File Management in Win

File Concept

Since main memory is usually too small to contain all the data and programs permanently, the computer system must provide secondary storage to back up main memory. The file system provides the mechanism for on-line storage of and access to both data and programs residing on the disks.

The operating system implements the abstract concept of a file by managing mass-storage media, such as tapes and disks, and the devices that control them. Also, files are normally organized into directories to make them easier to use. Finally, when multiple users have access to files, it may be desirable to control by whom and in what ways (for example, read, write, append) files may be accessed.

The files are mapped by the operating system onto physical devices. The devices that attach to a computer vary in many aspects. Some devices transfer a character or a block of characters at a time. Some can be accessed only sequentially, others randomly. Some transfer data synchronously, others asynchronously. They can be read-only or read-write. They vary greatly in speed. In many ways, they are also the slowest major component of the computer. Because of all this device variation, the operating system needs to provide a wide range of functionality to applications, to allow them to control all aspects of the devices.

Computers can store information on various storage media, such as magnetic disks, magnetic tapes, and optical disks. So that the computer system will be convenient to use, the operating system provides a uniform logical view of information storage. The operating system 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 through power failures and system reboots.

A file is a named collection of related information that is recorded on secondary storage. From a user's perspective, a file is the smallest allotment of logical secondary storage; that is, data cannot be written to secondary storage unless they are within a file.

Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic, alphanumeric, or binary, files may be free form, such as text files. In general, a file is a sequence of bits, bytes, lines, or records.

Types of Files

Windows O.S. recognizes many different types of files. The type depends on in which application the file has been created. Below are some of the examples of common file types identified by their extension or by the icon.

Many different types of information may be stored in a file-source programs, object programs, executable programs, numeric data, text, payroll records, graphic images, sound recordings, and so on. *A* file has a certain defined which depends on its type. *A text* file is a sequence of characters organized into lines (and possibly pages). *A source* file is a sequence of subroutines and functions, each of which is further organized as declarations followed by executable statements. An *object* file is a sequence of bytes organized into blocks understandable by the system's linker. An *executable* file is a series of code sections that the loader can bring into memory and execute.

File Extension	File Type
.txt	A text file
.bmp	Bit Map File
.docx	MS-Word file
.xlsx	MS-Excel file
.Zip or .rar	A compressed file
.pdf	A portable Document Format file
.htm	A web page file

File Attributes

A file is named, for the convenience of its human users, and is referred to by its name. *A* name is usually a string of characters, such as *example.cpp*. Some systems differentiate between uppercase and lowercase characters in names, whereas other systems do not. When a file is named, it becomes independent of the process, the user, and even the system that created it. For instance, one user might create the file *example.cpp,* and another user might edit that file by specifying its name.

The file's owner might write the file to a flash memory, send it in an e-mail, or copy it across a network, and it could still be called *example.cpp* on the destination system.

A file's attributes vary from one operating system to another but typically consist of these:

Name. The symbolic file name is the only information kept in human readable form.

Identifier. This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.

Type. This information is needed for systems that support different types of files.

Location. This information is a pointer to a device and to the location of the file on that device.

Size. The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.

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

To define a file properly, we need to consider the operations that can be performed on files. The operating system can provide system calls to create, write, read, reposition, delete, and truncate files. Let's examine **what the operating system must do to perform** each of these six basic 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 should be put. 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. Once the read has taken place, the read pointer is updated. Because a process is usually either reading from or writing to a file, the current operation location can be kept as a per-process . Both the read and write operations use this same pointer, saving space and reducing system complexity.

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/0. 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, so that it can be reused by other files, 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.

These six basic operations comprise the minimal set of required file operations. Other common operations include ***appending*** new information to the end of an existing file , ***renaming*** an existing file and ***execute*** (Load the file into memory and execute it). These primitive operations can then be combined to perform other file operations. For instance, we can create a ***copy*** of a file, or copy the file to another I/O device, such as a printer or a display, by creating a new file and then reading from the old and writing to the new.