FILE MANAGEMENT

Computer applications store and retrieve information. When a process is running, it can store information within its address space. The storage capacity is restricted to the size of the virtual address space. For some applications, this may be enough, whereas for others it is not. A problem with keeping information within the address space is that when a process terminates, that information is lost. For some applications like database systems, we need a permanent form of storage.

It is necessary to have multiple processes accessing the same information or parts of it at the same time.

Consequently, we have three essential requirements for long term storage:

- It must be possible to store a very large amount of information
- The information must survive the termination of the process using it
- Multiple processes must be able to access the information concurrently

The solution is to store the information on disks (or other media) in units called **files**. Processes can then read them and write new ones if need be. The information stored in files must be **persistent** (not affected by process creation and termination). A file should only disappear when its owner explicitly deletes it. Files are managed by the operating system. **The part of the operating system that deals with** file naming, structure, implementation, access, storage and protection is referred to as the **file system**.

To the users the most important aspect is the interface of the file system: what constitutes a file, how a file is named and protected and the operations allowed on files.

5.1 Files

5.1.1 File Naming

Files are an abstraction mechanism. They provide a way to store information on disk and read it later on. This is done in such a way that shields the user from the details of how and where the information is stored and how the disk actually works.

When a process creates a file, it gives it a name. When the process terminates, the file continues to exist and can be accessed by other processes using its name. The rule for naming files varies from system to system, but all operating systems allow strings of one to eight letters as legal file names. Many operating file systems support names as long as 255 characters. Some are case-sensitive (UNIX), while others (MS-DOS) are not.

Many operating systems support two-part file names, with the two parts separated by a period. The part following the period is called the **file extension** and usually indicates something about the file. For example, *html,* for WWW Hypertext Markup Language document, *pas,* for Pascal files, *doc*, for MS Word documents, *txt*, for text files etc. In MS-DOS, for example, file names are 1 to 8 characters, plus an optional extension of 1 to 3 characters

File Structure

Files can be structured in several ways. The three most common possibilities are:

- Unstructured sequence of bytes
- Fixed length records
- Tree of variable records

a) **Unstructured sequence of bytes** – The operating system does not care what is in the file. All it sees are bytes. Any meaning on the content of the file must be interpreted by the user programs. Both UNIX and MS-DOS use this approach. This approach gives maximum flexibility. User programs can put whatever they want in the files.

b) **Fixed-length records** – In this structure a file is sequence of fixed length records, each with some internal structure. In this model, a read operation reads one record and the write operation appends or overwrites one record. An example of an operating system that viewed files as a sequence of fixed-length records was CP/M. It used a 128-character record.

c) **Tree of variable records** – In this organization, a file consists of trees of records, not necessarily of 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.

5.1.3 File Types

Many operating systems support several types of files. UNIX and DOS for example have regular files and directories. UNIX also has **character** and **block** special files. **Regular files** are those that contain user information. **Directories** are system files for maintaining the structure of the file system. Character special files are related to input/output and used to model serial I/O devices such as terminals, printers and networks. Block special devices are used to model disks.

Regular files are generally either ASCII files or binary files. ASCII files consist of lines of text that need not to be of the same size, but are terminated by the carriage return or line feed character. ASCII files can be displayed and printing as is, and can be edited by an ordinary text editor. Other files are binary files, which simply mean they are not ASCII files. Listing them on the printer gives an incomprehensible random junk.

5.1.4 File Access

Early operating systems provided only one kind of file access: **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 and read them out of order. Sequential files can be rewound, however, so they can be read as often as needed. Sequential files are convenient for magnetic tapes and not disks.

With disks it is possible to read the bytes or records of a file out of order, or access records by key rather than by position. Files whose bytes or records can be read in any order are called random access files. Two methods are used for specifying where to start reading. In the first method, every read operation gives the position in the file to start reading at. In second one, a special operation, SEEK, is provided to set the current position. After a SEEK, the file can be read sequentially from the now-current position. In most modern operating systems, there is no distinction between random access and sequential access files.

5.1.5 File Attributes

In addition to the name of a file, the operating system associates other information with each file, for example, the date and time of creation and file size. This extra information is referred to as **file attributes**. The list of attributes varies from one system to another. The table below shows some attributes, which not all may be present in a single operating system.

Field	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 has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Figure 5.1 – File Attributes

The first four relate to file protection and tell who may access it and who may not. In some systems, the user must supply a password to access a file, in which case the password must be one of the attributes. Flags are bits or short fields that control some specific property. Hidden files, for example, do not appear in the listing of all files. The archive flag keeps track of whether a file has been backed up. Temporary flag allows a file to be marked for automatic deletion when the process that created it terminates.

5.1.5 File Operations

Files store information and allow it to be retrieved later on. Different systems provide different operations to allow storage and retrieval. Below are the most common system calls relating to files:

1. **CREATE** – Creates a file with no data. The purpose of this is to announce that a new file is being created and set some of the attributes.
2. **DELETE** – Deletes a file when it is no longer needed to free some disk space.
3. **OPEN** – Used to open a file before it can be used by a process. The purpose is to allow the operating system fetch the attributes and list of disk addresses into memory for rapid access later on.
4. **CLOSE** – When a process is through with a file, it should be closed to free up internal table space. A disk is written in blocks and closing a file forces writing of the file's last block even though the block may not be full.

5. **READ** – Data are read from a file. The caller must specify the size of data and provide a buffer to put them in.
6. **WRITE** – Data is usually written to the file at the current position. If the current position is at the end of the file, the size is increased. If it is in the middle, existing data are overwritten and lost forever.
7. **APPEND** – This is used to add data at the end of a file. Some systems do not provide for this since it can be derived from other system calls.
8. **SEEK –** For random access files, a method is needed to specify from where to read. The SEEK system call repositions the pointer from the current position to a specific position in the file. After this, data can be read from the current position.
9. **GET ATTRIBUTES –** Processes often need to read the file attributes to do their work. This system call is used to return specific attributes requested for.
10. **SET ATTRIBUTES –** Some of the file attributes are user settable and can be changed after the file has been created. This system call makes that possible.
11. **RENAME –** Often, users will want to change the name of an existing file. This system call is used for that, but it is not necessary since a file can usually be copied to a new file with the new name and then the old one deleted.

5.2 Directories

To keep track of files, file systems normally have directories in which in many systems are themselves files. A directory contains a number of entries, one per file. Directories normally have a hierarchical structure. There are two possible arrangements: one in which the directory has a file name and corresponding attributes in a table form or one in which we have a file name and a pointer to a data structure for attributes. This is shown in figure 5.2 below:

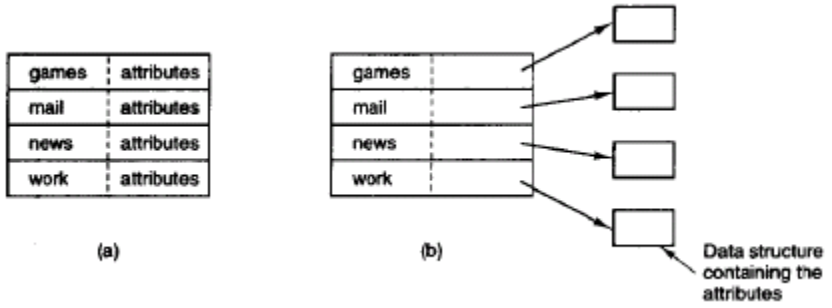
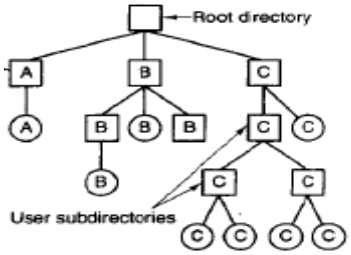


Figure 5.2

To open a file, the operating system normally searches a file in the directory until if finds it, then extracts its attributes and disk address either directly from the directory or from the data structure pointed to, and put them in main memory. All subsequent references use the information in the main memory.

What is needed is a general hierarchy (a tree of directories). With this approach, each user can have many directories as needed so that files can be grouped together in natural ways as shown below:



Directories A, B and C are contained in the root directory, each belonging to a different user. Two of them have created subdirectories.

When the file system is organized as a directory tree, some way is needed for specifying file names. In the first method each file is given an **absolute path name**, consisting of the path from the root directory. For example */usr/ast/mailbox* mean that the root directory contains a subdirectory *usr*, which in turn contains a subdirectory *ast*, which contains the file *mailbox*. Absolute path names always start from the root and are unique.

The other kind of name is the **relative path name**. It is used in conjunction with the concept of the **working** (current) directory. A user can designate one directory as the current working directory, in which case all path names do not begin at the root but are taken relative to the working directory. For example, if the current working directory is */usr/ast/*, then the file whose absolute path name is */usr/ast/mailbox* can be referenced simply as *mailbox*. In most systems each process has its own working directory. Some programs need to access a specific file regardless of the current working directory. In such cases they normally use absolute path names.

5.3 Directory Operations

The allowed system calls for managing directories exhibit more variation from system to system than system calls for files.

a) **CREATE** – Used for creating a directory.
b) **DELETE** – Deletes a directory. Only an empty directory can be deleted.
c) **OPENDIR** – Directories can be read. For example to list all files in a directory, a listing program opens the directory to read out the names of all the files it contains.
d) **CLOSEDIR** – When a directory has been read, it should be closed to free up internal table space.
e) **READDIR** – Returns the next entry in an open directory. Formerly, it was possible to read directories using the normal READ system call, but this has the disadvantage of forcing the programmer to know and deal with the internal structure of the directories.
f) **RENAME** – Used to rename directories just like in files
g) **LINK** – Linking is a technique that is used to allow a file appear in more than one directory. This system call specifies an existing file and a path name and creates a link from the existing file to a path specified by the path.

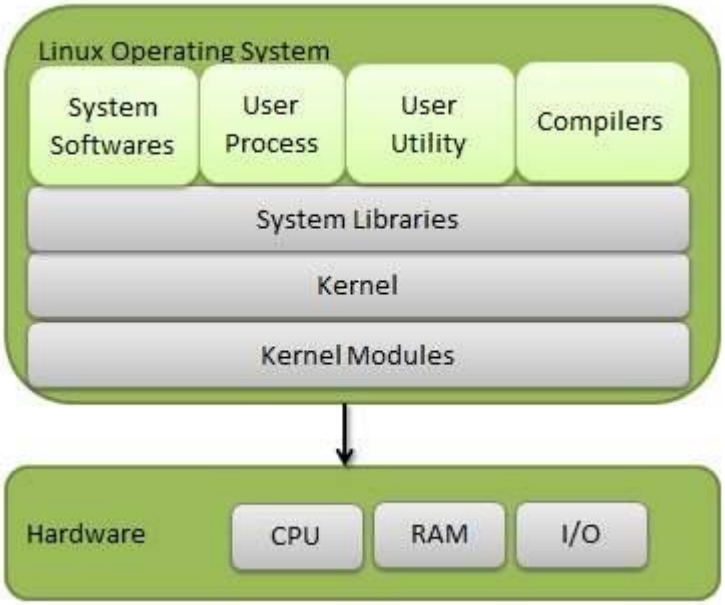
LINUX OPERATING SYSTEM CASE

Linux is one of popular version of UNIX operating System. It is open source as its source code is freely available. It is free to use. Linux was designed considering UNIX compatibility. Its functionality list is quite similar to that of UNIX.

Components of Linux System

Linux Operating System has primarily three components

- **Kernel** − Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.
- **System Library** − System libraries are special functions or programs using which application programs or system utilities accesses Kernel's features. These libraries implement most of the functionalities of the operating system and do not requires kernel module's code access rights.
- **System Utility** − System Utility programs are responsible to do specialized, individual level tasks.



Kernel Mode vs User Mode

Kernel component code executes in a special privileged mode called **kernel mode** with full access to all resources of the computer. This code represents a single process, executes in single address space and do not require any context switch and hence is very efficient and fast. Kernel runs each processes and provides system services to processes, provides protected access to hardware to processes.

Support code which is not required to run in kernel mode is in System Library. User programs and other system programs works in **User Mode** which has no access to system hardware and kernel code. User programs/ utilities use System libraries to access Kernel functions to get system's low level tasks.

Basic Features

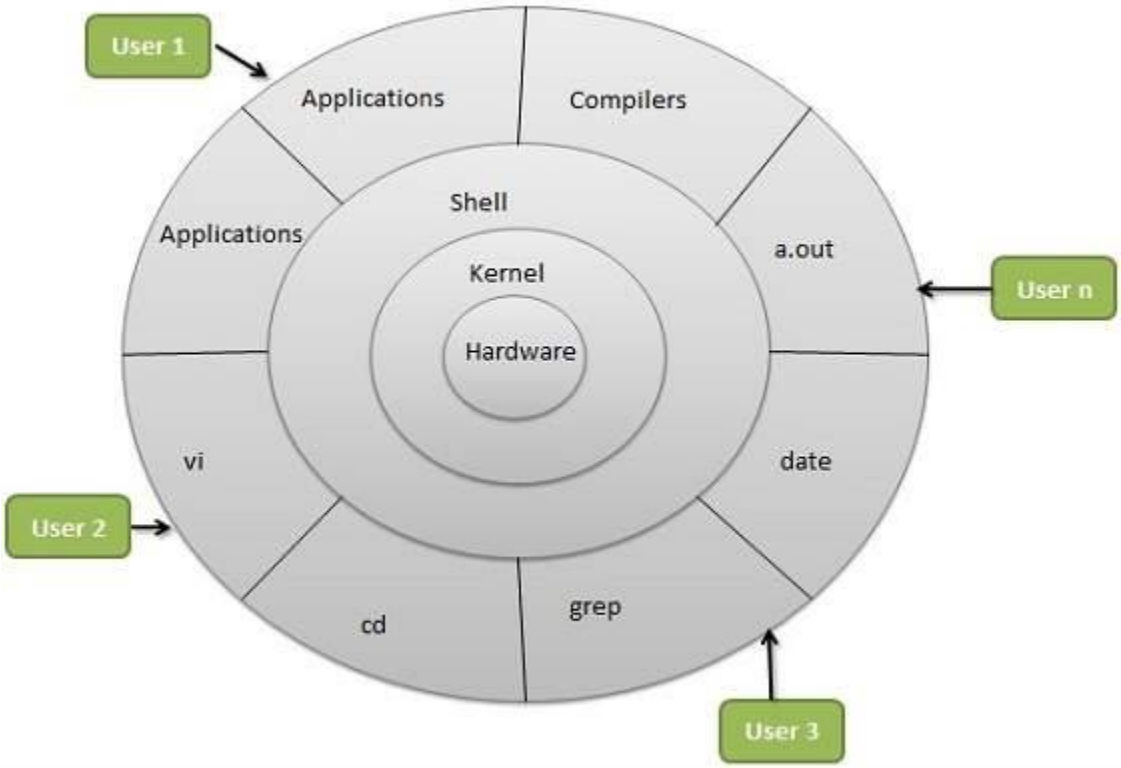
Following are some of the important features of Linux Operating System.

- **Portable** − Portability means software can works on different types of hardware in same way. Linux kernel and application programs supports their installation on any kind of hardware platform.
- **Open Source** − Linux source code is freely available and it is community based development project. Multiple teams work in collaboration to enhance the capability of Linux operating system and it is continuously evolving.

- **Multi-User** − Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.
- **Multiprogramming** − Linux is a multiprogramming system means multiple applications can run at same time.
- **Hierarchical File System** − Linux provides a standard file structure in which system files/ user files are arranged.
- **Shell** − Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs. etc.
- **Security** − Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

Architecture

The following illustration shows the architecture of a Linux system −



The architecture of a Linux System consists of the following layers −

- **Hardware layer** − Hardware consists of all peripheral devices (RAM/ HDD/ CPU etc).
- **Kernel** − It is the core component of Operating System, interacts directly with hardware, provides low level services to upper layer components.
- **Shell** − An interface to kernel, hiding complexity of kernel's functions from users. The shell takes commands from the user and executes kernel's functions.
- **Utilities** − Utility programs that provide the user most of the functionalities of an operating systems.