

File Systems

Week 09-Lecture 1

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?

FILES

Files are logical units of information created by processes.

A disk will usually contain thousands or even millions of them, each one independent of the others.

File Naming

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

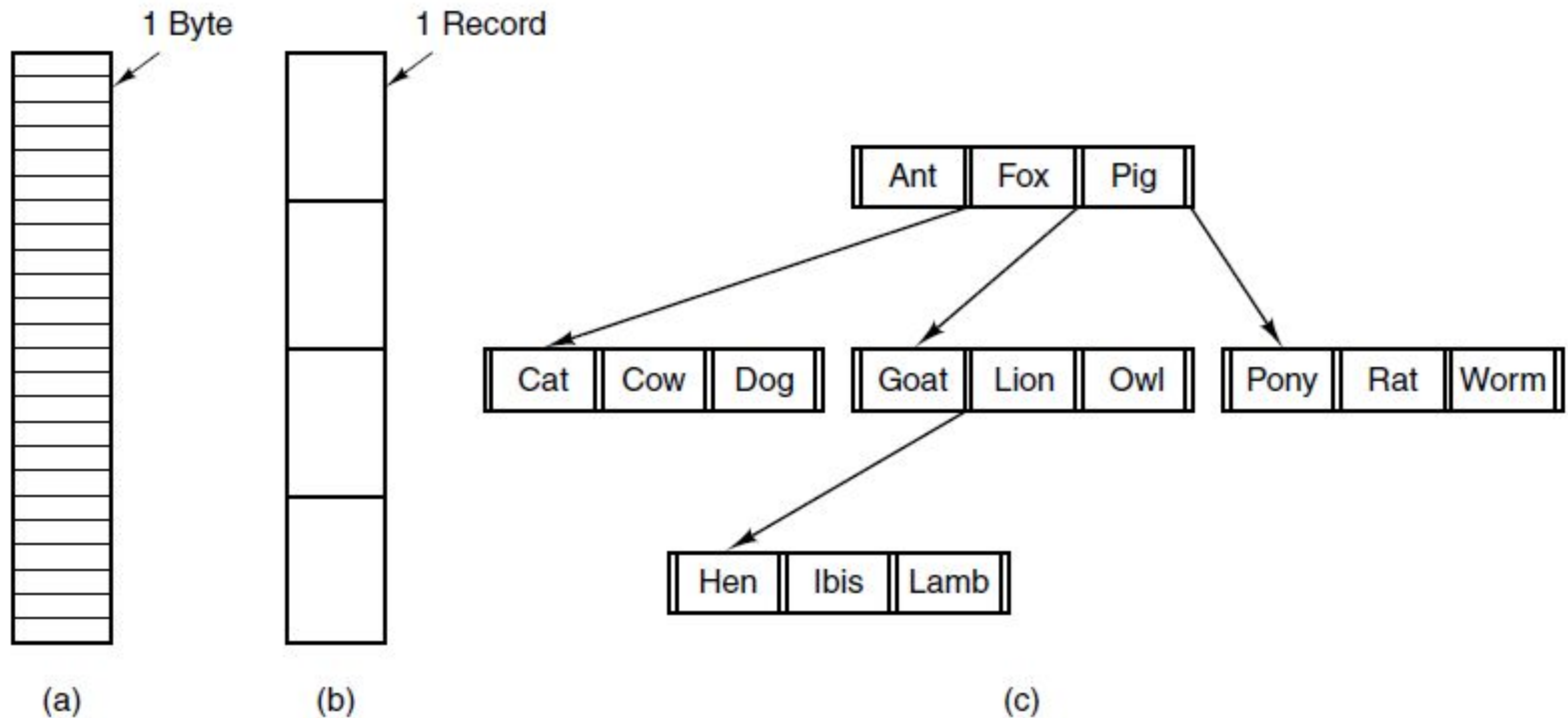


Figure 4-2. Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.

File Structure

The file in Fig. 4-2(a) is an unstructured sequence of bytes. Any meaning must be imposed by user-level programs. Both UNIX and Windows use this approach.

The file in Fig. 4-2(b) is a sequence of fixed-length records, each with some internal structure.

File Structure

The file in Fig. 4-2(c) consists of a tree of records, not necessarily all the same length, each containing a key field in a fixed position in the record.

The tree is sorted on the key field, to allow rapid searching for a particular key.

File Types

- 1) Regular files are the ones that contain user information.
- 2) Directories are system files for maintaining the structure of the file system.

File Types

- 3) Character special files are related to input/output and used to model serial I/O devices, such as terminals, printers, and networks.
- 4) Block special files are used to model disks.

Regular Files(1)

Regular files are generally either **ASCII files** or **binary files**.

ASCII Files consist of lines of text.

In some systems each line is terminated by a carriage return character.

In others, the line feed character is used.

Some systems (e.g., Windows) use both.

Regular Files(2)

Binary Files have internal structure known to programs that use them.

For Example: A simple executable binary file taken from an early version of UNIX has five sections: header, text, data, relocation bits, and symbol table (shown in next slide, Fig 4.3-a).

Second example of a binary file is an archive, also from UNIX. (Fig 4.3-b)

File Types

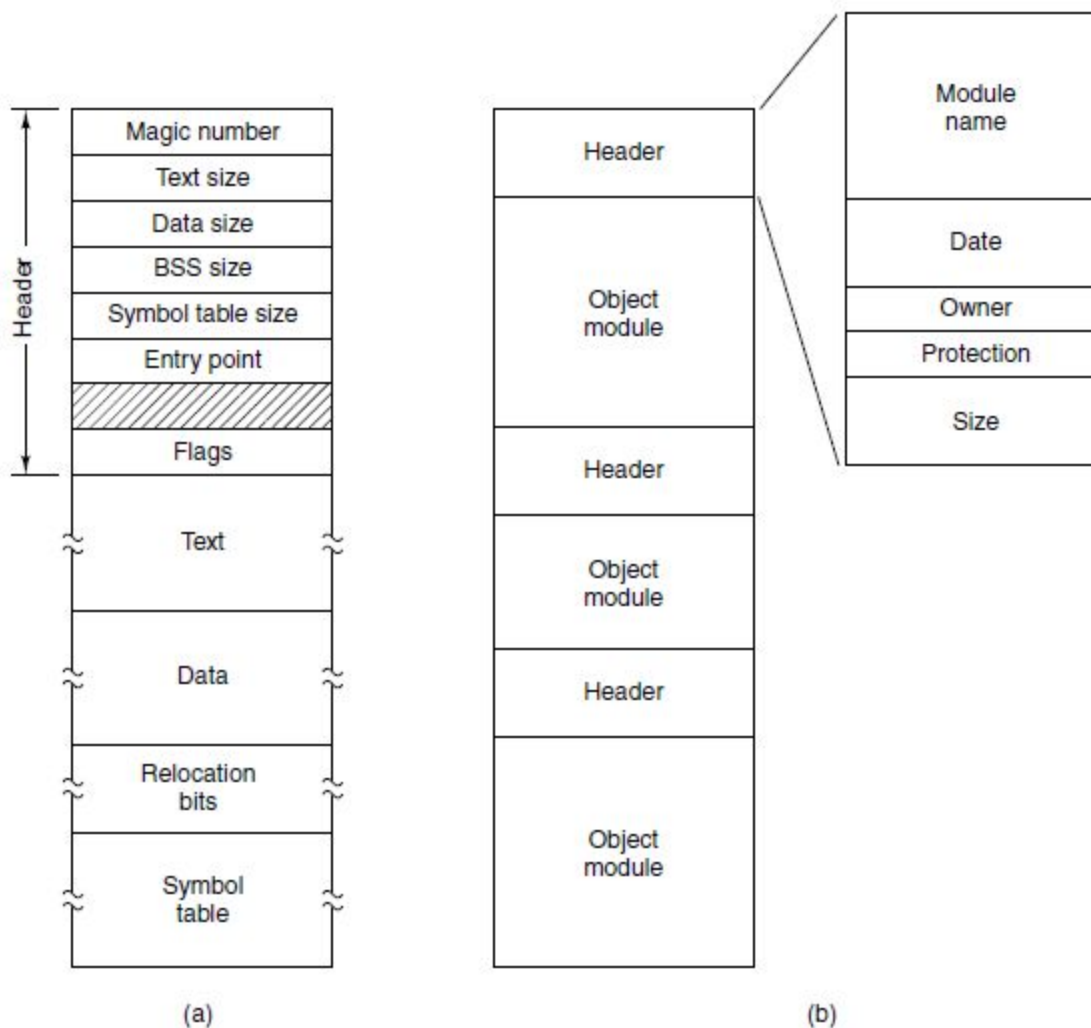


Figure 4-3. (a) An executable file. (b) An archive

File Access

Sequential Access:

Early operating systems provided only sequential access.

In these systems, a process could read all the bytes or records in a file in order, starting at the beginning, but could not skip around and read them out of order.

File Access

Random Access:

When disks came into use for storing files, it became possible to read the bytes or records of a file out of order, or to access records by key rather than by position.

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

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.

Single-Level Directory Systems

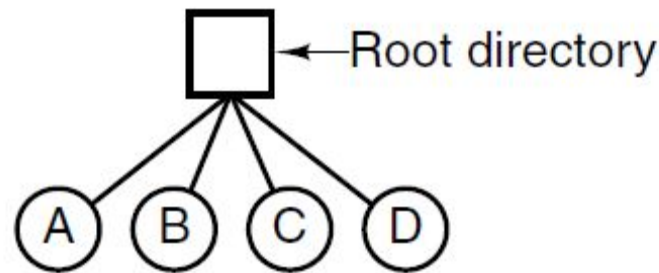


Figure 4-6. A single-level directory system containing four files.

Hierarchical Directory Systems

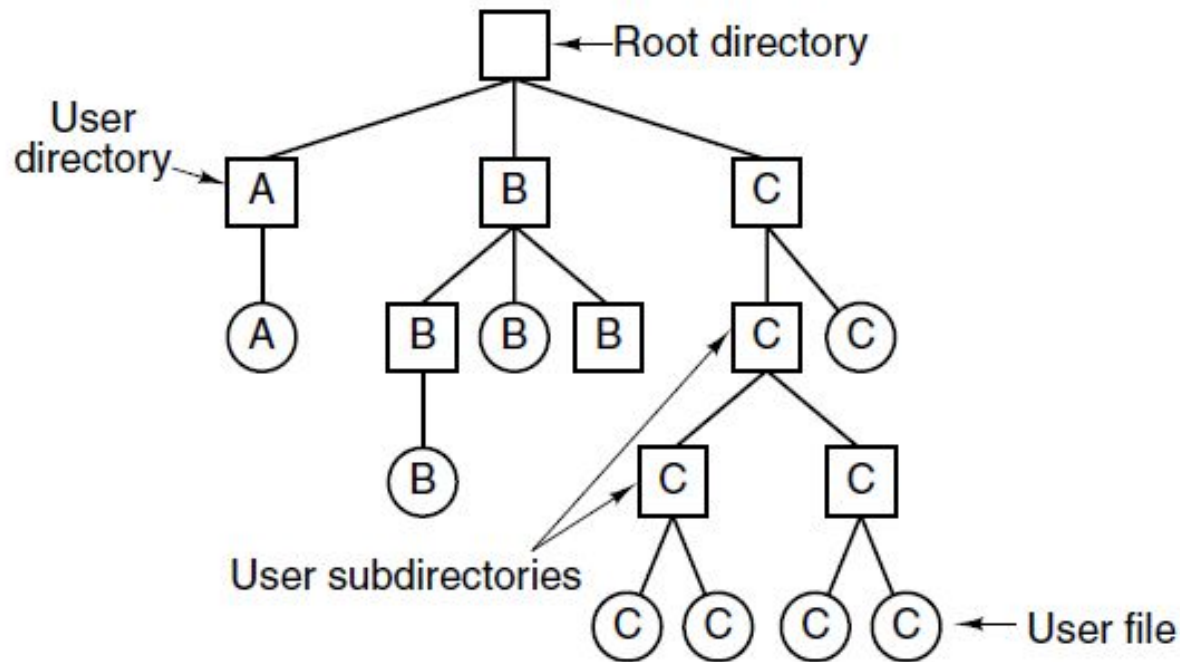


Figure 4-7. A hierarchical directory system.

Path Names

Two different methods are commonly used.

1) Absolute path name

2) Relative path name

Absolute path name

Absolute path name consists of the path from the root directory to the file.

Example:

The path `/usr/ast/mailbox` means that the root directory contains a subdirectory `usr`, which in turn contains a subdirectory `ast`, which contains the file `mailbox`.

Relative path name

Relative path name is used in conjunction with the concept of the working directory (also called the current directory).

Example:

If the current working directory is `/usr/ast`, then the file whose absolute path is

`/usr/ast/mailbox`

can be referenced simply as `mailbox`.

Path Names

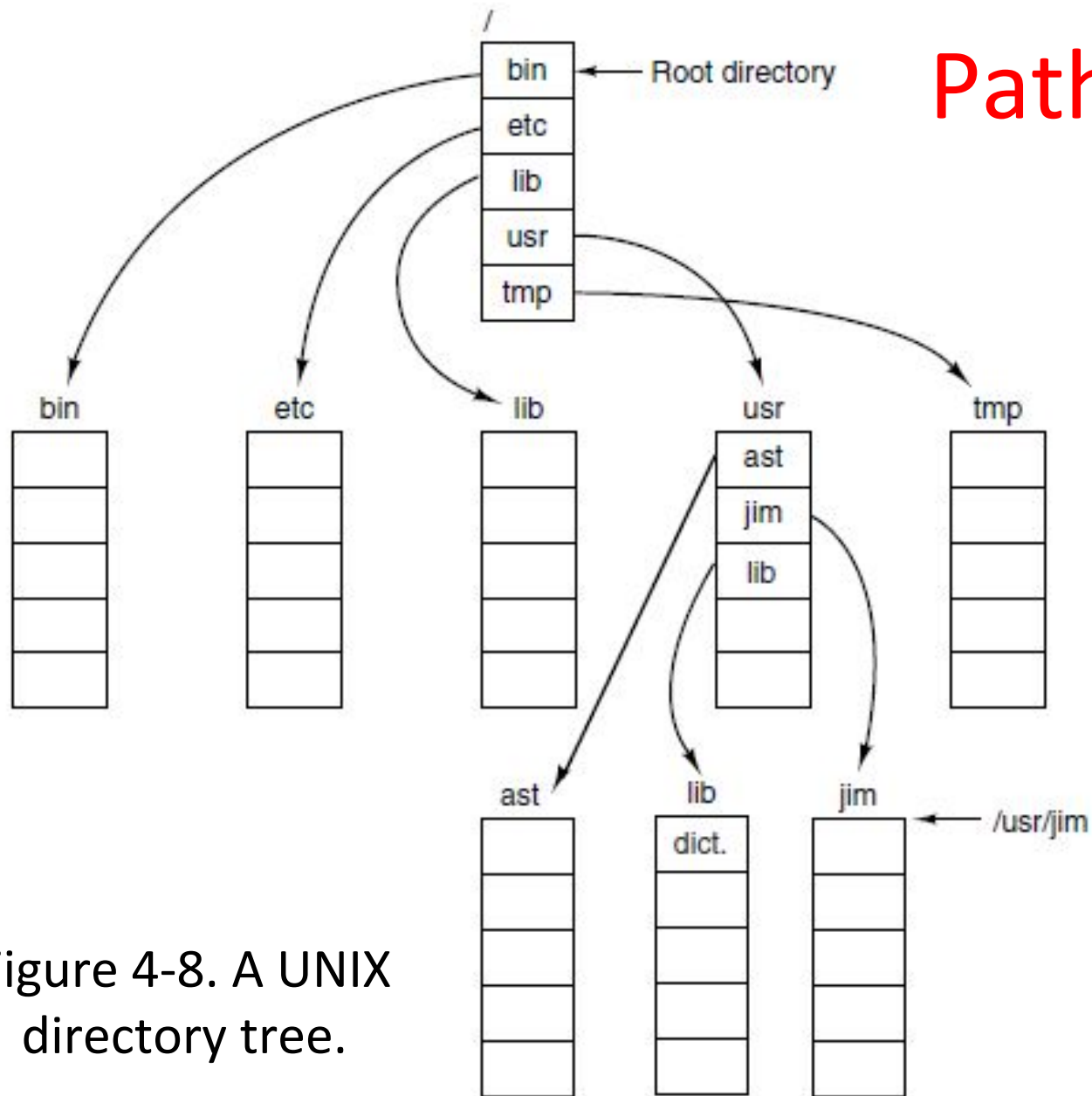


Figure 4-8. A UNIX directory tree.

Directory Operations

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

File System Layout

File systems are stored on disks.

Most disks can be divided up into one or more partitions, with independent file systems on each partition.

File System Layout

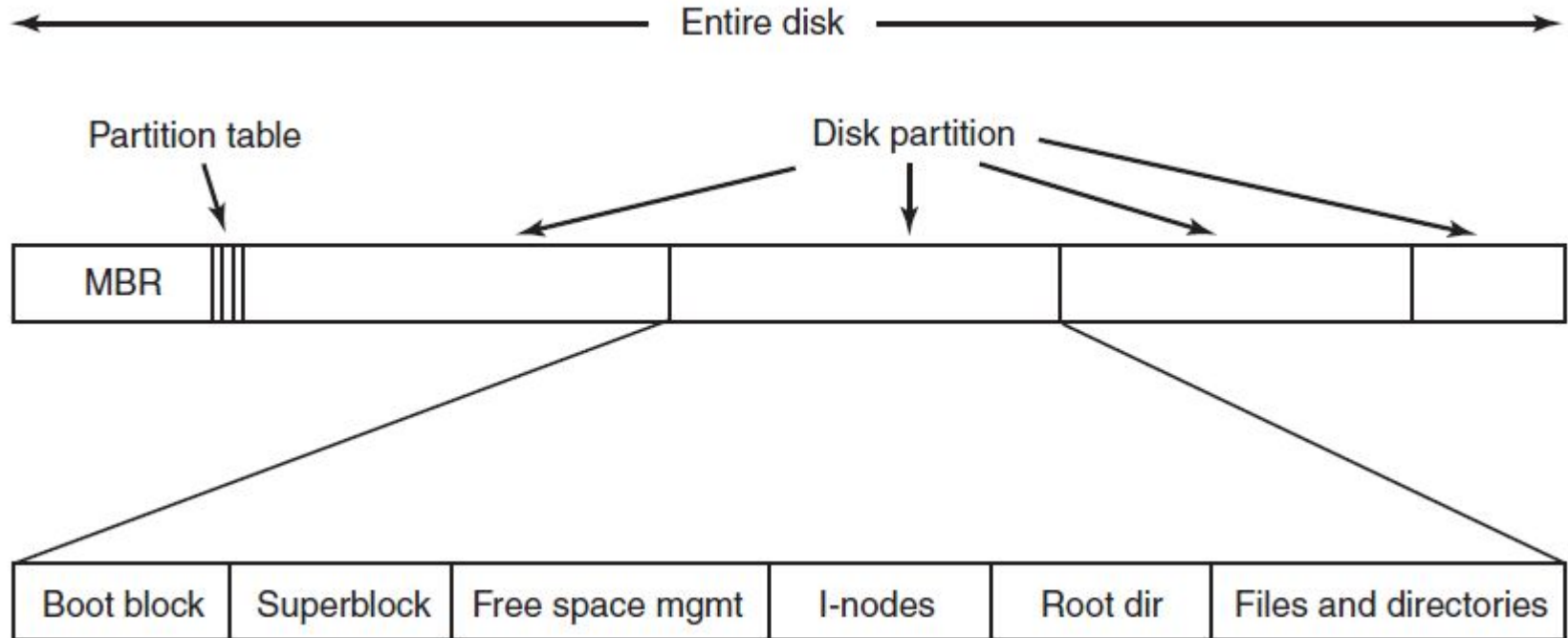


Figure 4-9. A possible file system layout.

File System Layout

Sector 0 of the disk is called the MBR (Master Boot Record) and is used to boot the computer.

The end of the MBR contains the partition table. This table gives the starting and ending addresses of each partition.

File System Layout

When the computer is booted, the BIOS reads in and executes the MBR. The first thing the MBR program does is locate the active partition, read in its first block, which is called the boot block, and execute it.

The program in the boot block loads the operating system contained in that partition.

File System Layout

The layout of a disk partition also contains superblock.

Information in the superblock includes:

- magic number: to identify the file-system type,
- the number of blocks in the file system, and
- other key administrative information.

Implementing Files

Various methods are used in implementing file storage to keep track of which disk blocks go with which file. Such as:

- 1) Contiguous Allocation
- 2) Linked-List Allocation
- 3) I-nodes

Implementing Files

Contiguous Layout

Contiguous allocation scheme store each file as a contiguous run of disk blocks.

Implementing Files

Contiguous Layout

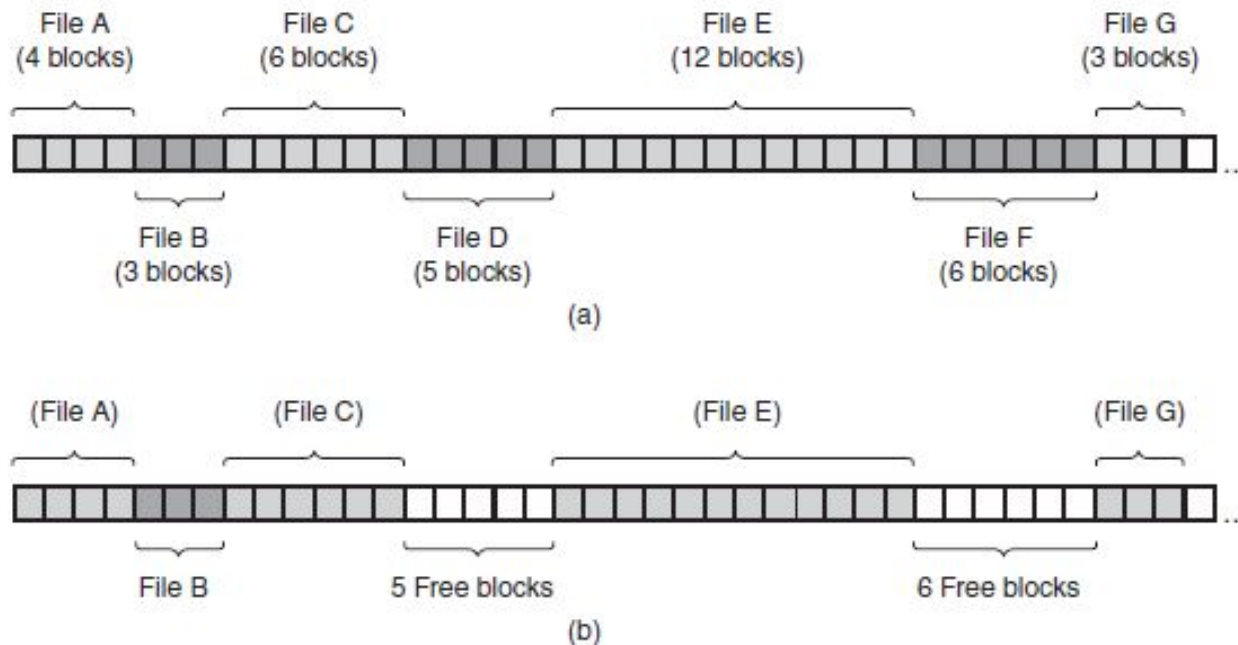


Figure 4-10. (a) Contiguous allocation of disk space for seven files. (b) The state of the disk after files *D* and *F* have been removed.

Implementing Files

Contiguous Layout

In Fig. 4-10(a), the first 40 disk blocks are shown, starting with block 0 on the left.

File A, of length four blocks, was written to disk starting at the beginning (block 0). After that a six-block file, B, was written starting right after the end of file A.

Implementing Files Linked List Allocation

The first word of each block is used as a pointer to the next one. The rest of the block is for data.

Implementing Files

Linked List Allocation

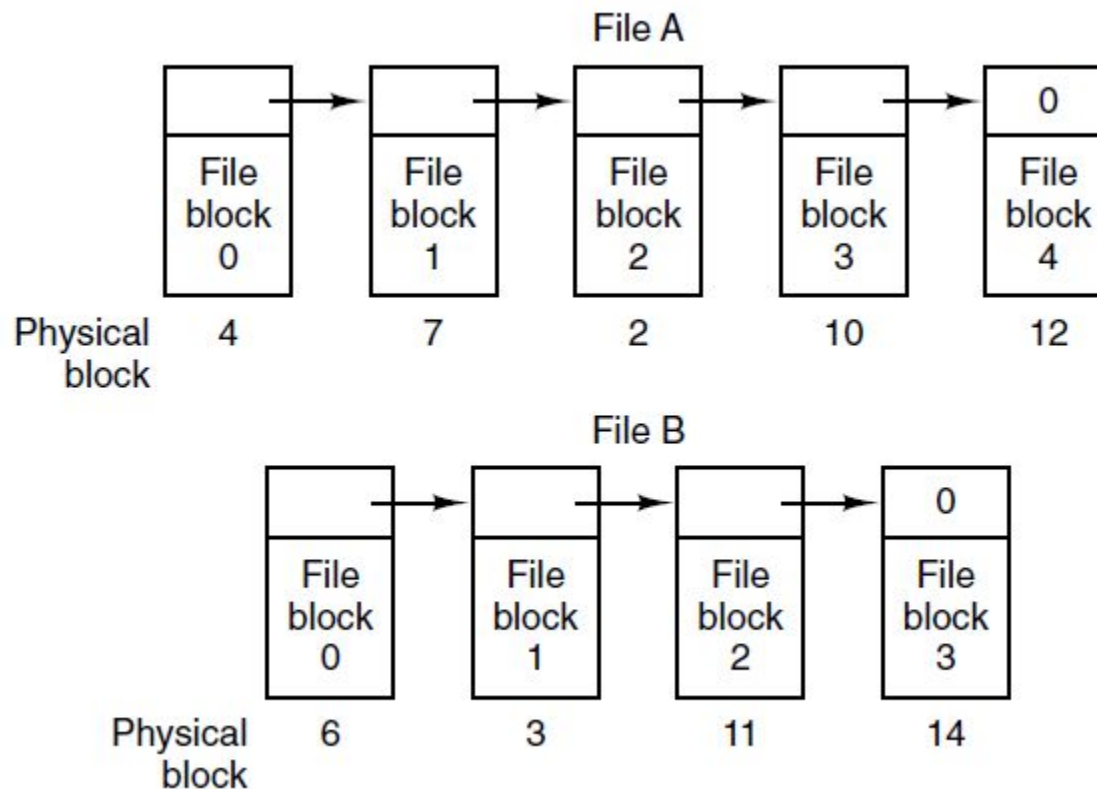


Figure 4-11. Storing a file as a linked list of disk blocks.

Implementing Files

Linked List – Table in Memory

Linked-list allocation using a table in memory takes the pointer word from each disk block and putting it in a table in memory.

Implementing Files

Linked List – Table in Memory

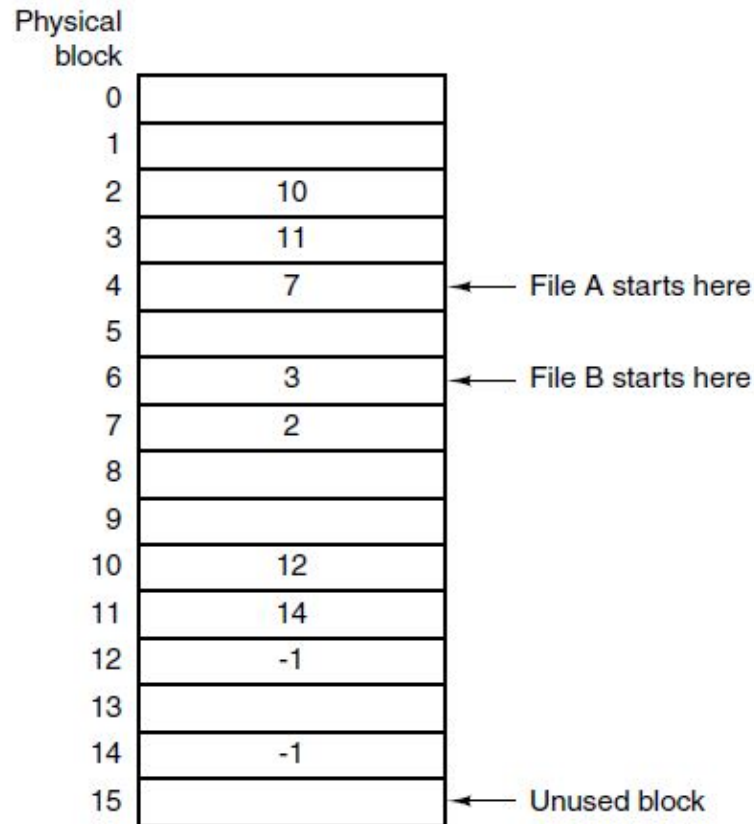


Figure 4-12. Linked list allocation using a file allocation table in main memory.

Implementing Files

Linked List – Table in Memory

Figure 4-12 shows that file A uses disk blocks 4, 7, 2, 10, and 12 and file B uses disk blocks 6, 3, 11, and 14.

Both chains are terminated with a special marker (e.g., -1) that is not a valid block number.

Implementing Files

I-nodes

A data structure called an i-node (index-node) is associated with each file, which lists the attributes and disk addresses of the file's blocks.

Implementing Files I-nodes

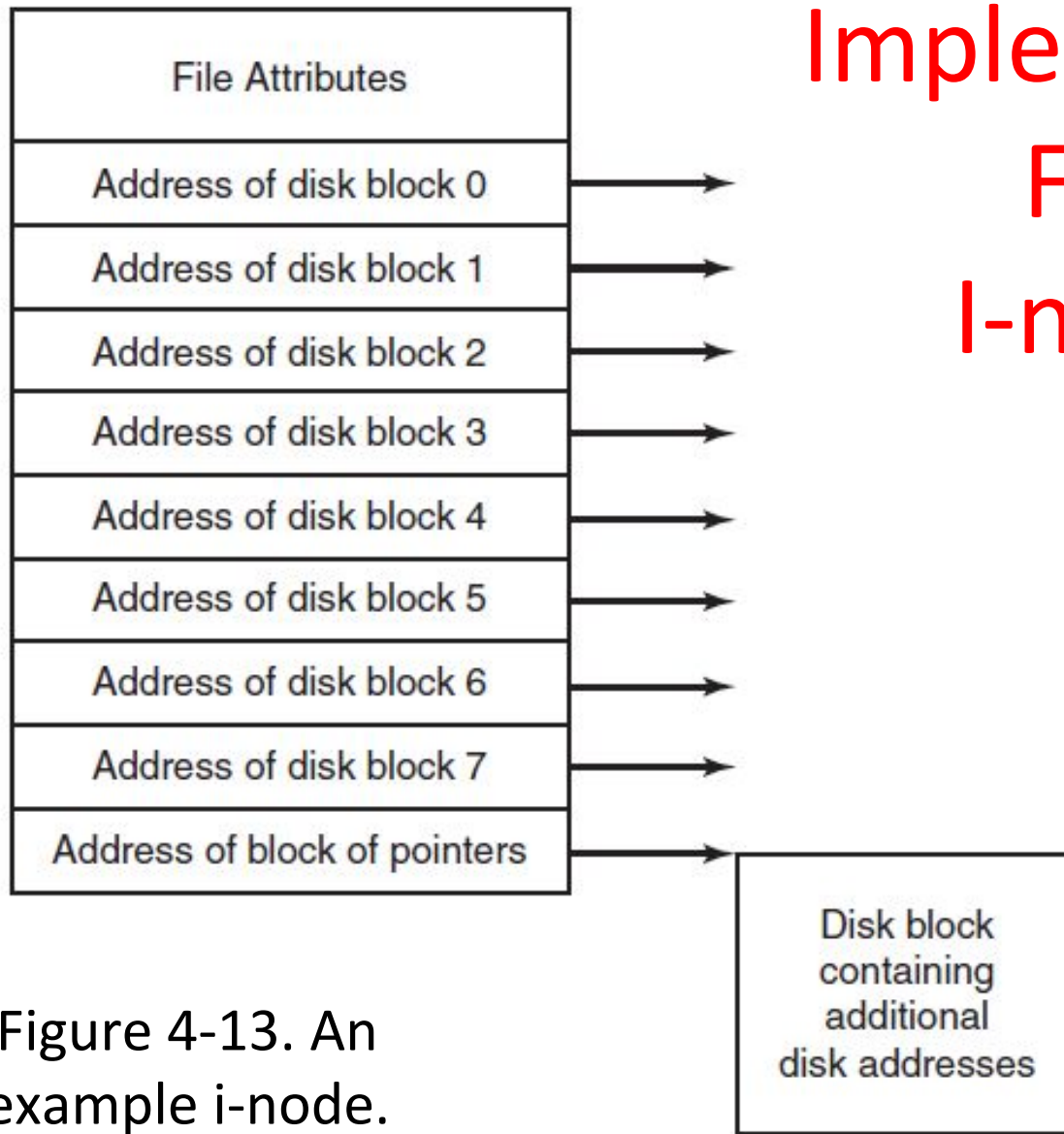


Figure 4-13. An example i-node.

Implementing Directories

When a file is opened, the operating system uses the path name supplied by the user to locate the directory entry on the disk.

The directory entry provides the information needed to find the disk blocks.

Depending on the system, this information may be the disk address of the entire file (with contiguous allocation), the number of the first block (both linked-list schemes), or the number of the i-node.

Implementing Directories (1)

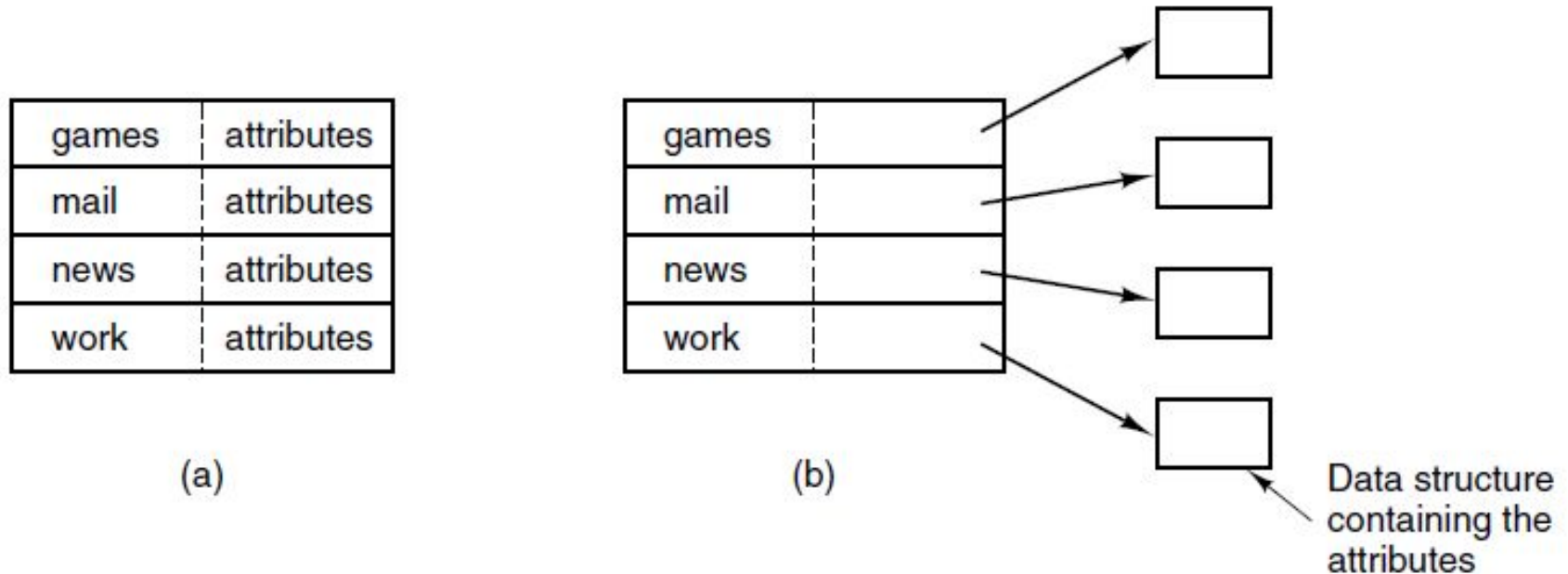


Figure 4-14. (a) A simple directory containing fixed-size entries with the disk addresses and attributes in the directory entry. (b) A directory in which each entry just refers to an i-node.

Implementing Directories (2)

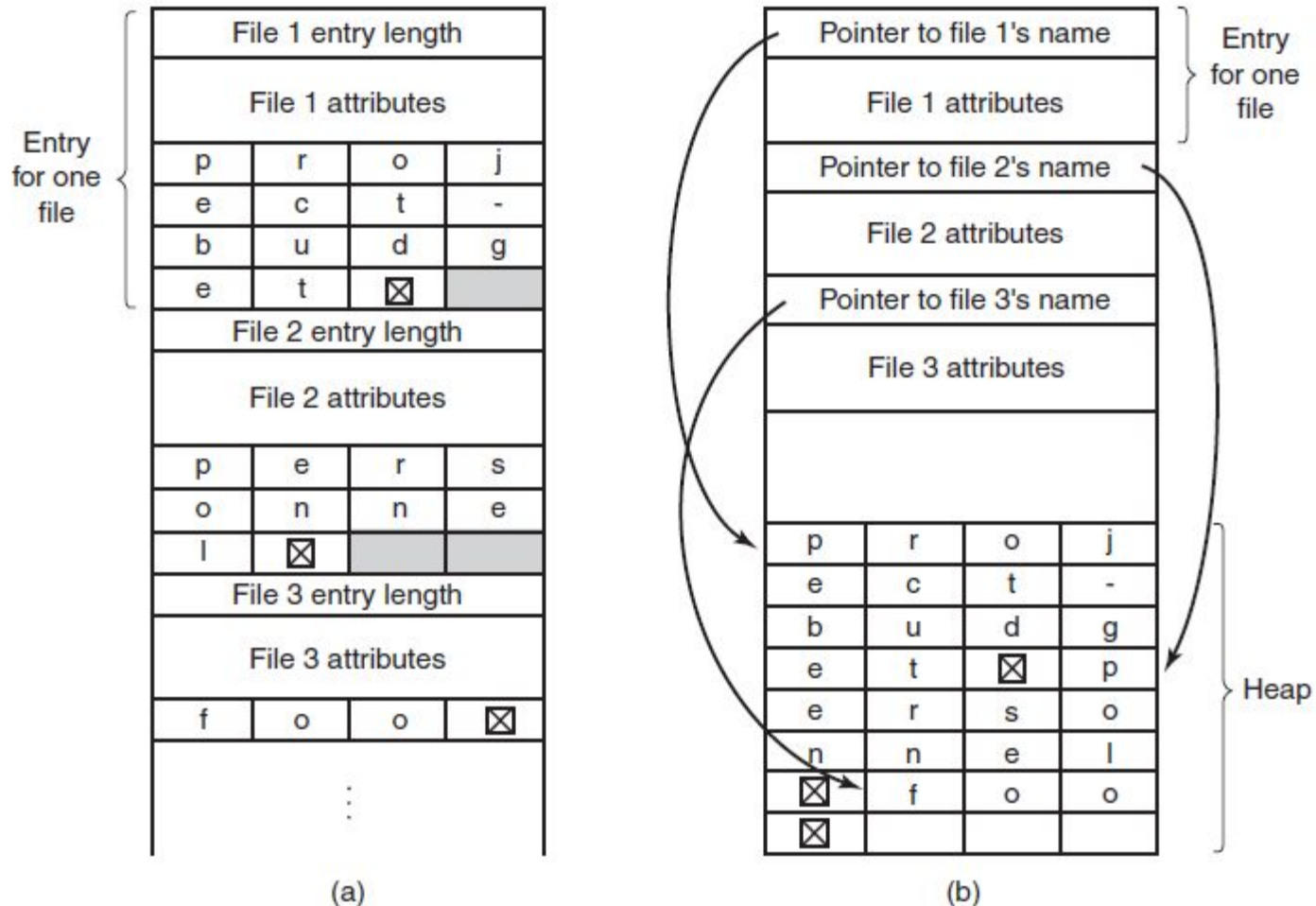


Figure 4-15. Two ways of handling long file names in a directory. (a) In-line. (b) In a heap.

Shared Files (1)

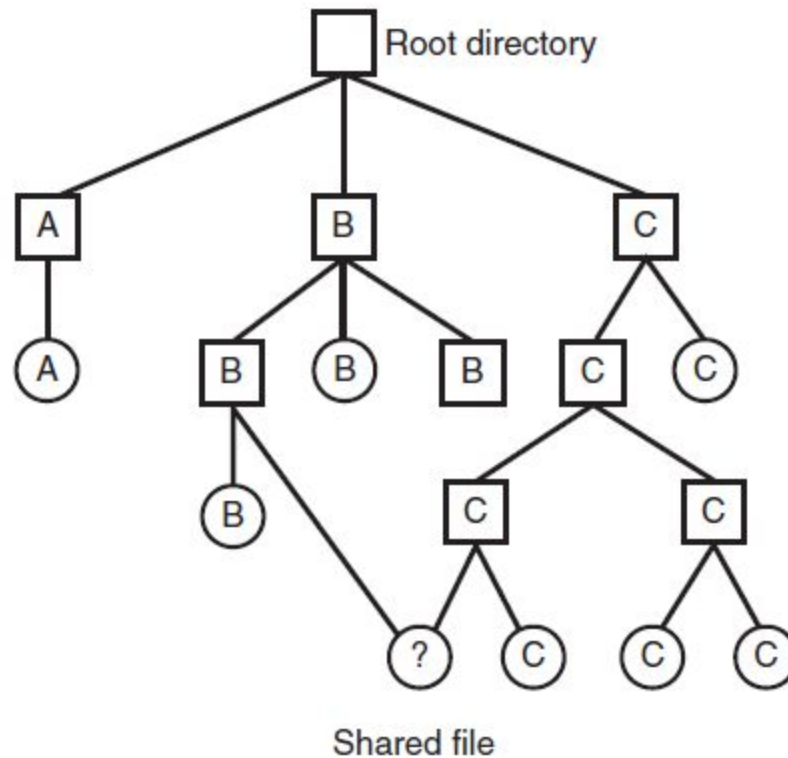


Figure 4-16. File system containing a shared file.

Shared Files (2)

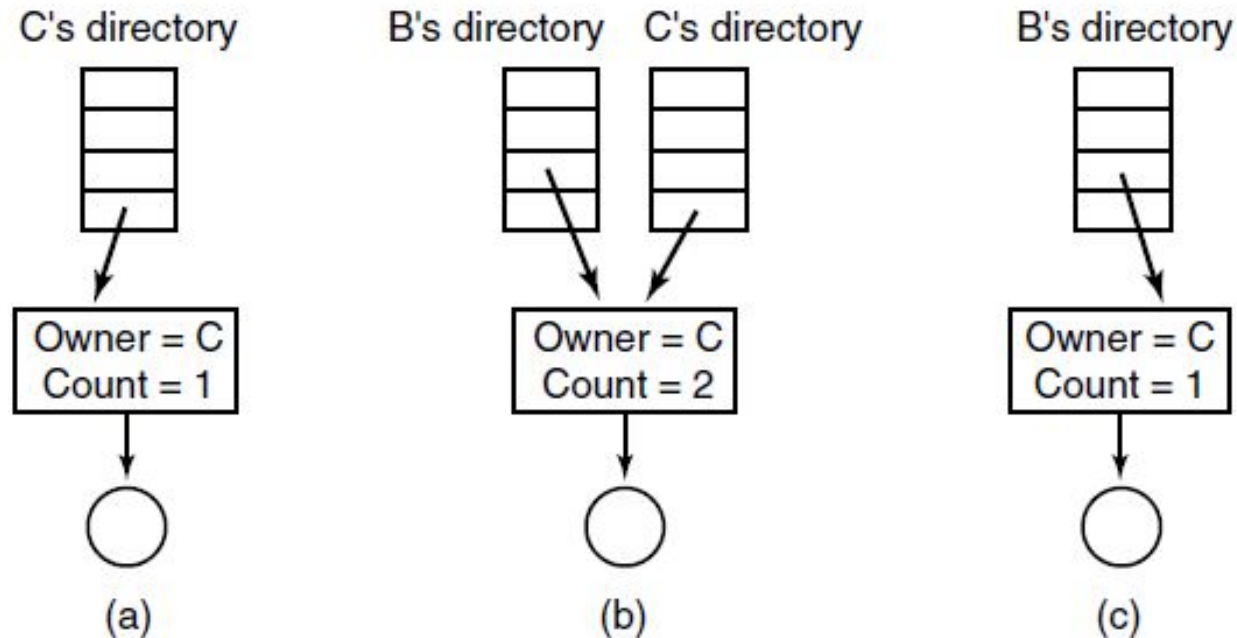


Figure 4-17. (a) Situation prior to linking. (b) After the link is created. (c) After the original owner removes the file.

END

Part 1-Chapter 4