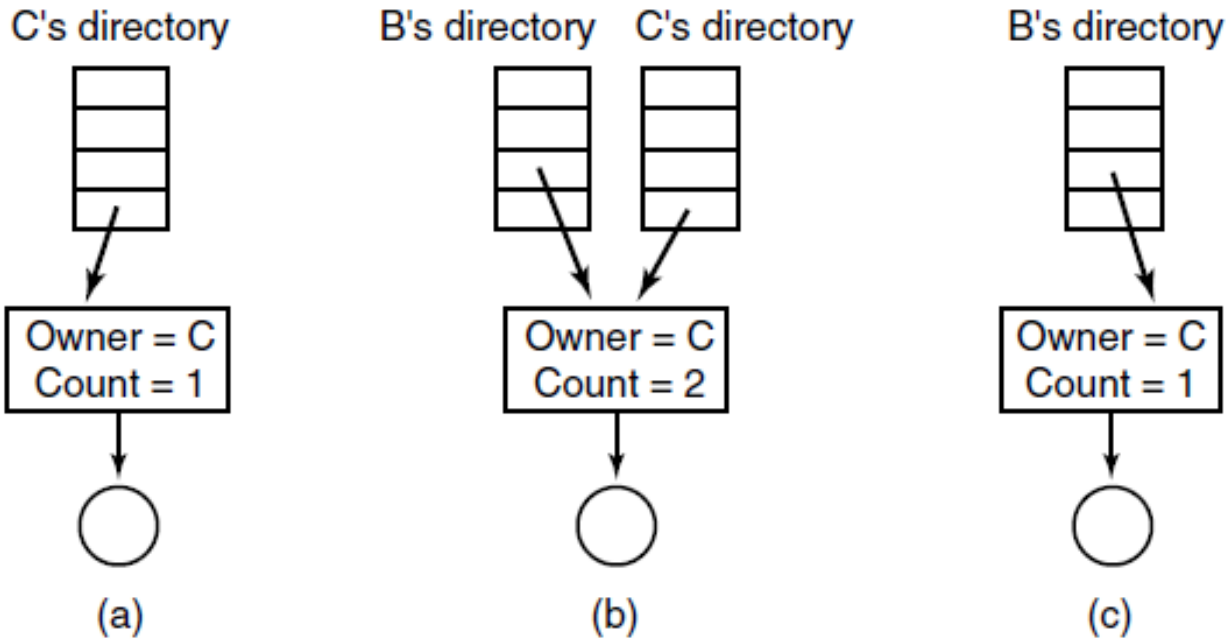


Figure 4-17. (a) Situation prior to linking. (b) After the link is created. (c) After the original owner removes the file.

Journaling File Systems

Steps to remove a file in UNIX:

1. Remove file from its directory.
2. Release i-node to the pool of free i-nodes.
3. Return all disk blocks to pool of free disk blocks.

Virtual File Systems (1)

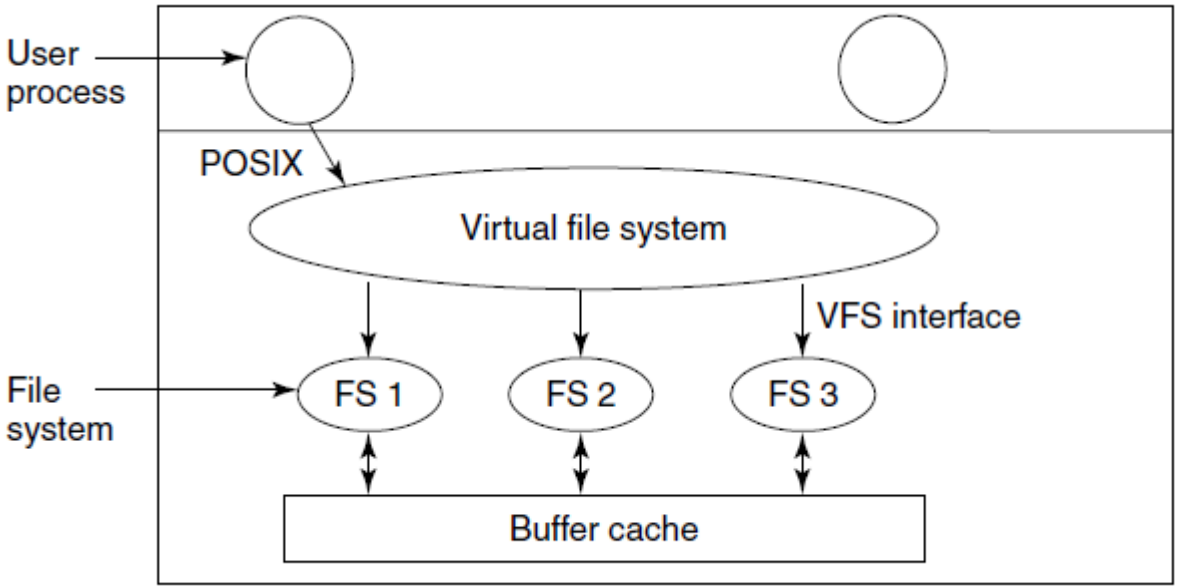


Figure 4-18. Position of the virtual file system.

Virtual File System

- Modern operating systems must concurrently support multiple types of file systems.
- Most OS, including UNIX, use object-oriented techniques to simplify, organize, and modularize the implementations.
- The use of these methods allows very dissimilar file-system types to be implemented within the same structure, including network file system, such as NFS.
- Users can access files that are contained within multiple file systems on the local disk or even on file systems available across the network.
- Data structures and procedures are used to isolate the basic system-call functionality from the implementation details.

Virtual File System

- The file-system implementation consists of three major layers:
- 1) file-system interface layer, based on the open(), read(), write(), and close() calls and on the file descriptors
- 2) Virtual file system (VFS) layer serves two functions. a) It separates file-system-generic operations from their implementation by defining a clean VFS interface. Several implementations for the VFS interface may coexist on the same machine, allowing transparent access to different types of file systems mounted locally. b)The VFS provides a mechanism for uniquely representing a file throughout a network. The VFS is based on a file-representation structure, called a vnode, that contains a numerical designator for a network-wide unique file. (UNIX inodes are unique within only a single file system). This network-wide uniqueness is required for support of network file systems. The kernel maintains one vnode structure for each active node (file or directory).

Virtual File System

- The VFS activates file-system-specific operations to handle local requests according to their file-system types and even calls the NFS protocol procedures for remote requests. File handles are constructed from the relevant vnodes and are passed as arguments to theses procedures.
- The layer implementing the file-system type or the remote-file-system protocol is the third layer of the architecture.
- Four main object types defined by the LINUX VFS are:
  - The inode object, which represents an individual file
  - The file object, which represents an open file
  - The suberblock object, which represents an entire file system
  - The dentry object, which represents an individual directory entry.

Virtual File System

- For each of these four object types, the VFS defines a set of operations that must be implemented. Every object of one of theses types contains a pointer to a function table. The function table lists the addresses of the actual functions that implemented the defined operations for that particular object.
- For example some of the operations for the file object include: int open(…), ssize_t read(…), ssize_t write(…), int mmap(…)

Virtual File Systems (2)

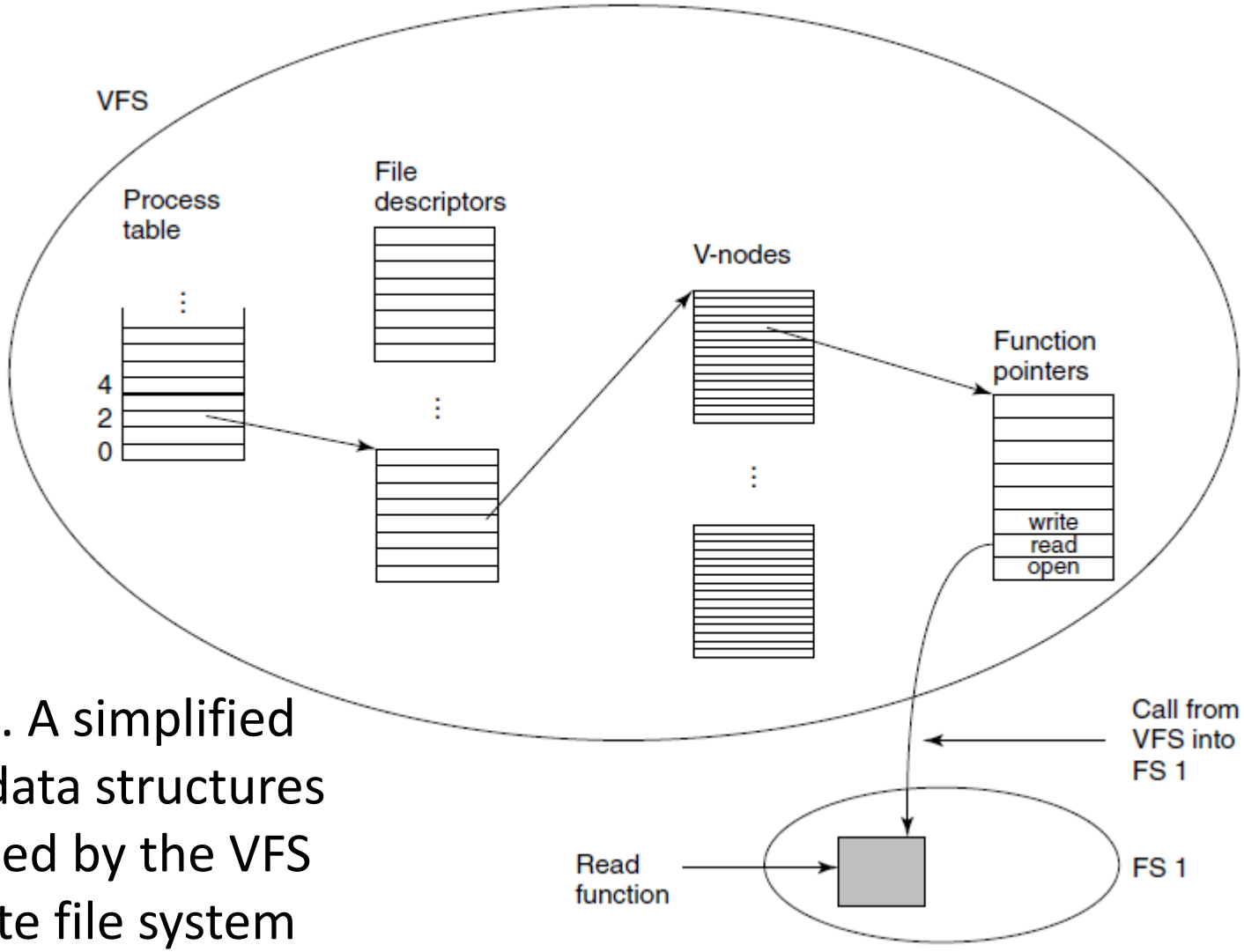


Figure 4-19. A simplified view of the data structures and code used by the VFS and concrete file system to do a *read*.

Virtual File System

- Most UNIX systems have used the concept of VFS to integrate multiple file systems into an orderly structure.
- The key idea is to abstract out the part of the file system that is common to all file systems that put that code n a separate layer that calls the underlying concrete file systems to actually manage the data.
- All system calls relating to files are directed to the virtual file system for initial processing. These calls, coming from the user processes are standard POSIX calls, such as open, read, write, lseek, and so on.
- VFS has an upper interface to the user processes and is well-known POSIX interface. VFS has also a lower interface to the concrete file systems and is known as VFS interface.

Virtual File System

- As long as the concrete file system supplies the functions the VFS requires, VFS does not care or know where the data are stored or what the underlying file system is like.
- An example of such a function is one that reads a specific block from a disk, puts it in a file system's buffer cache, and returns a pointer to it.
- VFS supports local file system as well as remote file system using the NFS (Network File System) protocol.

Disk Space Management (1)

Length	VU 1984	VU 2005	Web
1	1.79	1.38	6.67
2	1.88	1.53	7.67
4	2.01	1.65	8.33
8	2.31	1.80	11.30
16	3.32	2.15	11.46
32	5.13	3.15	12.33
64	8.71	4.98	26.10
128	14.73	8.03	28.49
256	23.09	13.29	32.10
512	34.44	20.62	39.94
1 KB	48.05	30.91	47.82
2 KB	60.87	46.09	59.44
4 KB	75.31	59.13	70.64
8 KB	84.97	69.96	79.69

Length	VU 1984	VU 2005	Web
16 KB	92.53	78.92	86.79
32 KB	97.21	85.87	91.65
64 KB	99.18	90.84	94.80
128 KB	99.84	93.73	96.93
256 KB	99.96	96.12	98.48
512 KB	100.00	97.73	98.99
1 MB	100.00	98.87	99.62
2 MB	100.00	99.44	99.80
4 MB	100.00	99.71	99.87
8 MB	100.00	99.86	99.94
16 MB	100.00	99.94	99.97
32 MB	100.00	99.97	99.99
64 MB	100.00	99.99	99.99
128 MB	100.00	99.99	100.00

Figure 4-20. Percentage of files smaller than a given size (in bytes).

Disk Space Management (2)

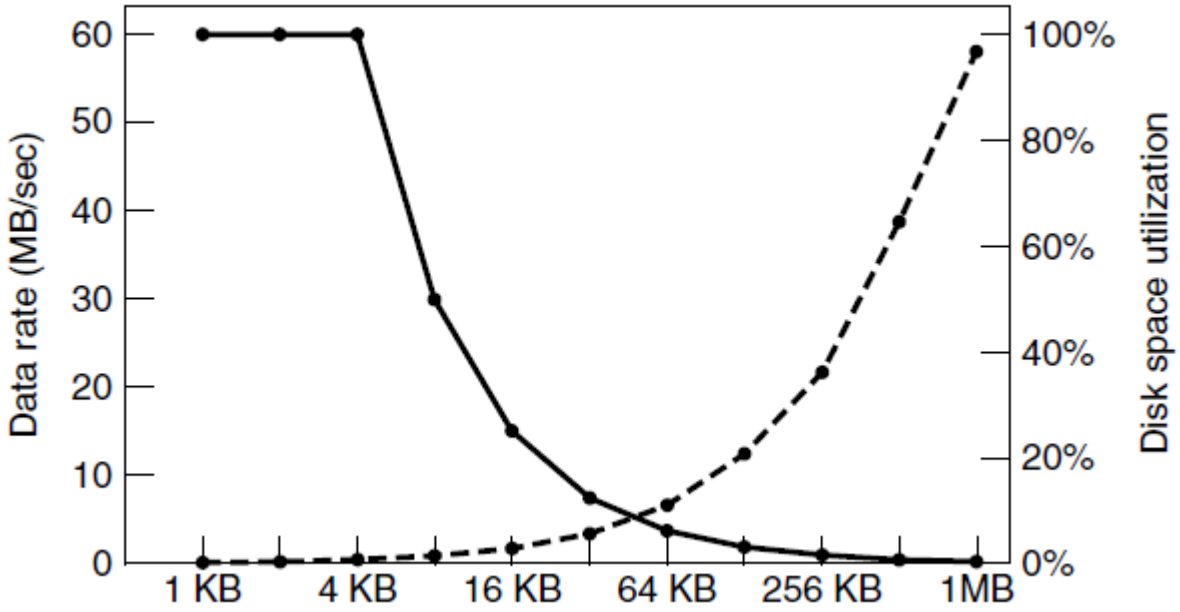


Figure 4-21. The dashed curve (left-hand scale) gives the data rate of a disk. The solid curve (right-hand scale) gives the disk space efficiency. All files are 4 KB.

Keeping Track of Free Blocks (1)

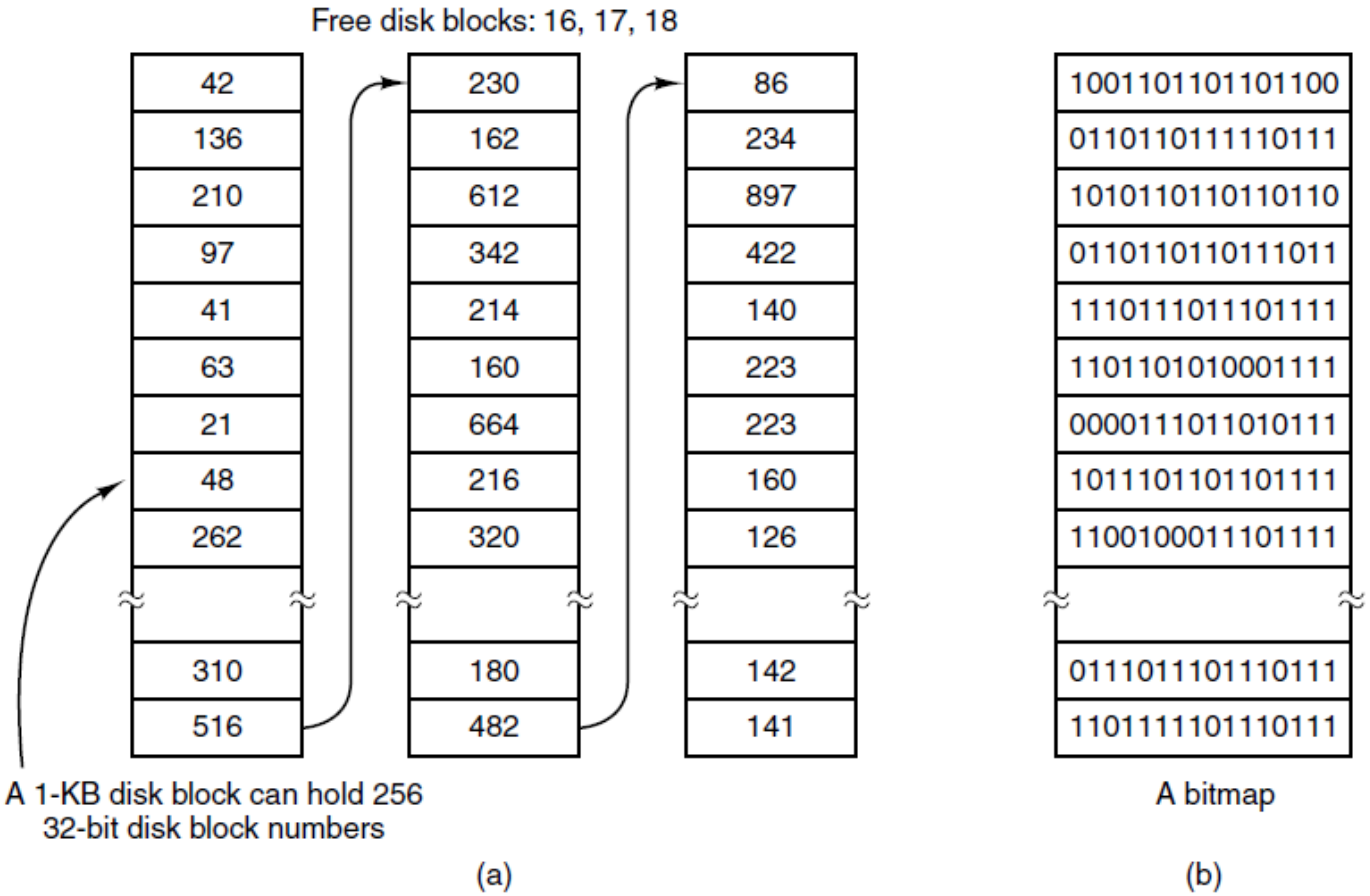


Figure 4-22. (a) Storing the free list on a linked list. (b) A bitmap.

Keeping Track of Free Blocks (2)

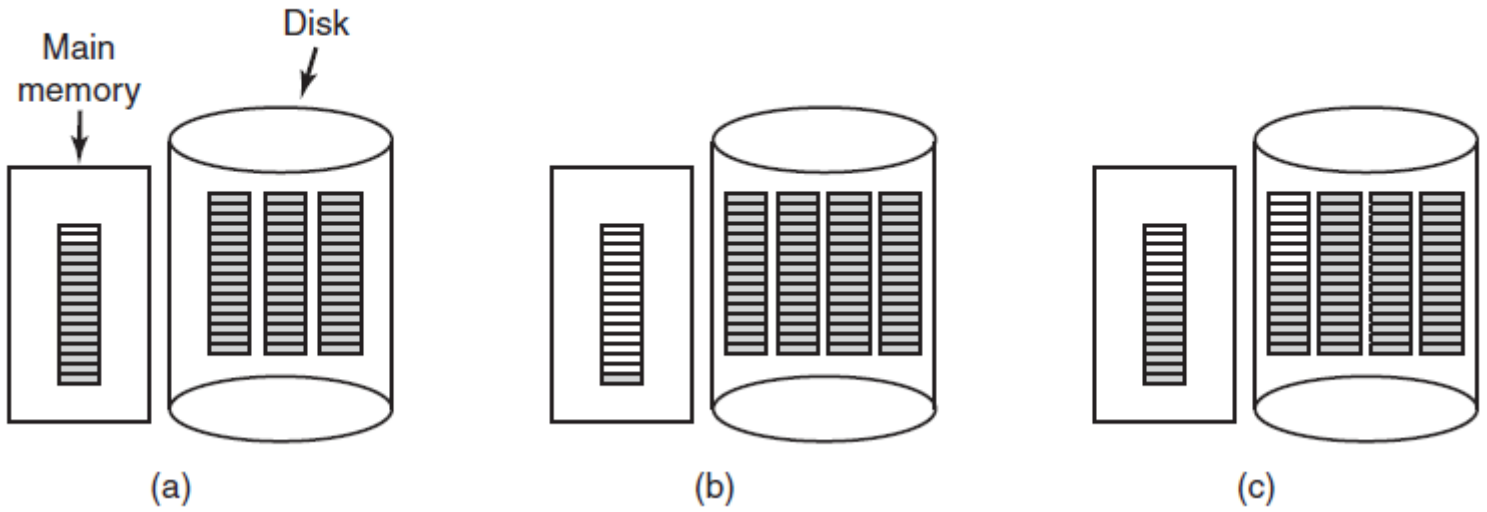


Figure 4-23. (a) An almost-full block of pointers to free disk blocks in memory and three blocks of pointers on disk. (b) Result of freeing a three-block file. (c) An alternative strategy for handling the three free blocks. The shaded entries represent pointers to free disk blocks.

Disk Quotas

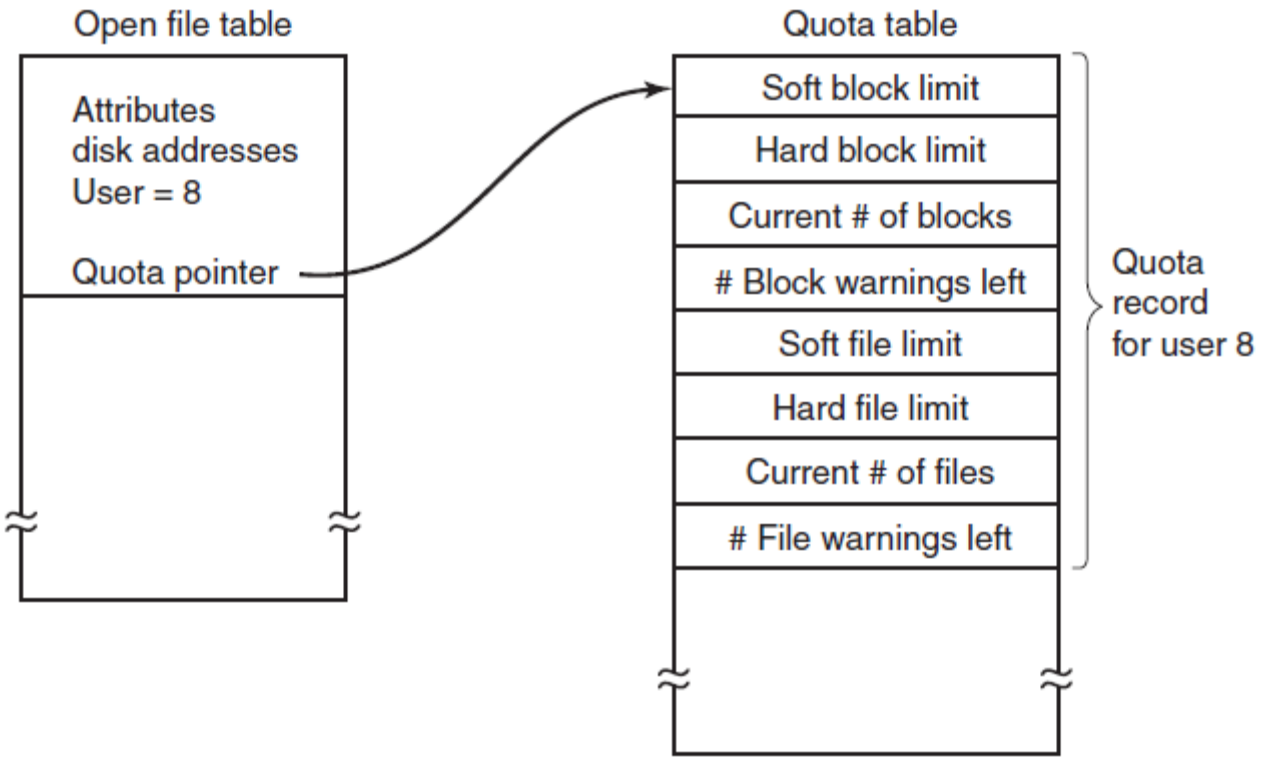


Figure 4-24. Quotas are kept track of on a per-user basis in a quota table.

File System Backups (1)

Backups to tape are generally made to handle one of two potential problems:

1. Recover from disaster.
2. Recover from stupidity.

File System Backups (2)

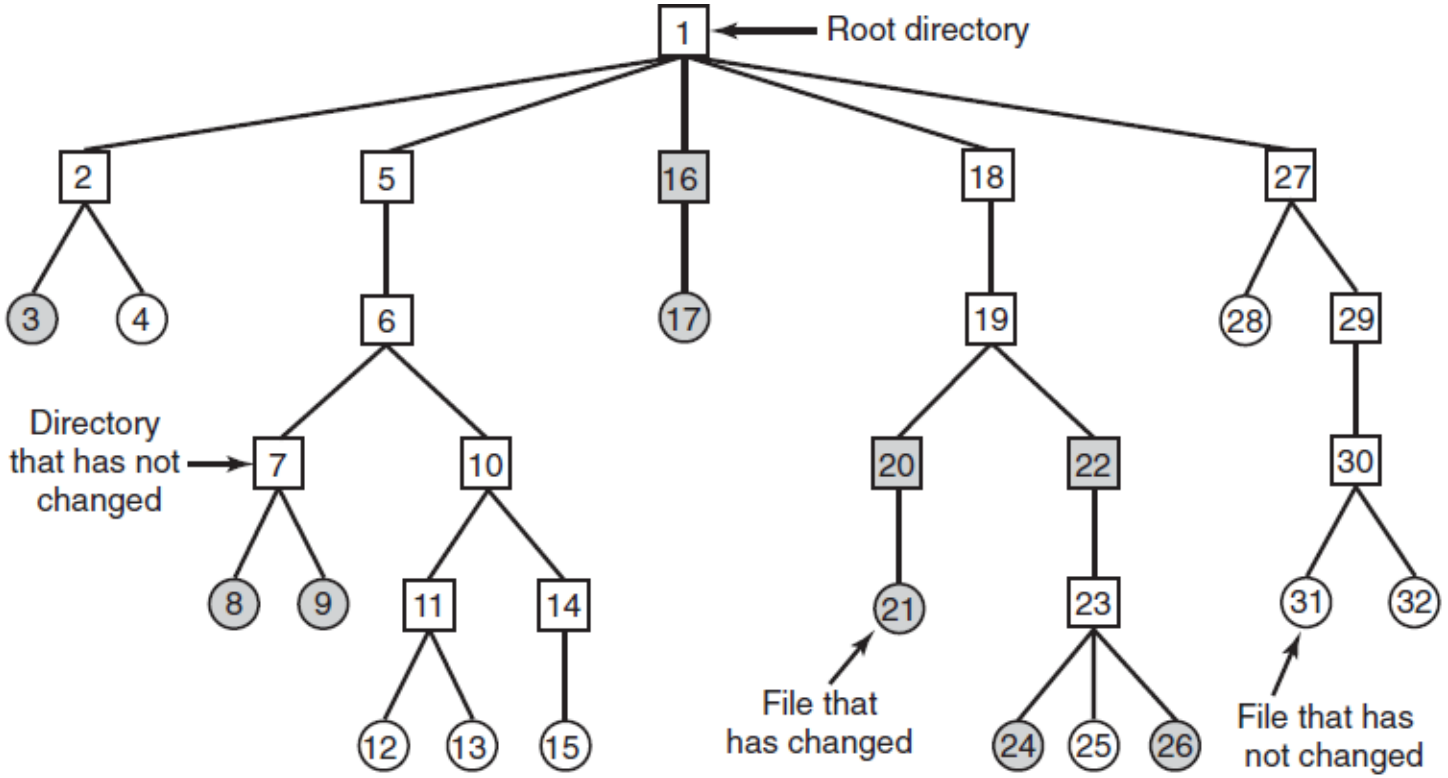


Figure 4-25. A file system to be dumped. The squares are directories and the circles are files. The shaded items have been modified since the last dump. Each directory and file is labeled by its i-node number.

File System Backups (3)

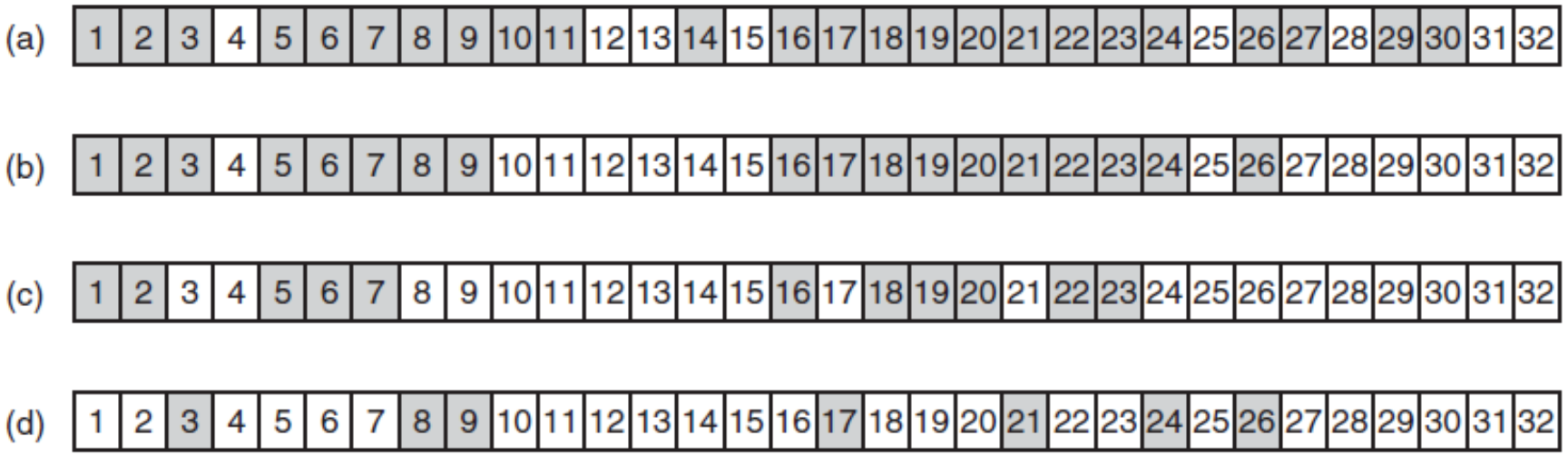


Figure 4-26. Bitmaps used by the logical dumping algorithm.

File System Consistency

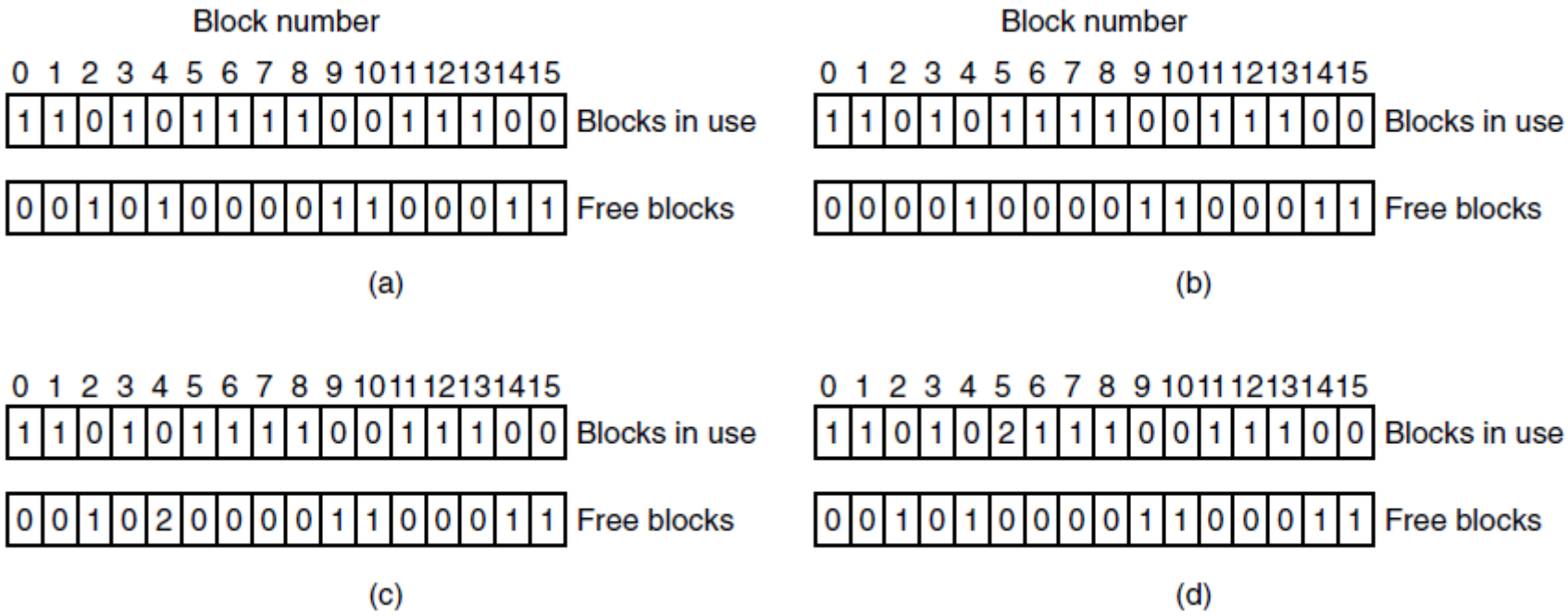


Figure 4-27. File system states. (a) Consistent. (b) Missing block. (c) Duplicate block in free list. (d) Duplicate data block.

File System Performance (1)

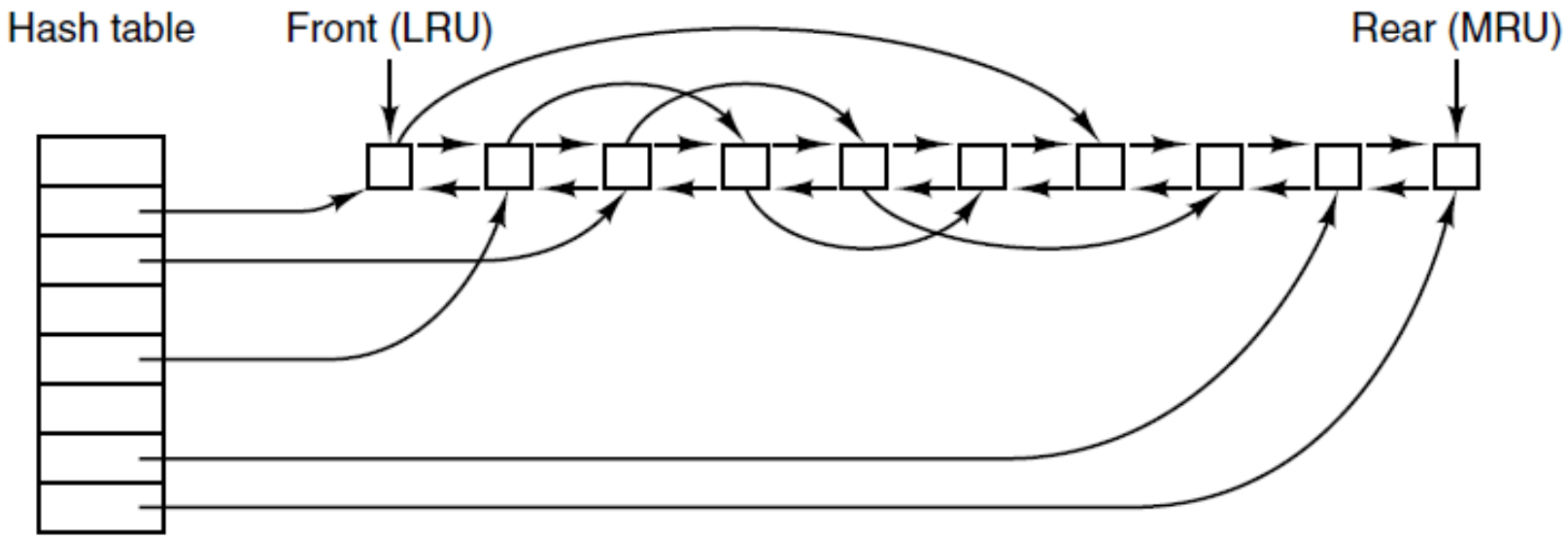


Figure 4-28. The buffer cache data structures.

File System Performance (2)

- Some blocks rarely referenced two times within a short interval.
- Leads to a modified LRU scheme, taking two factors into account:
  1. Is the block likely to be needed again soon?
  2. Is the block essential to the consistency of the file system?

Reducing Disk Arm Motion

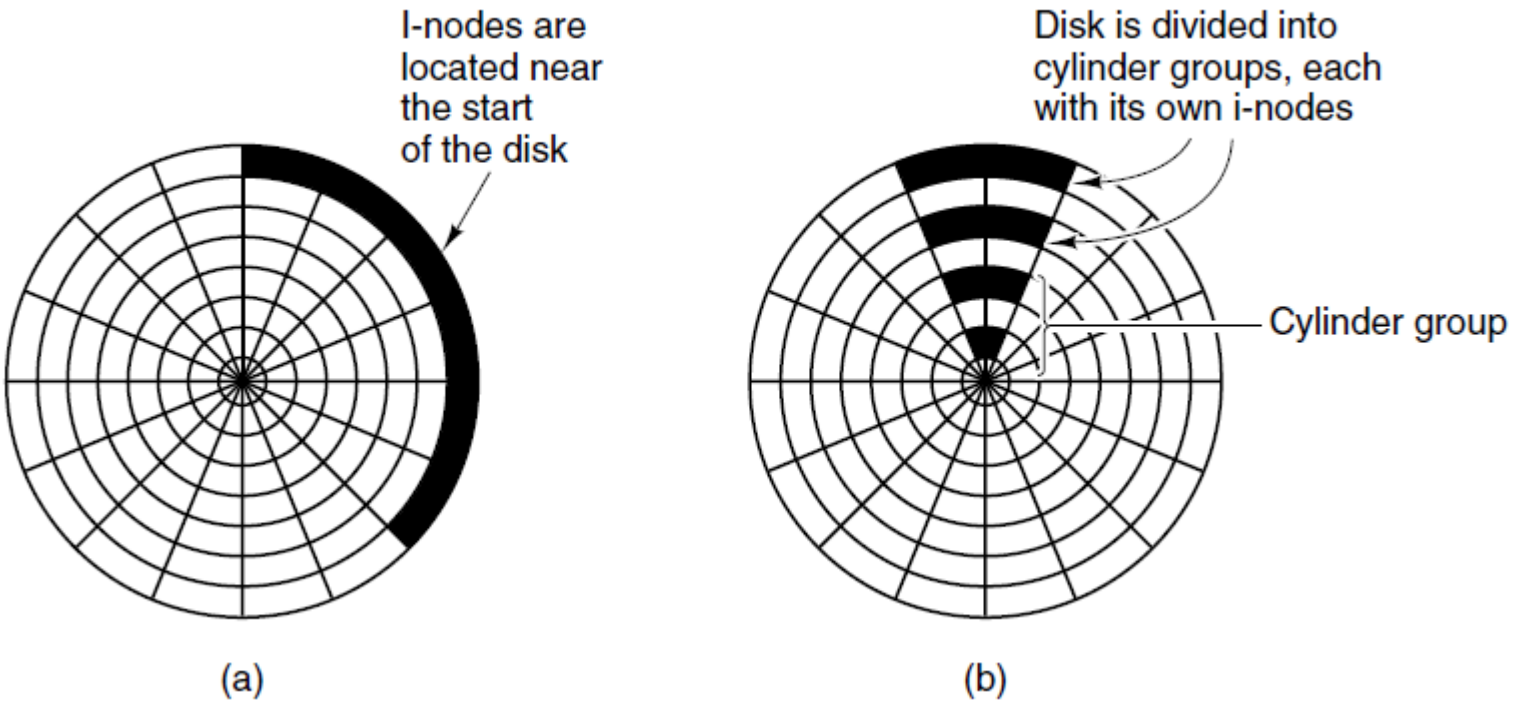


Figure 4-29. (a) I-nodes placed at the start of the disk. (b) Disk divided into cylinder groups, each with its own blocks and i-nodes.

The MS-DOS File System (1)

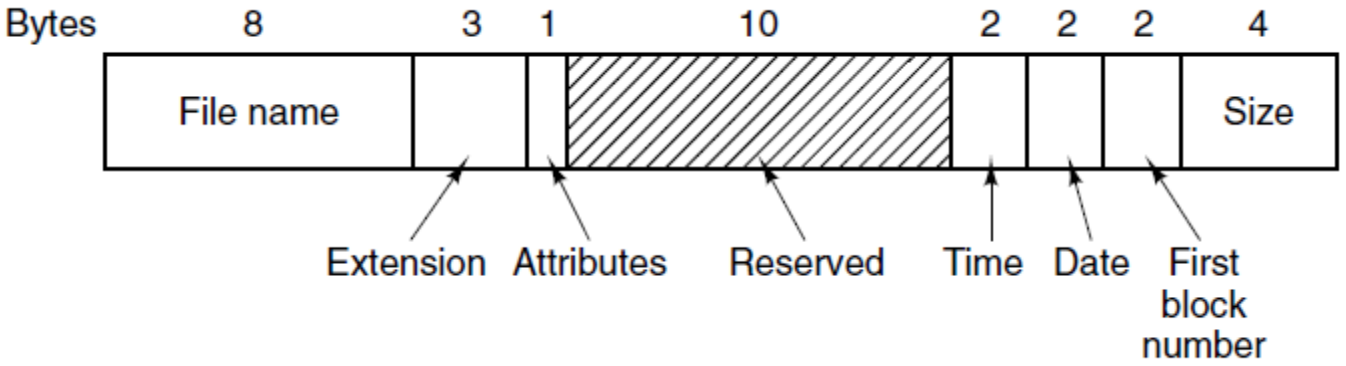


Figure 4-30. The MS-DOS directory entry.

The MS-DOS File System (2)

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

Figure 4-31. Maximum partition size for different block sizes. The empty boxes represent forbidden combinations.

The UNIX V7 File System (1)

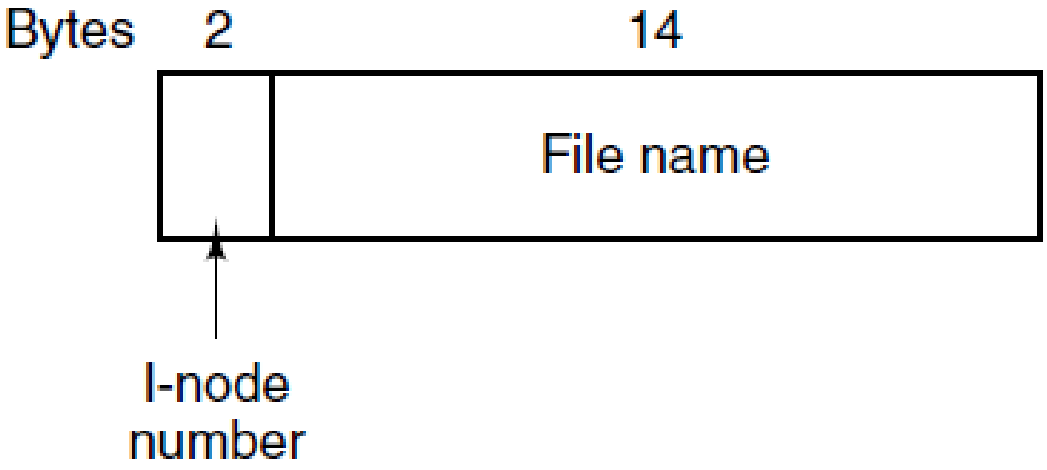


Figure 4-32. A UNIX V7 directory entry.

The UNIX V7 File System (2)

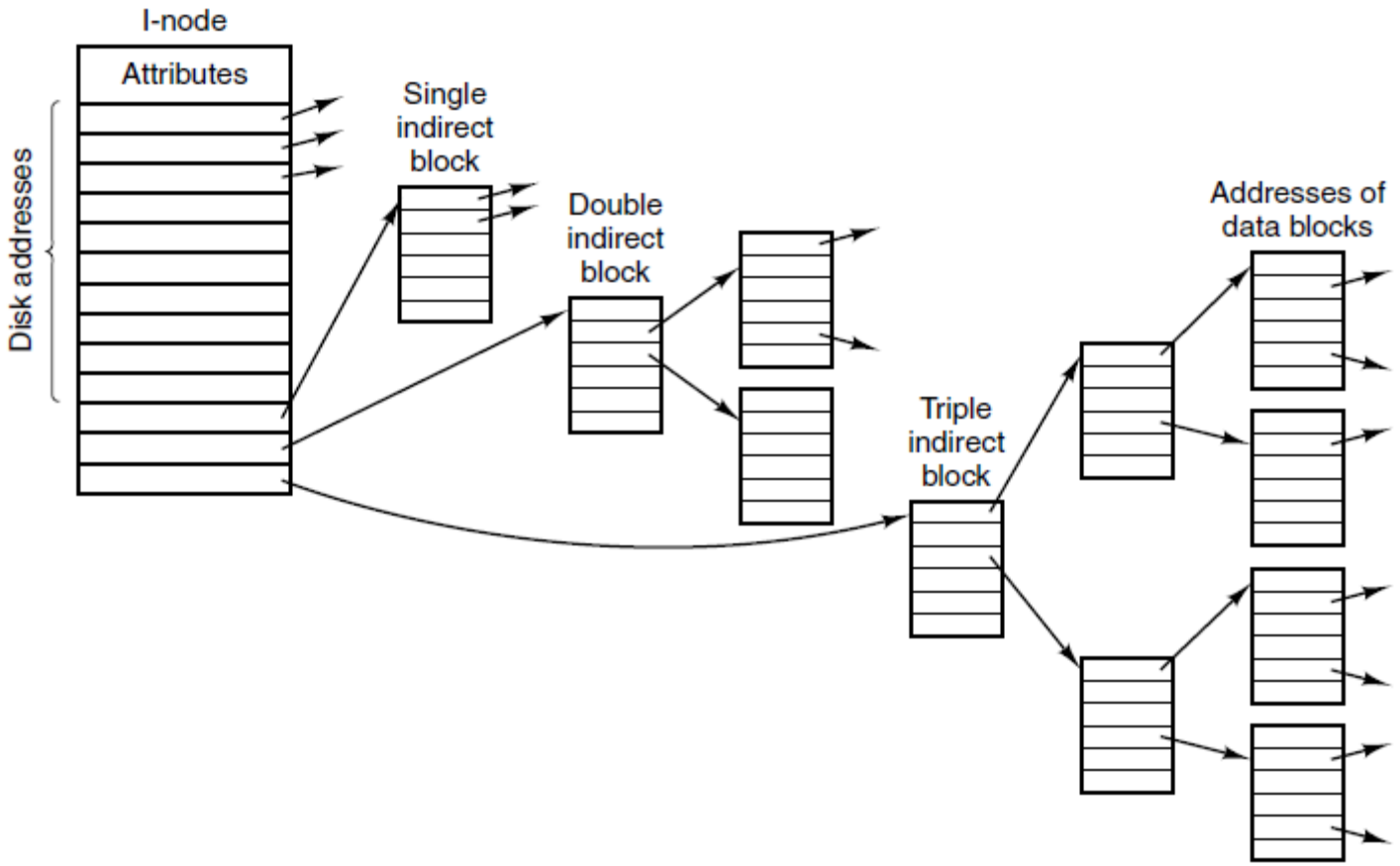


Figure 4-33. A UNIX i-node

The UNIX V7 File System (3)

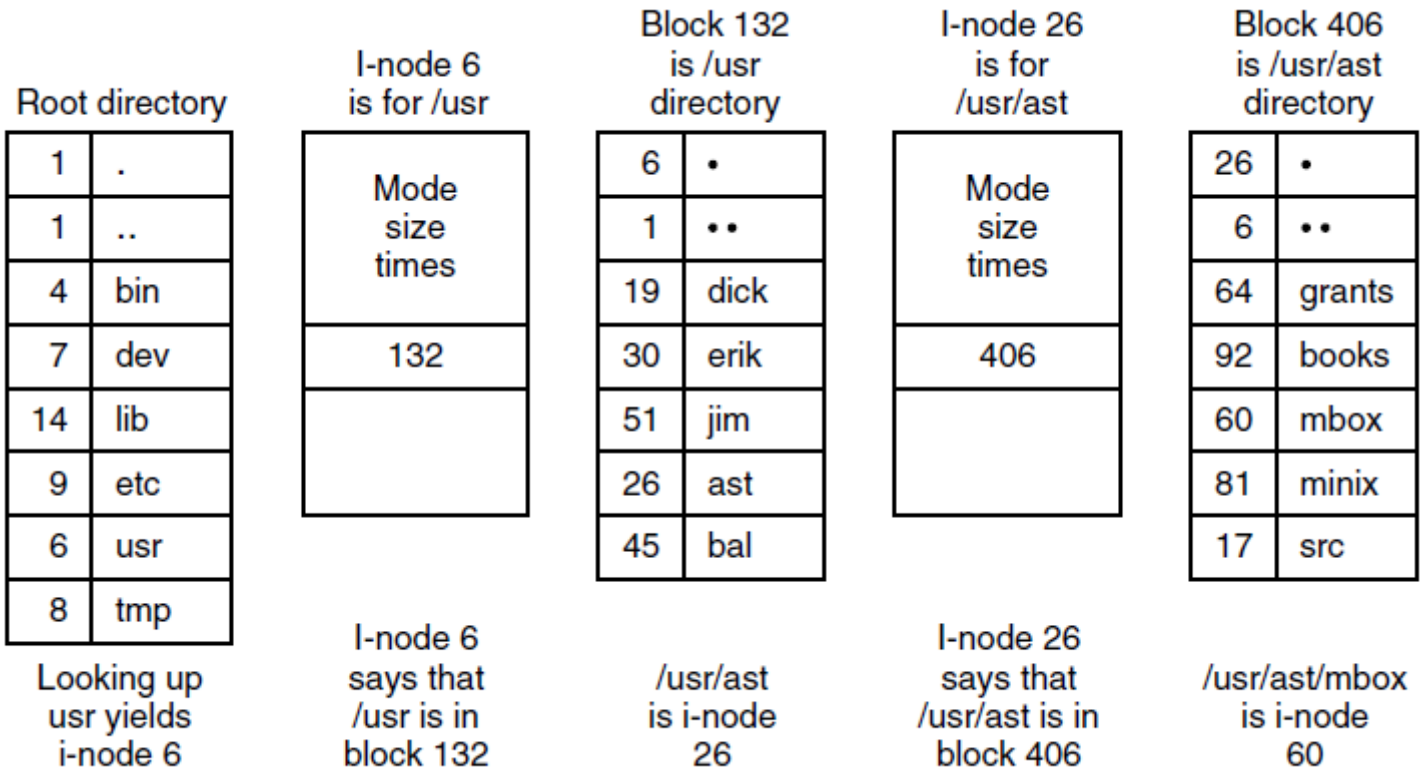


Figure 4-34. The steps in looking up */usr/ast/mbox*.

The ISO 9660 File System

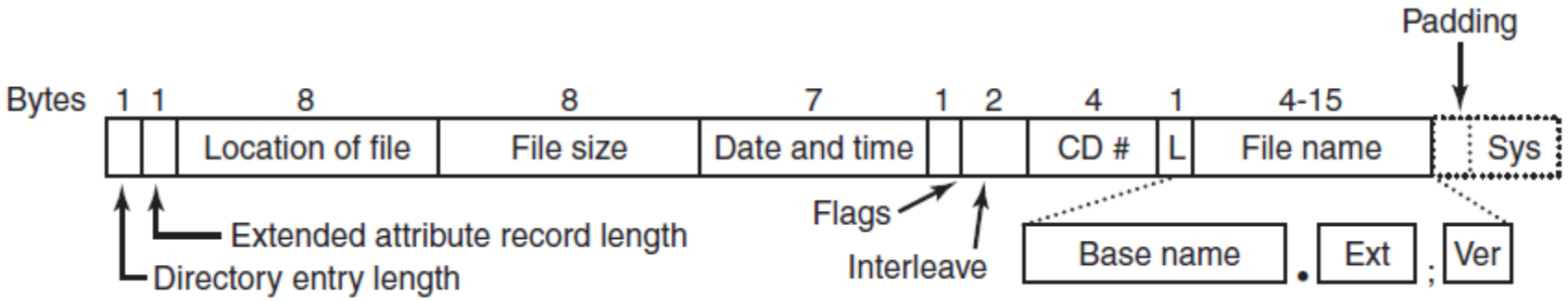


Figure 4-35. The ISO 9660 directory entry.

Rock Ridge Extensions

1. PX - POSIX attributes.
2. PN - Major and minor device numbers.
3. SL - Symbolic link.
5. NM - Alternative name.
6. CL - Child location.
7. PL - Parent location.
8. RE - Relocation.
9. TF - Time stamps.

Joliet Extensions

1. Long file names.
2. Unicode character set.
3. Directory nesting deeper than eight levels.
4. Directory names with extensions

End

Chapter 4