File -System

File Concept

OS provides a uniform logical view of stored information. OS abstracts from the physical properties of its storage devices to define a logical storage unit, the **file.**

Files are mapped by the operating system onto physical devices. These storage devices are usually nonvolatile, so the contents are persistent between system reboots.

File Attributes

- Name.
- Identifier.
- Type.
- Location. This information is a pointer to a device and to the location of the file on that device.
- Size.
- Protection. Access-control information determines who can do reading, writing, executing, and so on.
- Time, date, and user identification. This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

File Operations

Creating a file. Two steps are necessary to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.

Writing a file. To write a file, we make a system call specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the file's location. The system must keep a **write pointer** to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.

Reading a file. To read from a file, we use a system call that specifies the name of the file and where (in memory) the next block of the file. Again, the directory is searched for the associated entry, and the system needs to keep a **read pointer** to the location in the file where the next read is to take place.

Repositioning within a file. The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value. Repositioning within a file need not involve any actual I/O. This file operation is also known as a file **seek.**

Deleting a file. To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, and erase the directory entry.

Truncating a file. The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged—except for file length—but lets the file be reset to length zero and its file space released.

File Types

A common technique for implementing file types is to include the type as part of the file name. The name is split into two parts—a name and an extension, usually separated by a period.

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

Figure 11.3 Common file types.

Access Methods

Sequential Access: Information in the file is processed in order, one record after the other.

Direct Access: A file is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order. For the direct-access method, the file operations must be modified to include the block number as a parameter. Thus, we have read(n), where *n* is the block number, rather than read next(), and write(n) rather than write next().

Allocation Methods

The main problem is how to allocate space to files so that disk space is utilized effectively and files can be accessed quickly.

Three major methods of allocating disk space are in wide use: **contiguous, linked, and indexed**.

Each method has advantages and disadvantages.

Contiguous Allocation

It requires that each file occupy a set of contiguous blocks on the disk. Disk addresses define a linear ordering on the disk.

Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block.

If the file is *n* blocks long and starts at location b, then it occupies blocks b, b + 1, b + 2, ..., b + n − 1.

The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file.

Advantages

Accessing a file that has been allocated contiguously is **easy**. For sequential access, the file system remembers the disk address of the last block referenced and, when necessary, reads the next block.

For direct access to block i of a file that starts at block b, we can immediately access block b + i.

Thus, both sequential and direct access can be supported by contiguous allocation.

Disadvantages

- One difficulty is finding space for a new file.

- Another problem is external fragmentation, that can be solved by compaction which is time consuming.

Linked Allocation

Linked allocation solves all problems of contiguous allocation.

With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk.

The directory contains a pointer to the first and last blocks of the file.

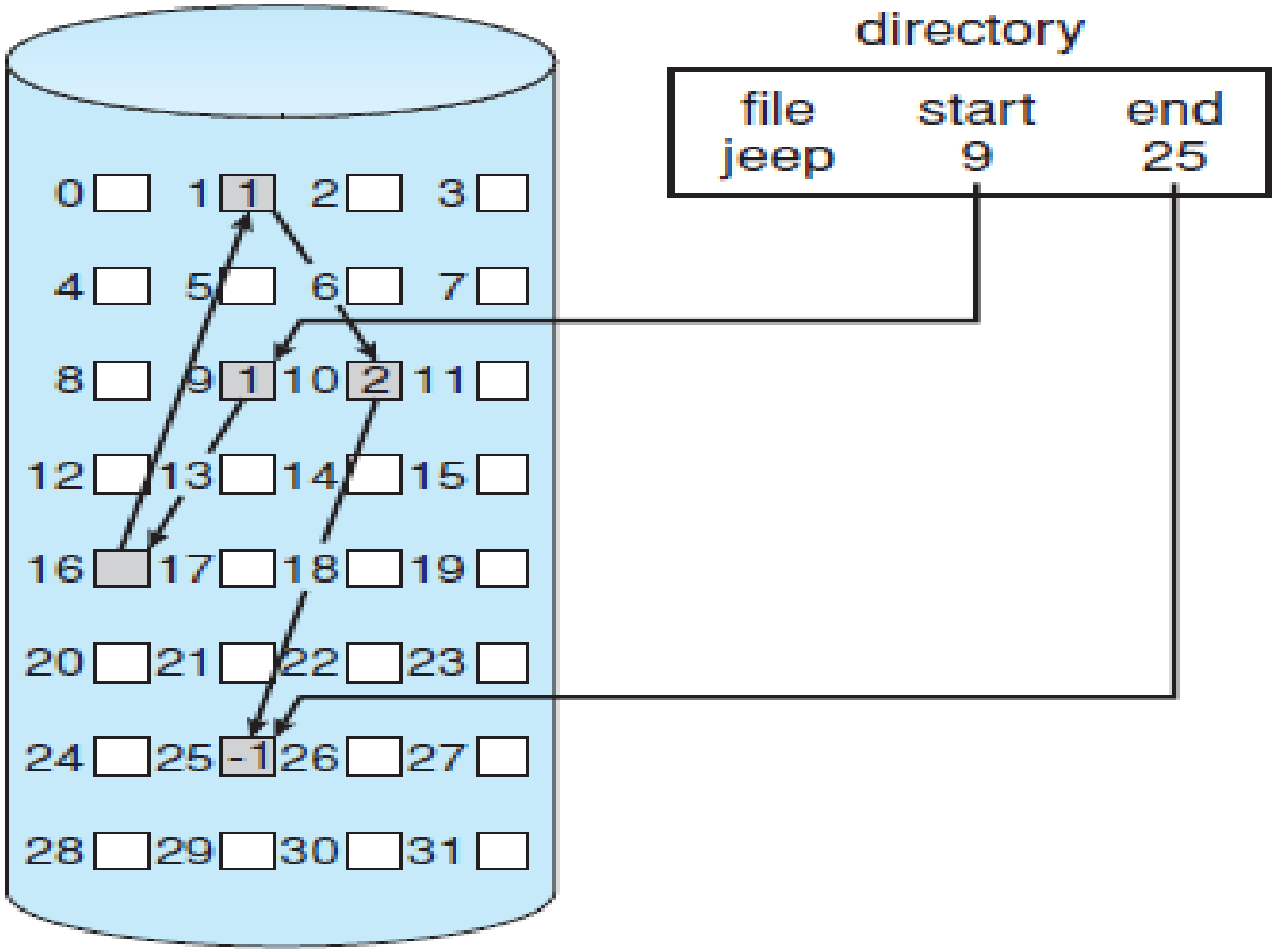

Figure 12.6 Linked allocation of disk space.

To **create** a new file, we simply create a new entry in the directory. With linked allocation, each directory entry has a pointer to the first disk block of the file. This pointer is initialized to null (the end-of-list pointer value) to signify an empty file. The size field is also set to 0.

A **write** to the file causes the free-space management system to find a free block, and this new block is written to and is linked to the end of the file.

To **read** a file, we simply read blocks by following the pointers from block to block.

Advantages

- There is no external fragmentation with linked allocation.
- Any free block on the free-space list can be used to satisfy a request.
- The size of a file need not be declared when the file is created. A file can continue to grow as long as free blocks are available.
- Consequently, it is never necessary to compact disk space.

Disadvantages

- The major problem is that it can be used effectively only for sequential-access files.
- The space required for the pointers

Free-Space Management

To keep track of free disk space, the system maintains a free-space list.

The free-space list records all free disk blocks—those not allocated to some file or directory.

To create a file, we search the free-space list for the required amount of space and allocate that space to the new file. This space is then removed from the free-space list.

When a file is deleted, its disk space is added to the free-space list.

There are a number of methods to implement the free-space list, one of them is **Bit Vector**

Bit Vector

Frequently, the free-space list is implemented as a bit map or bit vector. Each block is represented by 1 bit. If the block is free, the bit is 1; if the block is allocated, the bit is 0.