FILE MANAGEMENT

File System and File System Organization

For most users, the file system is the most visible aspect of an operating system. It provides the mechanism for on-line storage of and access to both data and programs of the operating system and all the users of the computer system. In definition, a **File system** is the sub-system of the operating system that provides services to users and applications in the use of files. The file system consists of two distinct parts: **a collection of files**, each storing related data, and **a directory structure**, which organizes and provides information about all the files in the system. A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directories. In respect to file management, the operating system performs the following activities;

- Keeps track of information, location, uses, status etc. The collective facilities are often known as file system.
- Decides who gets the resources (files).
- Allocates the resources (files) to both programs and users.
- De-allocates the resources (files) to both programs and users of the computer system.

File concept

Computers store information on various storage media such as magnetic disks, magnetic tapes and optical disks. To allow convenient usage of a computer system, the operating system provides a uniform logical view of stored information. The operating system abstracts the physical storage devices to define a logical storage unit, which is the file. Files are mapped by the operating system onto physical devices which are usually non-volatile, meaning the contents are persistent between computer reboots.

A File is a named collection of related information that is recorded on secondary storage. From user's perspective, a file is the smallest allotment of logical secondary storage, i.e. data cannot be written to secondary storage unless they are within a file. Information stored in a file is defined by the creator of the file and such information may be of different types, which may include, source or executable programs, numeric or text data, music, videos, photos etc. Depending on the type of a file, files have certain defined structures. For example:

- **A text file** is sequence of characters organized into lines (and possibly pages).

- **A source file** is a sequence of functions, each of which is further organized as declarations followed by executable statements.
- **An executable file** is a series of code sections that the loader can bring into memory and execute.

The purpose of file is to hold data required for providing information and therefore, files can be viewed as logical and physical files.

- **A Logical file** is a file viewed in terms of what data items its records contain and what processing operations may be performed on the file.
- **A Physical file** is a file viewed in terms of how the data is stored on storage device such as magnetic disc and how processing operations are made possible.

File attributes

For the convenience of human users, each file is given a specific name and therefore, a file is referred to by its name. A name is usually a string of characters, e.g., *example.doc*. Once a file is named, it becomes independent of the process, the user, and even the system that created it. For example, one user might create a file *example.doc* and another user might edit that file by specifying its name. The file owner might write the same file to a USB disk, copy it, send it across a network and it could be still called *example.doc* on the destination system.

File attributes vary from one operating system to another but typically a file consists:

- **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's a non-human readable name of 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

Any file system provides not only a means to store data organized as files, but a collection of functions that can be performed on files. Typical operations include:

- **Create:** A new file is defined and positioned within the file system. Two necessary steps involved for this operation. First, a space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.
- **Write:** a process updates a file, either by adding new data that expands the size of the file or by changing the values of existing data items in the file. Usually a system call is made specifying both the name of file and the information to be written to that file.
- **Read:** a process reads all or portion of the data in a file. To read from a file a system call is made specifying the name of file and where (in memory) the next block of the file should be put.
- **Delete:** A file is removed from the file structure and destroyed. Here the directory is searched for the named file. Once found, all file space is released so that other files can reuse the space. Lastly, the directory entry of that file is erased.
- **Reposition:** this involves moving a file from one directory to another or a different location in the storage device. Usually, the directory is searched for the appropriate entry and the current file position pointer is repositioned to a given value.
- **Truncate:** The user may want to erase the contents of a file but keep its attributes. Rather than deleting the file and recreating it afresh, this operation allows all attributes to remain unchanged – except for file length – but lets the file be reset to length zero and its file space released.

Other common operations include append, where new information is added to the end of an existing file and rename where an existing file is given another name. Most of the operations mentioned involve searching the directory for entry associated with the named file. To avoid constant searching, many systems require that **open** system call be made before a file is first used. The operating system keeps a table, called the **open-file table,** containing information about all open files. When the file is no longer being actively used, it is closed using **close** system call where the operating system removes its entry from open-file table.

File Access Methods

When the information stored in file is used, that information must be accessed and read into computer memory. There are several ways for accessing this information and choosing the right method for a particular application poses the major design problem.

1. **Sequential access**

   This is the simplest access method in which, information in the file is processed in order, one record after the other. This method of access is by far the most common. For example, editors and compilers usually access files in this fashion.

2. **Indexed sequential access**

   This mechanism is built upon the base of sequential access. An index is created for each file. Index contains pointer to a block. To find a record in the file, the index is first searched and then the pointer is used to access the file directly and to find the desired record. **NB:** index is searched sequentially and its pointer is used to access the file directly.

3. **Direct access**

   This method is also referred to as relative access. In this method, a file is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order. A file is viewed as a numbered sequence of blocks or records. As a result, there are no restrictions on the order of reading or writing for this method of access. Direct access files are of great use for immediate access to large amounts of information. A good example is a database in which, when a query concerning a particular subject arrives, the block containing the answer is computed and then that block is read directly to provide the desired information.

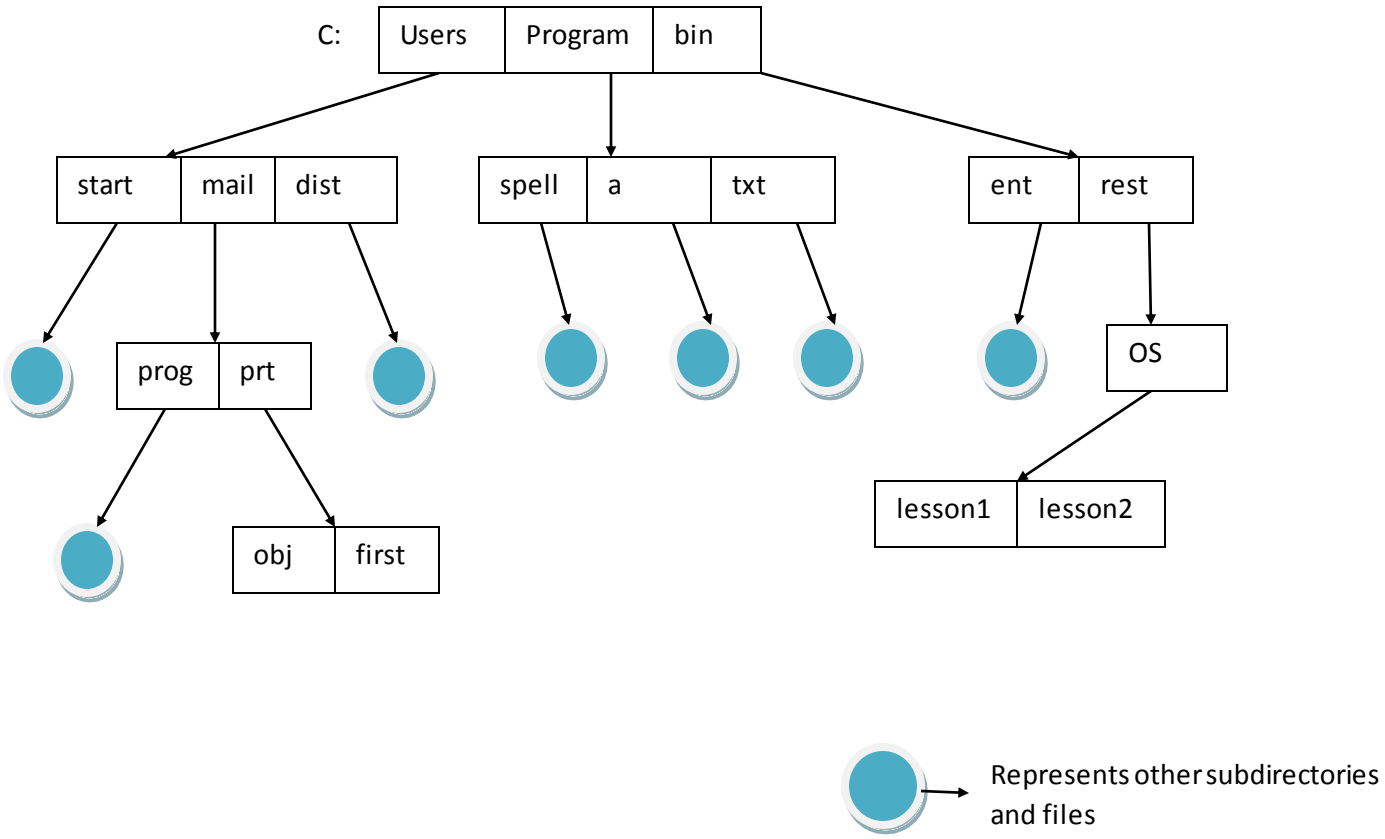
Directory structure organization

There are typically thousands, millions, and even billions of files within a computer which are stored on random access storage devices. Files are usually segregated into groups which are easier to manage and act upon. This organization involves the use of directories. A directory contains a set of files or subdirectories. In modern operating systems, directories are tree-structured which allow users to create their own subdirectories and to organize their files accordingly. In this tree structure, the tree has a root directory and every file in the system has a unique path name. **A path name** is defined by the user name and a file name.

In normal use, each process has a **current directory,** i.e. the directory containing most of the files that are of current interest to the process. When reference is made to a file, the current

directory is searched. If a needed file is not in the current directory, then the user must either specify a path name or change the current directory to be the directory holding that file. Path names can be of two types: **absolute and relative. An absolute path name** begins at the root and follows a path down to the specified file, giving the directory names on the path. **A relative path name** defines a path from the current directory.

For example, consider the figure below



If the current directory is ***C: /Users/mail*** then the relative path ***prt/first*** refers to the same file as the absolute path ***C: /Users/mail/prt/first*** refers to.