NEPAL COLLEGE OF INFORMATION TECHNOLOGY

BALKUMARI, LALITPUR

A REPORT

ON

Subject: Applied Operating System

Title: Case Study on Windows and Linux Systems

Submitted By:

Birat Aryal – 221614

Dipesh D.C. – 221714

Submitted To:

Amit K Srivastava Sir

WINDOWS

DESIGN PRINCIPLES

Windows was designed to be a versatile, user-friendly, and powerful operating system. The following principles guide its architecture and functionality:

1. Compatibility

- Windows supports a **vast range** of hardware and software, ensuring that older applications and devices can still function on newer versions.
- Windows system ensures that both hardware drivers and software libraries are **frequently updated**, reducing the need for hardware upgrades.
- **Example:** Running softwares like older versions of Microsoft Office on Windows 11.

2. Portability

- The operating system is written using **high-level languages** like C and C++ to make it easier to adapt to different hardware platforms.
- This makes Windows usable across devices such as PCs, laptops, tablets, and servers.
- One of the ways Windows achieves portability is through the use of a **Hardware Abstraction Layer (HAL)**.

- The HAL is a layer of code that abstracts **(provide abstraction)** the details of the underlying hardware, providing a uniform interface for the rest of the operating system. (**Hides the complex details** of how the hardware works.)
- **Example:** Windows running on Intel-based CPUs, ARM processors, or even virtual machines.

3. Extensibility

- Windows allows developers to enhance its functionality by adding new features without rewriting the system.
- It ensures that the OS can evolve over time with minimal disruption.
- **Example:** Adding custom device drivers for specialized hardware. (Installing a printer)

4. Performance

- Windows is optimized for speed and efficiency. It ensures fast program execution, quick responses, and proper resource allocation.
- Improves user experience, especially for multitasking, designing, editing and gaming.
- **Techniques used:**
    - Multithreading: Running multiple tasks in parallel.
    - Caching: Storing frequently used data for faster access.

5. Security

- Security is built into **every layer** of Windows to protect against threats like viruses, malware, and unauthorized access.

- **Key Features:**
  - User Account control (UAC) to prevent unauthorized changes.
  - Windows Defender for real-time protection.
- Protects sensitive data and ensures system integrity.

6. Multitasking

- Windows supports **preemptive multitasking**, where the OS decides which task gets CPU time and for how long.
- Allows users to run multiple programs simultaneously without crashes.
- **Example:** Watching a video while downloading files and editing a document.

7. User-Centered Design

- Windows features a **Graphical User Interface (GUI)** with easy-to-use components like the Start menu, taskbar, and system tray.
- Enhances accessibility and reduces the learning curve for new users.
- **Example:** Drag-and-drop functionality for moving files etc.

8. Networking

- Windows is designed for seamless networking, from local area networks (LANs) to the internet.
- **Key Features:**
  - File and printer sharing.
  - Remote desktop and cloud integration.
- Makes Windows ideal for both personal use and enterprise environments.

PROGRAMMER INTERFACE

The programmer interface in Windows provides a rich set of tools, frameworks, and libraries to help developers **create applications** and **interact** with the **operating system**.

1. Win32 API (Core Programming Interface)

- The **Win32 API** is the backbone of Windows programming.
- It gives developers **direct access** to low **level system features** like file handling, memory management, process control, and graphical user interfaces.
- Enables **low-level system programming** and **fine control over system resources.**
- Allows developers to write highly efficient and customized code that **interacts directly** with the hardware and the operating system.

2. Microsoft .NET Framework

- **Microsoft .NET Framework** is a software development platform designed to simplify the process of building applications for Windows.
- The .NET Framework simplifies application development with a **managed runtime environment** and **extensive libraries**.
- **Note:** Speeds up development by reducing the need for low-level coding.

- **Languages supported:** C#, VB.NET, and F#.
- **Example features:**
    - Windows Forms for building desktop applications.
    - ASP.NET for web applications.

3. UWP (Universal Windows Platform)

- UWP allows developers to **build applications** that **work seamlessly** across all Windows devices, including PCs, tablets, Xbox and many other devices.
- **Note:** Ensures a single app can adapt to various screen sizes and input methods (mouse, touch, or pen).
- **Example:** OneNote is a prime example of how UWP **enables** cross-device compatibility, allowing a single app to work consistently across PCs, tablets, and other devices.

4. PowerShell and Command-Line Tools

- PowerShell scripts is a command-line shell and scripting language developed by Microsoft that allow developers to **automate tasks** and **manage the system efficiently**.
- **Note:** Increases productivity, especially for system administrators and developers managing servers.
- **Examples:** A system administrator can write a PowerShell script that automatically backs up important files on a server at scheduled intervals

5. COM (Component Object Model)

- The **Component Object Model (COM)** is a framework developed by Microsoft that allows **software components** to interact with each other, regardless of the language in which they are written.
- **Note:** Allows integration of different programs and systems, making them work together.
- **Examples:**
    - ActiveX controls in web browsers.
    - Microsoft Office add-ins.

6. Driver Development

- Developers can write device drivers using the **Windows Driver Development Kit (DDK)**.
- **Note:** Ensures compatibility between the OS and new hardware devices.
- **Example:** Writing a driver for a custom printer or USB device.

7. Windows Subsystem for Linux (WSL)

- WSL allows developers to run Linux distributions and tools natively within Windows.
- **Note:** Bridges the gap between Linux and Windows environments, making it easier for developers who work across both systems.
- **Example:** Running a Linux shell script directly in Windows.

SYSTEM COMPONENTS

A **system component** refers to the part of software or hardware that **works together** with the other parts to make a computer or operating system function properly.

These components are essential building blocks of the system, and each one performs a specific role to help the system run smoothly.

1 .User-Mode Components

User-mode components **directly interact with users**, providing the visual interface and access to resources and applications:

1. **Desktop**
2. **Taskbar**
3. **Start Menu**
4. **Icons**
5. **File Explorer**
6. **Control Panel**
7. **Task Manager**

2. Kernel-Mode Components

Kernel-mode components are parts of the operating system that run at a **low level**, very close to the hardware.

They have **complete control** over the system and can **directly interact** with the computer's hardware (like the CPU, memory, and devices)

Kernel-mode components operate at a **deeper level** within the operating system, managing system resources and hardware interactions.

I/O Manager: handles communication between the operating system and external devices

Memory Manager: Responsible for allocating and deallocating memory to running processes

Process Manager: Oversees process creation, scheduling, and termination

Device Drivers: enable Windows to communicate with hardware components like printers, graphics cards, and network adapters

3. System Management Components

System Management Components are tools and utilities in an operating system that help administrators and users **manage**, **configure**, and **monitor** various aspects of the system.

Event Viewer

Microsoft Management Console (MMC)

PowerShell

Server Manager

Task Scheduler

FILE SYSTEMS

The file system in Windows is a critical part of the operating system, which is responsible for how data **is organized, stored, and retrieved** from **storage devices** like hard drives, USB drives, and external storage.

1. NTFS (New Technology File System)

- NTFS is the **default file system** for modern Windows versions. It's designed for reliability, security, and large storage capacities.
- **Key Features:**
  - **File Compression:** Saves disk space by compressing files.
  - **Encryption (EFS):** Protects sensitive files from unauthorized access.
  - **Permissions:** Controls who can access or modify files and folders.
- **Note:** It's ideal for modern needs like large hard drives, security, and reliability.

2. FAT32 (File Allocation Table)

- FAT32 is an older file system mainly used for USB drives and portable storage devices.
- **Key Features:**
  - Compatible with many operating systems.
  - Limited file size: Maximum file size is 4 GB, and maximum partition size is 32 GB.
- **Note:** Ensures **cross-platform compatibility**.

- **Example:** Used for USB drives that need to work on both Windows and macOS.

3. exFAT (Extended File Allocation Table)

- **exFAT (Extended File Allocation Table)** is a file system developed by Microsoft, designed to handle large files and improve compatibility across different devices and operating systems.
- It **bridges the gap** between NTFS and FAT32.
- **Key Features:**
  - **Supports** large files (over 4 GB), unlike FAT32.
  - **Lightweight** and **faster** than NTFS for portable devices.
- **Note:** Ideal for modern high-capacity external storage devices.
- **Example:** Commonly used for SD cards and USB drives.

4. ReFS (Resilient File System)

- **Overview:** ReFS is a modern file system optimized for reliability, scalability, and fault tolerance.
- **Key Features:**
  - Data integrity checks to prevent corruption.
  - Supports large volumes and files.
  - Integration with storage pools for data redundancy.
- **Note:** Primarily used in enterprise environments like data centers.
- **Example:** Ideal for storage servers and virtual machines.

5. HPFS (High-Performance File System)

A **High-Performance File System (HPFS)** is a system designed to **store** and **access data** quickly and efficiently.

It is used in situations where fast and reliable data handling is important, such as in servers, data centers, or scientific computing.

- **Fast Access**: Retrieves and saves files quickly, even for large files.
- **Handles Large Data**: Works well with big files and many files at once.
- **Reliable**: Reduces errors and ensures data is safe.
- **Efficient Use of Space**: Organizes data neatly to use storage effectively.

Note: HPFS is all about speed, reliability, and handling large amounts of information smoothly.

File System Comparison Table

File System	Compatibility	Key Features	Use Cases
FAT32	High	Simple, widely compatible	USB drives, memory cards
exFAT	Moderate	Optimized for flash drives, cross-platform	External flash drives, SD cards
NTFS	High	Security, journaling, disk quotas	Internal drives, secure environments
ReFS	Limited (Server)	Data integrity, automatic data repair	High-availability servers
HPFS	Obsolete	Legacy support only	Older systems using Windows NT or OS/2

WINDOWS SECURITY

Security is a critical aspect of the Windows operating system, ensuring user data, applications, and system resources are protected from **threats** like **malware**, **unauthorized access**, and system vulnerabilities.

1. User Account Control (UAC)

- UAC prevents unauthorized changes to the system by requiring administrative approval.
- **How it works:** Prompts the user for **permission** or an **admin password** when critical changes are attempted.
- **Note:** It helps prevent malware from gaining administrative access and making system changes without the user's knowledge.
- **Example:** When installing new software, UAC will ask for permission to proceed.

2. Windows Defender Antivirus

- Built-in antivirus software that provides real-time protection against viruses, malware, and spyware.
- **Features:**
    - Automatic updates to recognize new threats.
    - Scanning and frequently checks for infected files.
- **Note:** It offers comprehensive protection without needing third-party antivirus software, making it an effective and built-in security solution.

3. Firewall Protection

- The **Windows Defender Firewall** is a key component of the Windows security system, designed to protect your computer from unauthorized access, particularly from external threats over a network.
- The **Windows Defender Firewall** monitors and controls incoming and outgoing network traffic.
- **How it works:**
    - Blocks unauthorized access to the system.
    - Allows trusted applications to communicate with the internet.
- **Note:** Protects the system from network-based attacks, such as hackers trying to access data remotely.

4. BitLocker Drive Encryption

- BitLocker encrypts entire drives, making data inaccessible without the proper credentials.
- **Features:**
    - Protects against data theft if a device is lost or stolen.
    - Supports encryption of external drives
- **Note:** Ensures data security efficiently by protecting sensitive data and preventing unauthorized access.

5. Windows Hello

- A biometric authentication system that allows users to log in with facial recognition, fingerprints, or a PIN.

- **Note:** Provides a secure and convenient alternative to passwords.
- **Example:** Logging into a Windows device with facial recognition.

6. Secure Boot

- Secure Boot ensures that only trusted software can load during the boot process.
- **How it works:** Verifies the digital signatures of boot components.
- **Note:** Protects against rootkits and other boot-level malware.

7. Windows Security App

- A centralized app for managing all security settings, including antivirus, firewall, and device health.
- **Features:**
  - Security status dashboard.
  - Recommendations for improving system security.
- **Note:** Provides an easy way for users to monitor and adjust their system security.

8. Credential Guard

- Protects sensitive information like user credentials from being accessed by malicious software.
- **How it works:** Stores credentials in an isolated environment.

- **Note:** Prevents credential theft, especially in enterprise environments.

9. Windows Sandbox

- A lightweight, isolated environment for running untrusted applications safely.
- **How it works:** Any changes made in the sandbox are discarded when it's closed.
- **Note:** Prevents untrusted software from affecting the main system.
- **Example:** Testing suspicious programs without risking the main OS.

10. Access Controls

- Windows enforces permissions and access levels for files, folders, and system resources.
- **Features:**
  - File permissions (read, write, execute).
  - Role-based access control for enterprise users.
- **Note:** Prevents unauthorized access to sensitive data.

LINUX

What is Linux?

Linux is a free and open-source family of operating systems that is resilient and flexible. In 1991, an individual by the name as Linus Torvalds constructed it. The system's source code is accessible to everyone for anyone to look at and change, making it cool that anyone can see how the system works. People from all across the world are urged to work together and keep developing Linux due to its openness.

Since the beginning, Linux has grown into a dependable and safe OS that is used in an array of gadgets, including PCs, cell phones, and huge supercomputers. It is well-known for being cost-effective, which implies that employing it doesn't cost a lot, and efficient, which indicates it can complete a lot of jobs quickly.

What is Linux Operating System?

Developed by Linus Torvalds in 1991, the Linux operating system is a powerful and flexible open-source software platform. It acts as the basis for a variety of devices, such embedded systems, cell phones, servers, and personal computers. Linux, that's well-known for its reliability, safety, and flexibility, allows users to customize and improve their environment to suit specific needs. With an extensive and active community supporting it, Linux is an appealing choice for people as well as companies due to its wealth of resources and constant developments.

What is a “distribution?”

Linux distribution is an operating system that is made up of a collection of software based on Linux kernel or you can say distribution contains the Linux kernel and supporting libraries and software. And you can get Linux-based operating system by downloading one of the Linux distributions and these distributions are available for different types of devices like embedded devices, personal computers, etc. Around 600 + Linux Distributions are available and some of the popular Linux distributions are:

- MX Linux
- Manjaro
- Linux Mint
- elementary
- Ubuntu
- Debian
- Solus
- Fedora
- openSUSE
- Deepin

Design Principles

- Linux is a multiuser, multitasking system with a full set of UNIX-compatible tools.
- Its file system adheres to traditional UNIX semantics, and it fully implements the standard UNIX networking model.
- Main design goals are speed, efficiency, and standardization.
- Linux is designed to be compliant with the relevant POSIX documents; at least two Linux distributions have achieved official POSIX certification.
- The Linux programming interface adheres to the SVR4 UNIX semantics, rather than to BSD behavior.

Components of Linux System

The components of a Linux system are :

1. Kernel: The kernel is responsible for maintaining all the important abstractions of the

operating system such as virtual memory and processes.

2. System libraries: The system libraries define a standard set of functions through which

applications can interact with the kernel. These functions implement much of the operating-

system functionality that does not need the full privileges of kernel code. The most important

system library is the C library, known as libc.

3. System utilities: The system utilities are programs that perform individual specialized

management tasks. Some system utilities are invoked just once to initialize and configure

some aspect of the system. Others —known as daemons in UNIX terminology— run

permanently, handling such tasks as responding to incoming network connections, accepting

logon requests from terminals, and updating log files

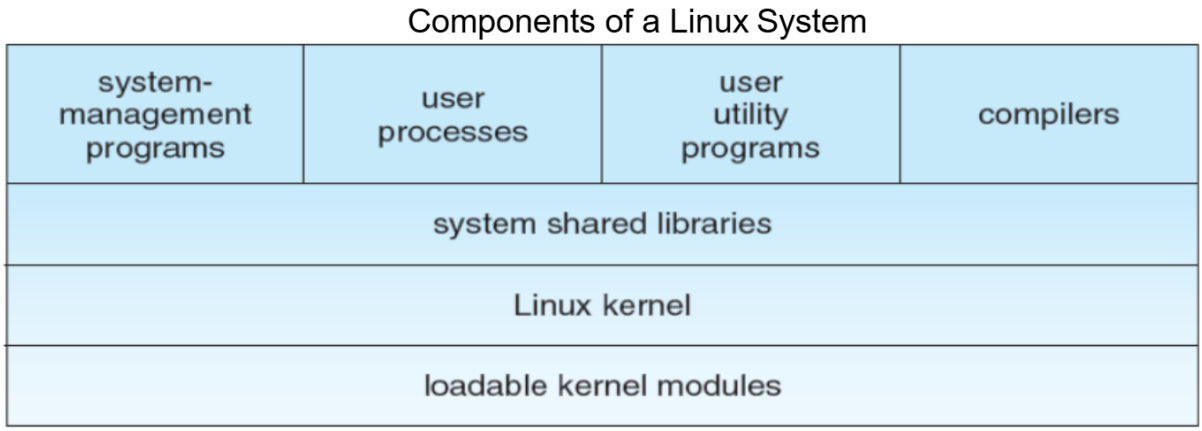


Figure 1: Components of a Linux System

Inter-process Communication

Inter-process Communication (IPC) refers to a mechanism, where the operating systems allow various process to communicate with each other. This involves synchronizing their actions and managing shared data. For inter-process communication LINUX has three main components:

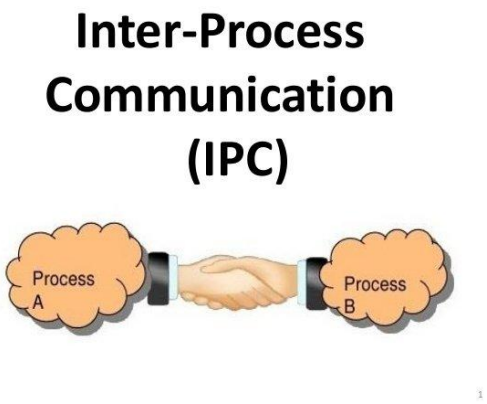


Figure 2: IPC

1. **Module Management:**
   For new modules this is done at two levels- the management of kernel references symbols and the management of the code in kernel memory. The LINUX kernel maintains a symbol table and symbols defined here can be exported explicitly. The module management system also defines all the required communication interfaces for newly inserted module. With this done, process can request the services from this module.

2. **Driver Management:**
   Usually, the registration of drivers is maintained in a registration table of the module. The registration of drivers contains the following:
   - Driver context identification as a bulk device or network driver.
   - File system context to store files in LINUX virtual file system or network file system like NFS

- Network protocols and packet filtering rules
- File formats for executable and other files

3. **Conflict Resolution:**
   The PC hardware configuration is supported by a large number of chip set configurations and with a large range of drivers for SCSI devices, video display devices and adapters, network cards. This results in the situation where we have module device drivers which vary over a very wide range of capabilities and options. This necessitates a conflict resolution mechanism to resolve accesses in a variety of conflicting concurrent accesses. The conflict resolution mechanisms help in preventing modules from having an access conflict.

Process Management:

Multitasking is the illusion created by the kernel. The processor is capable of performing the processes in parallel by switching in between multiple tasks that are running on the system. The context switching is done in parallel rapidly and repeatedly in specific intervals. The user is not able to notice the switching of the tasks due to small intervals. Linux process can be visualized as running instance of a program. For example, just open a text editor on your Linux box and a text editor process will be born.

The following are some of the tasks of process management that are handled by the kernel.

All the processes should work independently without interfering each other until they desired to interact. The Linux environment is a multiuser system. Even though multiple processes are running concurrently, the problem occurred in one application will not affect some other application. All these applications/programs that are running in parallel will not be able to access the memory content of the other

application/program. This feature provides the security for the application's data from the other users.

All the processes of the system will fairly utilize CPU time and share in between multiple applications. The applications are assigned their priorities based on which the order of the execution is determined.

The execution time allocated to each process (time quantum) and when to context switch between processes should take place is decided by the kernel. This begs the question as to which process is actually the next. Decisions on time slot allocation and context switching are not platform-dependent.

While context switching by the kernel, the kernel will ensure that the execution environment of a process brought back is exactly the same when it last withdrew processor resources. For example, the contents of the processor registers and the structure of virtual address space must be identical. This latter task is extremely dependent on processor type. How CPU time is allocated is determined by the scheduler policy which is totally separate from the task switching mechanism needed to switch between processes.

Process Priorities: Not all processes are of equal importance. In addition to process priority, there are different criticality classes to satisfy differing demands. In general processes can be split into real-time processes and non-real-time processes.

Real-time processes are also of two types i.e., hard real-time and delicate real-time or soft real-time processes. Hard real-time processes are subject to strict time limits during which certain tasks must be completed. If the flight control commands of an aircraft are processed by computer, they must be forwarded as quickly as possible—within a guaranteed period of time. The key characteristic of hard real-time processes is that they must be processed within a guaranteed time frame. Note that this does not imply that the time frame is particularly short. Instead, the system must guarantee that a certain time frame is never exceeded, even when unlikely or adverse conditions prevail.

In soft real time systems, the meeting of deadline is not compulsory for every time for every task but process should get processed and give the result. Even the soft real time systems cannot miss the deadline for every task or process according to the priority it should meet the deadline or can miss the deadline. If system is missing the deadline for every time the performance of the system will be worse and cannot be used by the users. Best example for soft real time system is personal computer, audio and video systems, etc.. An illustration of a delicate real-time process is a compose activity to a CD. Information should be procured by the CD author at a specific rate since information are kept in touch with the medium in a nonstop stream. On the off chance that framework stacking is too high, the information stream might be intruded on quickly, and this may bring about an unusable CD, far less exceptional than a plane accident. All things considered, the compose process should consistently be conceded CPU time when required — before any remaining typical processes.

Linux does not support hard real-time processing, at least not in the vanilla kernel. There are, however, modified versions such as RTLinux, Xenomai, or RATI that offer this feature. The Linux kernel runs as a separate "process" in these approaches and handles less important software, while real-time work is done outside the kernel. The kernel may run only if no real-time critical actions are performed. Since Linux is optimized for throughput and tries to handle common cases as fast as possible, guaranteed response times are very hard to achieve. Nevertheless, quite a bit of progress has been made during last years to decrease the overall kernel latency, i.e., the time that elapses between making a request and its fulfillment.

Process Heirarchy

Each process is identified by a unique positive integer called the process ID (pid). The pid of the first process is 1, and each subsequent process receives a new, unique pid.

In Linux, processes form a strict hierarchy, known as the process tree. The process tree is rooted at the first process, known as the init process, which is typically the init program. New processes are created via the fork() system call. This system call creates a duplicate of the calling process. The original process is called the parent; the new process is called the child. Every process except the first has a parent. If a parent

process terminates before its child, the kernel will reparent the child to the init process.

When a process terminates, it is not immediately removed from the system. Instead, the kernel keeps parts of the process resident in memory to allow the process's parent to inquire about its status upon terminating. This inquiry is known as waiting on the terminated process. Once the parent process has waited on its terminated child, the child is fully destroyed. A process that has terminated, but has not yet been waited upon, is called a zombie. The init process routinely waits on all of its children, ensuring that re-parented processes do not remain zombies forever.

Linux Process States

Processes are born, share resources with parents for some time, get their own copy of resources when they are ready to make changes, go through various states depending upon their priority and then finally die. Following are the various states of Linux processes:

- RUNNING – This state specifies that the process is either in execution or waiting to get executed.

- INTERRUPTIBLE – This state specifies that the process is waiting to get interrupted as it is in sleep mode and waiting for some action to happen that can wake this process up. The action can be a hardware interrupt, signal etc. UN-INTERRUPTIBLE - It is just like the INTERRUPTIBLE state, the only difference being that a process in this state cannot be wake up by delivering a signal.

- STOPPED - This state specifies that the process has been stopped. This may happen if a signal like SIGSTOP, SIGTTIN, etc. is delivered to the process.

- ZOMBIE-This state specifies that the process is terminated but still hanging around in kernel process table because the parent of this process has still not fetched the termination status of this process. Parent uses wait() family of functions to fetch the termination status.

- DEAD - This state specifies that the process is terminated and entry is removed from process table. This state is achieved when the parent successfully fetches the termination status as explained in ZOMBIE state.

Linux Threads Vs Light Weight Processes

Threads in Linux are nothing but a flow of execution of the process. A process containing multiple execution flows is known as multi-threaded process. For a non-multi-threaded process there is only execution flow that is the main execution flow and hence it is also known as single threaded process.

Threads are often mixed with the term Light Weight Processes (LWPs). The reason dates back to those times when Linux supported threads at user level only. This means that even a multi-threaded application was viewed by kernel as a single process only. This posed big challenges for the library that managed these user level threads because it had to take care of cases that a thread execution did not hinder if any other thread issued a blocking call. Later on, the implementation changed and processes were attached to each thread so that kernel can take care of them. Linux kernel does not see them as threads; each thread is viewed as a process inside kernel. These processes are known as light weight processes.

The main difference between a LWP and a normal process is that LWPs share same address space and other resources like open files etc. As some resources are shared so these processes are considered to be light weight as compared to other normal processes and hence the name light weight processes.

So, effectively we can say that threads and light weight processes are same. It's just that thread is a term that is used at user level while light weight process is a term used at kernel level.

From implementation point of view, threads are created using functions exposed by POSIX compliant pthread library in Linux. Internally, the clone() function is used to create a normal as well as a light weight process. This means that to create a normal process fork() is used that further calls clone() with appropriate arguments while to create a thread or LWP, a function from pthread library calls clone() with relevant flags. So, the main difference is generated by using different flags that can be passed to clone() function.

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.

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.

Directory Structure

The directories help us to store the files and locate them when we need them. Also, directories are called folders as they can be assumed of as folders where files reside in the form of a physical desktop analogy. Directories can be organized in a tree-like hierarchy in Linux and several other operating systems.

The directory structure of Linux is well-documented and defined in the Linux FHS (Filesystem Hierarchy Standard). Referencing those directories if accessing them is achieved via the sequentially deeper names of the directory linked by '/' forward slash like /var/spool/mail and /var/log. These are known as paths.

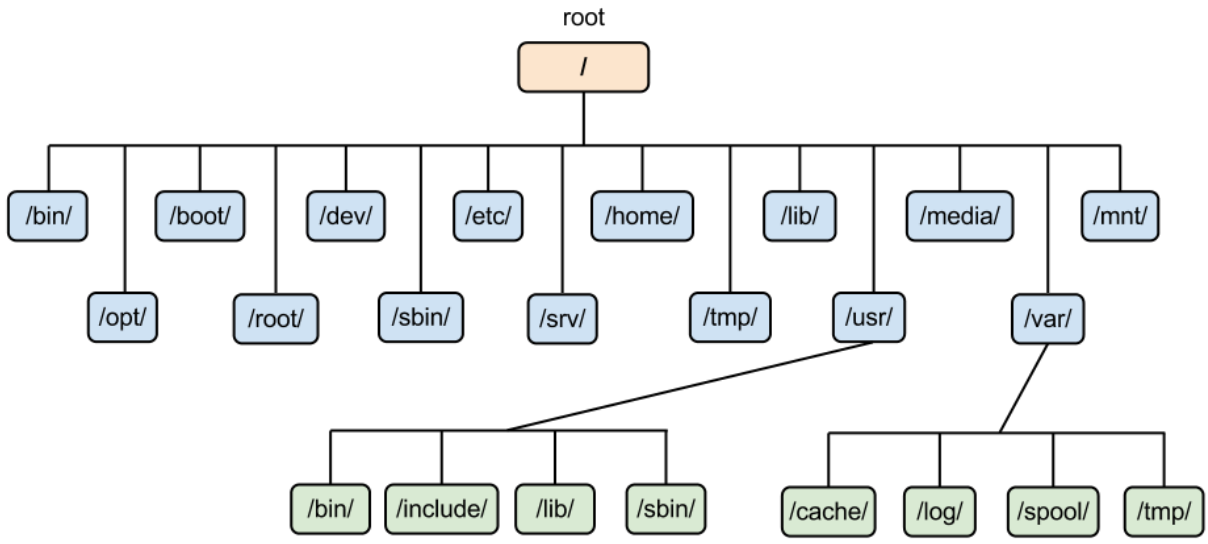


Figure 3: Directory Structure

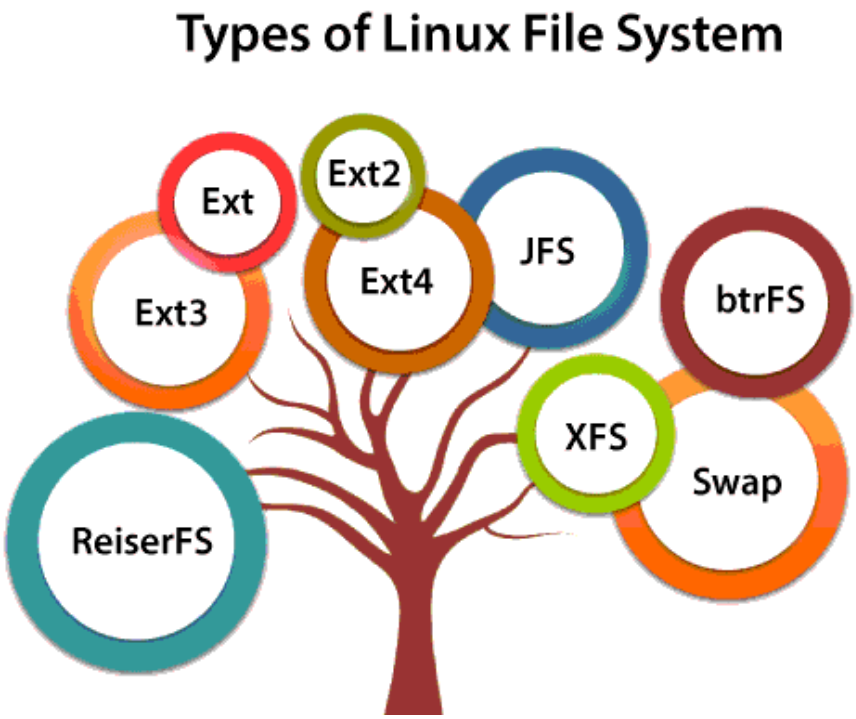
The below table gives a very short standard, defined, and well-known top-level Linux directory list and their purposes:

- **/ (root filesystem):** It is the top-level filesystem directory. It must include every file needed to boot the Linux system before another filesystem is mounted. Every other filesystem is mounted on a well-defined and standard mount point because of the root filesystem directories after the system is started.
- **/boot:** It includes the static kernel and bootloader configuration and executable files needed to start a Linux computer.
- **/bin:** This directory includes user executable files.
- **/dev:** It includes the device file for all hardware devices connected to the system. These aren't device drivers; instead, they are files that indicate all devices on the system and provide access to these devices.
- **/etc:** It includes the local system configuration files for the host system.
- **/lib:** It includes shared library files that are needed to start the system.

- **/home:** The home directory storage is available for user files. All users have a subdirectory inside /home.
- **/mnt:** It is a temporary mount point for basic filesystems that can be used at the time when the administrator is working or repairing a filesystem.
- **/media:** A place for mounting external removable media devices like USB thumb drives that might be linked to the host.
- **/opt:** It contains optional files like vendor supplied application programs that must be placed here.
- **/root:** It's the home directory for a root user. Keep in mind that it's not the '/' (root) file system.
- **/tmp:** It is a temporary directory used by the OS and several programs for storing temporary files. Also, users may temporarily store files here. Remember that files may be removed without prior notice at any time in this directory.
- **/sbin:** These are system binary files. They are executables utilized for system administration.
- **/usr:** They are read-only and shareable files, including executable libraries and binaries, man files, and several documentation types.
- **/var:** Here, variable data files are saved. It can contain things such as MySQL, log files, other database files, email inboxes, web server data files, and much more.

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**.



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.

Kernel Modules

The Linux kernel has the ability to load and unload arbitrary sections of kernel code on demand. These loadable kernel modules run in privileged kernel mode and as a consequence have full access to all the hardware capabilities of the machine on which they run.

A kernel module can implement a device driver, a file system, or a networking protocol.

The kernel's module interface allows third parties to write and distribute, on their own terms, device drivers or file systems that could not be distributed under the GPL. Kernel modules allow a Linux system to be set up with a standard minimal kernel, without any extra device drivers built in.

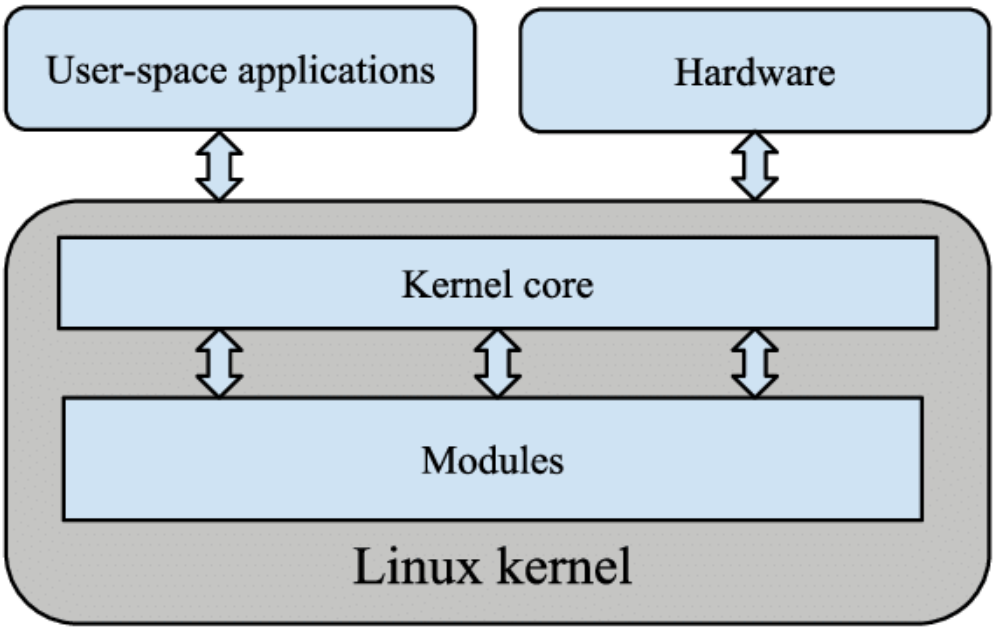


Figure 4: Kernel Modules

Any device drivers that the user needs can be either loaded explicitly by the system at startup or loaded automatically by the system on demand and unloaded when not in use.

Example:

A mouse driver can be loaded when a USB mouse is plugged into the system and unloaded when the mouse is unplugged.

The module support under Linux has **four components:**

a) The module-management system allows modules to be loaded into memory and to communicate with the rest of the kernel.
b) The module loader and unloader, which are user-mode utilities, work with the module-management system to load a module into memory.
c) The driver-registration system allows modules to tell the rest of the kernel that a
new driver has become available.
d) A conflict-resolution mechanism allows different device drivers to reserve hardware resources and to protect those resources from accidental use by another driver.

Module management

Loading a module requires more than just loading its binary contents into kernel memory. The system must also make sure that any references the module makes to kernel symbols or entry points are updated to point to the correct locations in the kernel's address space.

Module loading is split into two separate sections:

- Managing sections of module code in kernel memory
- Handling symbols that modules are allowed to reference

The loading of the module is performed in two stages.

- First, the module loader utility asks the kernel to reserve a continuous area of virtual kernel memory for the module.
- A second system call then passes the module, plus any symbol table that the new
  module wants to export, to the kernel.

The module itself is now copied verbatim into the previously allocated space, and the kernel's symbol table is updated with the new symbols for possible use by other modules not yet loaded.

The final module-management component is the module requester. The kernel defines a communication interface to which a module-management program can connect.

Driver Registration

The kernel maintains dynamic tables of all known drivers and provides a set of routines to

allow drivers to be added to or removed from these tables at any time.

The kernel makes sure that it calls a module's startup routine when that module is loaded and calls the module's cleanup routine before that module is unloaded. These routines are responsible for registering the module's functionality.

Registration tables include the following items:

- Device drivers
- File systems
- Network protocols
- Binary format

Conflict Resolution

Linux provides a central conflict-resolution mechanism to help arbitrate access to certain hardware resources.

It includes:

- To prevent modules from clashing over access to hardware resources
- To prevent autoprobes—device-driver probes that auto-detect device configuration—
  from interfering with existing device drivers
- To resolve conflicts among multiple drivers trying to access the same hardware.

The kernel maintains lists of allocated hardware resources. When any device driver wants to access a resource, it is expected to reserve the resource with the kernel database first. This requirement incidentally allows the system administrator to determine exactly which resources have been allocated by which driver at any given point.

Network Structure

Networking is a key area of functionality for Linux.

- It supports the standard Internet protocols for UNIX to UNIX communications.
- It also implements protocols native to non-UNIX operating systems, in particular, protocols used on PC networks, such as AppleTalk and IPX.

Internally, networking in the Linux kernel is implemented by three layers of software:

- The socket interface
- Protocol drivers
- Network device drivers

The most important set of protocols in the Linux networking system is the internet protocol suite.

- It implements routing between different hosts anywhere on the network.
- On top of the routing protocol are built the UDP, TCP and ICMP protocols

Security

The pluggable authentication modules (PAM) system is available under Linux.

PAM is based on a shared library that can be used by any system component that needs to authenticate users.

Access control under UNIX systems, including Linux, is performed through the use of unique numeric identifiers (uid and gid).

Access control is performed by assigning objects a protections mask, which specifies which access modes—read, write, or execute—are to be granted to processes with owner, group, or world access.

Linux augments the standard UNIX setuid mechanism in two ways:

- It implements the POSIX specification's saved user-id mechanism, which allows a process to repeatedly drop and reacquire its effective uid.
- It has added a process characteristic that grants just a subset of the rights of the effective uid.

Linux provides another mechanism that allows a client to selectively pass access to a single file to some server process without granting it any other privileges.