VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

WINTER SEMESTER 2021

CSE2005

OPERATING SYSTEMS

SLOT: B1

FACULTY: DR.RENUKADEVI S

24 APRIL 2021

PRADUSH J N

19BCE1521

ASSIGNMENT 2

Linux File System

A Linux file system is a structured collection of files on a disk drive or a partition. A partition is a segment of memory and contains some specific data. In our machine, there can be various partitions of the memory. Generally, every partition contains a file system.

The general-purpose computer system needs to store data systematically so that we can easily access the files in less time. It stores the data on hard disks (HDD) or some equivalent storage type. There may be below reasons for maintaining the file system:

- Primarily the computer saves data to the RAM storage; it may lose the data if it gets turned off. However, there is non-volatile RAM (Flash RAM and SSD) that is available to maintain the data after the power interruption.
- Data storage is preferred on hard drives as compared to standard RAM as RAM costs more than disk space. The hard disks costs are dropping gradually comparatively the RAM.

The Linux file system contains the following sections:

- The root directory (/)
- A specific data storage format (EXT3, EXT4, BTRFS, XFS and so on)
- A partition or logical volume having a particular file system.

What is the Linux File System?

Linux file system is generally a built-in layer of a Linux operating system used to handle the data management of the storage. It helps to arrange the file on the disk storage. It manages the file name, file size, creation date, and much more information about a file.

If we have an unsupported file format in our file system, we can download software to deal with it.

Linux File System Structure

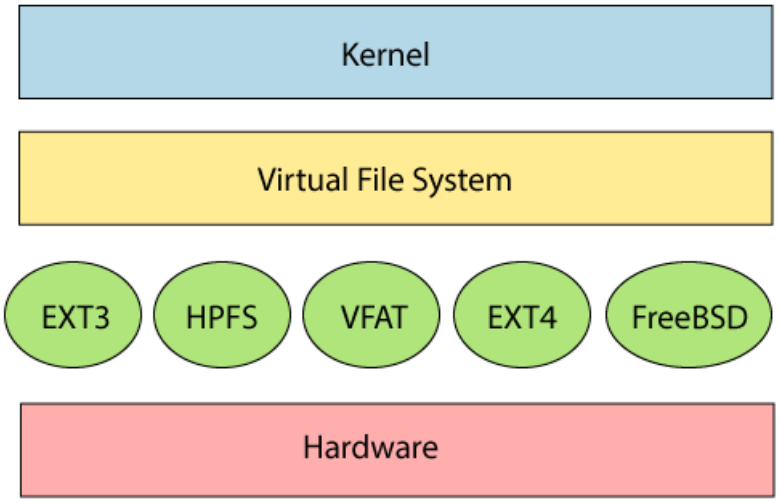
Linux file system has a hierarchal file structure as it contains a root directory and its subdirectories. All other directories can be accessed from the root directory. A partition usually has only one file system, but it may have more than one file system.

A file system is designed in a way so that it can manage and provide space for non-volatile storage data. All file systems required a namespace that is a naming and organizational methodology. The namespace defines the naming process, length of the file name, or a subset of characters that can be used for the file name. It also defines the logical structure of files on a memory segment, such as the use of directories for organizing the specific files. Once a namespace is described, a Metadata description must be defined for that particular file.

The data structure needs to support a hierarchical directory structure; this structure is used to describe the available and used disk space for a particular block. It also has the other details about the files such as file size, date & time of creation, update, and last modified.

Also, it stores advanced information about the section of the disk, such as partitions and volumes.

The advanced data and the structures that it represents contain the information about the file system stored on the drive; it is distinct and independent of the file system metadata.



The file system requires an API (Application programming interface) to access the function calls to interact with file system components like files and directories. API facilitates tasks such as creating, deleting, and copying the files. It facilitates an algorithm that defines the arrangement of files on a file system.

The first two parts of the given file system together called a **Linux virtual file system**. It provides a single set of commands for the kernel and developers to access the file system. This virtual file system requires the specific system driver to give an interface to the file system.

Linux File System Features

In Linux, the file system creates a tree structure. All the files are arranged as a tree and its branches. The topmost directory called the **root (/) directory**. All other directories in Linux can be accessed from the root directory.

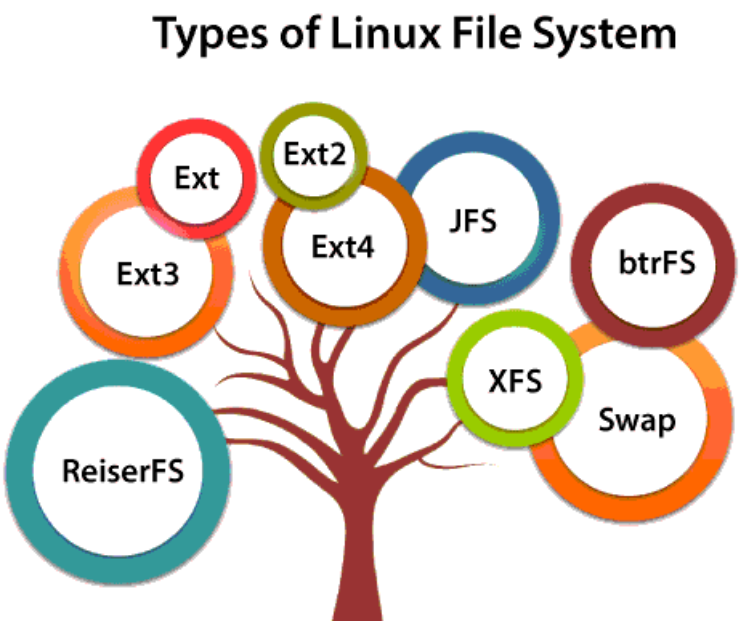
Some key features of Linux file system are as following:

- **Specifying paths:** Linux does not use the backslash (\) to separate the components; it uses forward slash (/) as an alternative. For example, as in Windows, the data may be stored in C:\ My Documents\ Work, whereas, in Linux, it would be stored in /home/ My Document/ Work.
- **Partition, Directories, and Drives:** Linux does not use drive letters to organize the drive as Windows does. In Linux, we cannot tell whether we are addressing a partition, a network device, or an "ordinary" directory and a Drive.
- **Case Sensitivity:** Linux file system is case sensitive. It distinguishes between lowercase and uppercase file names. Such as, there is a difference between test.txt and Test.txt in Linux. This rule is also applied for directories and Linux commands.
- **File Extensions:** In Linux, a file may have the extension '.txt,' but it is not necessary that a file should have a file extension. While working with Shell, it creates some problems for the beginners to differentiate between files and directories. If we use the graphical file manager, it symbolizes the files and folders.

- **Hidden files:** Linux distinguishes between standard files and hidden files, mostly the configuration files are hidden in Linux OS. Usually, we don't need to access or read the hidden files. The hidden files in Linux are represented by a dot (.) before the file name (e.g., .ignore). To access the files, we need to change the view in the file manager or need to use a specific command in the shell.

Types of Linux File System

When we install the Linux operating system, Linux offers many file systems such as **Ext, Ext2, Ext3, Ext4, JFS, ReiserFS, XFS, btrfs,** and **swap**.



Let's understand each of these file systems in detail:

1. Ext, Ext2, Ext3 and Ext4 file system

The file system Ext stands for **Extended File System**. It was primarily developed for **MINIX OS**. The Ext file system is an older version, and is no longer used due to some limitations.

Ext2 is the first Linux file system that allows managing two terabytes of data. Ext3 is developed through Ext2; it is an upgraded version of Ext2 and contains backward compatibility. The major drawback of Ext3 is that it does not support servers because this file system does not support file recovery and disk snapshot.

Ext4 file system is the faster file system among all the Ext file systems. It is a very compatible option for the SSD (solid-state drive) disks, and it is the default file system in Linux distribution.

2. JFS File System

JFS stands for **Journaled File System**, and it is developed by **IBM for AIX Unix**. It is an alternative to the Ext file system. It can also be used in place of Ext4, where stability is needed with few resources. It is a handy file system when CPU power is limited.

3. ReiserFS File System

ReiserFS is an alternative to the Ext3 file system. It has improved performance and advanced features. In the earlier time, the ReiserFS was used as the default file system in SUSE Linux, but later it has changed some policies, so SUSE returned to Ext3. This file system dynamically supports the file extension, but it has some drawbacks in performance.

4. XFS File System

XFS file system was considered as high-speed JFS, which is developed for parallel I/O processing. NASA still using this file system with its high storage server (300+ Terabyte server).

5. Btrfs File System

Btrfs stands for the **B tree file system**. It is used for fault tolerance, repair system, fun administration, extensive storage configuration, and more. It is not a good suit for the production system.

6. Swap File System

The swap file system is used for memory paging in Linux operating system during the system hibernation. A system that never goes in hibernate state is required to have swap space equal to its RAM size.

Basic filesystem functions

Disk storage is a necessity that brings with it some interesting and inescapable details. Obviously, a filesystem is designed to provide space for non-volatile storage of data; that is its ultimate function. However, there are many other important functions that flow from that requirement.

All filesystems need to provide a namespace—that is, a naming and organizational methodology. This defines how a file can be named, specifically the length of a filename and the subset of characters that can be used for filenames out of the total set of characters available. It also defines the logical structure of the data on a disk, such as the use of directories for organizing files instead of just lumping them all together in a single, huge conglomeration of files.

Once the namespace has been defined, a metadata structure is necessary to provide the logical foundation for that namespace. This includes the data structures required to support a hierarchical directory structure; structures to determine which blocks of space on the disk are used and which are available; structures that allow for maintaining the names of the files and directories; information about the files such as their size and times they were created, modified or last accessed; and the location or locations of the data belonging to the file on the disk. Other metadata is used to store high-level information about the subdivisions of the disk, such as logical volumes and partitions. This higher-level metadata and the structures it represents contain the information describing the filesystem stored on the drive or partition, but is separate from and independent of the filesystem metadata.

Filesystems also require an Application Programming Interface (API) that provides access to system function calls which manipulate filesystem objects like files and directories. APIs provide for tasks

such as creating, moving, and deleting files. It also provides algorithms that determine things like where a file is placed on a filesystem. Such algorithms may account for objectives such as speed or minimizing disk fragmentation.

Modern filesystems also provide a security model, which is a scheme for defining access rights to files and directories. The Linux filesystem security model helps to ensure that users only have access to their own files and not those of others or the operating system itself.

The final building block is the software required to implement all of these functions. Linux uses a two-part software implementation as a way to improve both system and programmer efficiency.

Directory structure

As a usually very organized Virgo, I like things stored in smaller, organized groups rather than in one big bucket. The use of directories helps me to be able to store and then locate the files I want when I am looking for them. Directories are also known as folders because they can be thought of as folders in which files are kept in a sort of physical desktop analogy.

In Linux and many other operating systems, directories can be structured in a tree-like hierarchy. The Linux directory structure is well defined and documented in the Linux Filesystem Hierarchy Standard (FHS). Referencing those directories when accessing them is accomplished by using the sequentially deeper directory names connected by forward slashes (/) such as /var/log and /var/spool/mail. These are called paths.

The following table provides a very brief list of the standard, well-known, and defined top-level Linux directories and their purposes.

Directory	Description
/ (root filesystem)	The root filesystem is the top-level directory of the filesystem. It must contain all of the files required to boot the Linux system before other filesystems are mounted. It must include all of the required executables and libraries required to boot the remaining filesystems. After the system is booted, all other filesystems are mounted on standard, well-defined mount points as subdirectories of the root filesystem.
/bin	The /bin directory contains user executable files.
/boot	Contains the static bootloader and kernel executable and configuration files required to boot a Linux computer.
/dev	This directory contains the device files for every hardware device attached to the system. These are not device drivers, rather they are files that represent each device on the computer and facilitate access to those devices.
/etc	Contains the local system configuration files for the host computer.
/home	Home directory storage for user files. Each user has a subdirectory in /home.
/lib	Contains shared library files that are required to boot the system.
/media	A place to mount external removable media devices such as USB thumb drives that may be connected to the host.

Directory	Description
/mnt	A temporary mountpoint for regular filesystems (as in not removable media) that can be used while the administrator is repairing or working on a filesystem.
/opt	Optional files such as vendor supplied application programs should be located here.
/root	This is not the root (/) filesystem. It is the home directory for the root user.
/sbin	System binary files. These are executables used for system administration.
/tmp	Temporary directory. Used by the operating system and many programs to store temporary files. Users may also store files here temporarily. Note that files stored here may be deleted at any time without prior notice.
/usr	These are shareable, read-only files, including executable binaries and libraries, man files, and other types of documentation.
/var	Variable data files are stored here. This can include things like log files, MySQL, and other database files, web server data files, email inboxes, and much more.

Table 1: The top level of the Linux filesystem hierarchy.

The directories and their subdirectories shown in Table 1, along with their subdirectories, that have a teal background are considered an integral part of the root filesystem. That is, they cannot be created as a separate filesystem and mounted at startup time. This is because they (specifically, their contents) must be present at boot time in order for the system to boot properly.

The /media and /mnt directories are part of the root filesystem, but they should never contain any data. Rather, they are simply temporary mount points.

The remaining directories, those that have no background color in Table 1 do not need to be present during the boot sequence, but will be mounted later, during the startup sequence that prepares the host to perform useful work.

Be sure to refer to the official Linux Filesystem Hierarchy Standard (FHS) web page for details about each of these directories and their many subdirectories. Wikipedia also has a good description of the FHS. This standard should be followed as closely as possible to ensure operational and functional consistency. Regardless of the filesystem types used on a host, this hierarchical directory structure is the same.

Journaling file system

A journaling file system is a file system that keeps track of changes not yet committed to the file system's main part by recording the goal of such changes in a data structure known as a "journal", which is usually a circular log. In the event of a system crash or power failure, such file systems can be brought back online more quickly with a lower likelihood of becoming corrupted.

Depending on the actual implementation, a journaling file system may only keep track of stored metadata, resulting in improved performance at the expense of increased possibility for data corruption. Alternatively, a journaling file system may track both stored data and related metadata, while some implementations allow selectable behavior in this regard.

Physical journals

A *physical journal* logs an advance copy of every block that will later be written to the main file system. If there is a crash when the main file system is being written to, the write can simply be replayed to completion when the file system is next mounted. If there is a crash when the write is being logged to the journal, the partial write will have a missing or mismatched checksum and can be ignored at next mount.

Physical journals impose a significant performance penalty because every changed block must be committed *twice* to storage, but may be acceptable when absolute fault protection is required.

Logical journals

A *logical journal* stores only changes to file metadata in the journal, and trades fault tolerance for substantially better write performance.[7] A file system with a logical journal still recovers quickly after a crash, but may allow unjournaled file data and journaled metadata to fall out of sync with each other, causing data corruption.

For example, appending to a file may involve three separate writes to:

1. The file's inode, to note in the file's metadata that its size has increased.
2. The free space map, to mark out an allocation of space for the to-be-appended data.
3. The newly allocated space, to actually write the appended data.

In a metadata-only journal, step 3 would not be logged. If step 3 was not done, but steps 1 and 2 are replayed during recovery, the file will be appended with garbage.

Write hazards

The write cache in most operating systems sorts its writes (using the elevator algorithm or some similar scheme) to maximize throughput. To avoid an out-of-order write hazard with a metadata-only journal, writes for file data must be sorted so that they are committed to storage before their associated metadata. This can be tricky to implement because it requires coordination within the operating system kernel between the file system driver and write cache. An out-of-order write hazard can also occur if a device cannot write blocks immediately to its underlying storage, that is, it cannot flush its write-cache to disk due to deferred write being enabled.

To complicate matters, many mass storage devices have their own write caches, in which they may aggressively reorder writes for better performance. (This is particularly common on magnetic hard drives, which have large seek latencies that can be minimized with elevator sorting.) Some journaling file systems conservatively assume such write-reordering always takes place, and sacrifice performance for correctness by forcing the device to flush its cache at certain points in the journal (called barriers in ext3 and ext4).