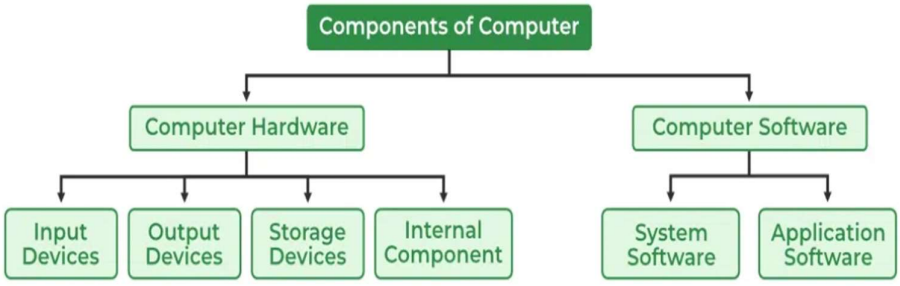
OPERATING SYSTEMS

UNIT – 1

A computer system is divided into two categories: Hardware and Software.

Hardware refers to the physical and visible components of the system such as a monitor, CPU, keyboard and mouse.

Software, on the other hand, refers to a set of instructions which enable the hardware to perform a specific set of tasks.



Types of Computer Software

- System Software
- Application Software

System Software

System software is essential for running and managing computer hardware. It includes operating systems, device drivers, and utility programs.

Features of System Software

- Written in a low-level language for better hardware interaction.
- Functions close to the system hardware.
- Executes faster than application software.
- Difficult to design and understand.
- Acts as an interface between hardware and application software.

Functions of System Software

- Memory Management: Allocates and deallocates memory dynamically.
- Processor Management: Manages CPU scheduling for efficient execution.
- File Management: Organizes, retrieves, and controls access to files.
- Security Management: Protects data and system resources from unauthorized access.

- Error Handling: Detects and resolves system errors.

Example of System Software:

Operating systems like Linux, Windows, and MacOS are examples of system software that manage hardware and provide a platform for applications.

Application Software

Application software is designed to perform specific user tasks, such as word processing, gaming, or web browsing.

Features of Application Software

- Written in high-level languages like Python, Java, or C++.
- Provides a user-friendly interface.
- Requires user interaction.
- Designed for end-users.

Functions of Application Software

- Data Management: Handles storage and processing of user data.
- Multimedia Processing: Includes graphics, animations, and video editing tools.
- Communication Tools: Facilitates email, messaging, and video conferencing.
- Industry-Specific Applications: Includes software for healthcare, finance, and education.

Example of Application Software:

Microsoft Word is an example of application software used for document editing and formatting.

Introduction to Operating Systems

An operating system (OS) is a fundamental component of a computer system that acts as an interface between the user and the hardware. It ensures efficient management of resources and provides essential services required for smooth execution of programs. Without an operating system, a computer would not be able to function effectively as there would be no systematic way to manage hardware and software interactions.

What is an Operating System?

An operating system is an interface between the user and the computer, facilitating communication and coordination between them.

Definition of an Operating System

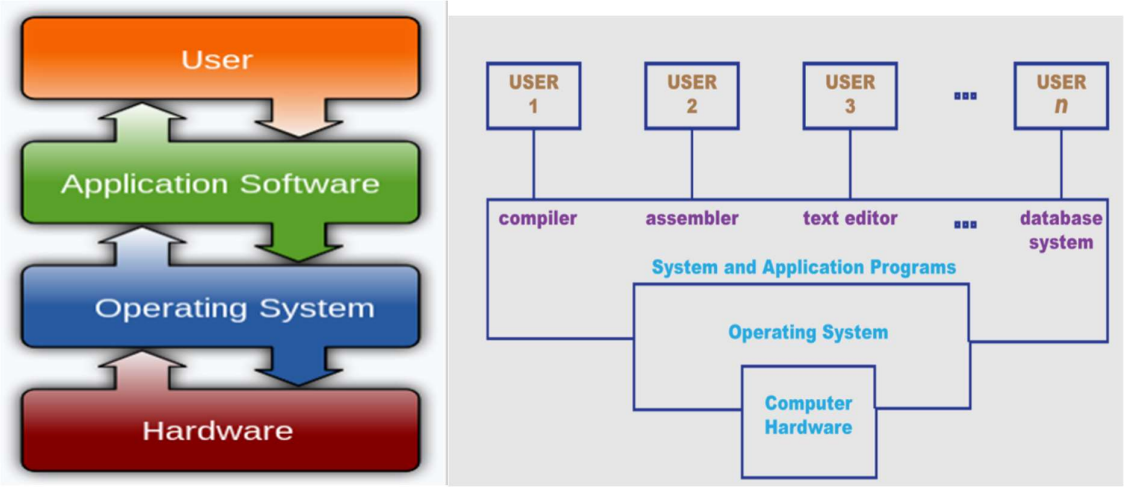
An Operating System (OS) is a collection of software that:

- Manages hardware resources
- Provides various services to the users

Goals of an Operating System

1. Execute user programs and make problem-solving easier – The OS provides an environment where user applications can run efficiently.
2. Enhance convenience – It ensures the computer system is user-friendly and simplifies interactions with hardware.
3. Optimize hardware utilization – The OS maximizes the efficiency of system resources such as CPU, memory, and storage.

For example, in a multitasking environment like Windows, multiple applications such as a web browser, a media player, and a document editor can run simultaneously without interference, thanks to the OS managing the system resources effectively.



Operating System Objectives

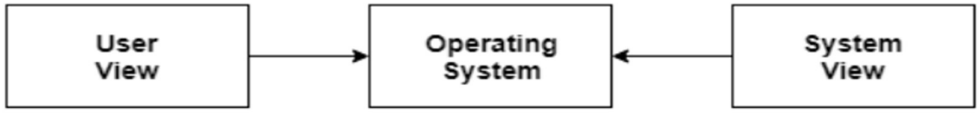
The OS is designed to achieve several objectives that ensure smooth and efficient system operations.

1. **Resource Management:** Manages CPU, memory, storage, and input/output devices to optimize system performance.
2. **Abstraction:** Hides hardware complexity, allowing users to interact with the system easily.
3. **Security and Protection:** Implements authentication and encryption to protect system resources.
4. **Concurrency:** Supports multitasking and multithreading for efficient process execution.
5. **User Interface:** Provides graphical (GUI) or command-line (CLI) interfaces for user interaction.

6. **Error Detection and Handling:** Detects faults in hardware and software and prevents failures.
7. **Efficiency:** Optimizes system resource utilization and minimizes response time.

Operating System Views

The operating system can be observed from the point of view of the user or the system. This is known as the user view and the system view respectively.



User View:

The user view depends on the system interface that is used by the users. The different types of user view experiences can be explained as follows −

- If the user is using a **personal computer**, the operating system is largely designed to make the interaction easy. Some attention is also paid to the performance of the system, but there is no need for the operating system to worry about resource utilization. This is because the personal computer uses all the resources available and there is no sharing.
- If the user is using a system connected to a **mainframe or a minicomputer**, the operating system is largely concerned with resource utilization. This is because there may be multiple terminals connected to the mainframe and the operating system makes sure that all the resources such as CPU, memory, I/O devices etc. are divided uniformly between them.
- If the user is sitting on a **workstation** connected to other workstations through networks, then the operating system needs to focus on both individual usage of resources and sharing though the network. This happens because the workstation exclusively uses its own resources but it also needs to share files etc. with other workstations across the network.
- If the user is using a **handheld computer** such as a mobile, then the operating system handles the usability of the device including a few remote operations. The battery level of the device is also taken into account.
- There are some devices that contain very less or no user view because there is no interaction with the users. Examples are embedded computers in home devices, automobiles etc.

System View:

From the System point of view the operating system is the program which is most intermediate with the hardware.

- The system views the operating system as a **resource allocator**. There are many resources such as CPU time, memory space, file storage space, I/O devices etc. that are required by processes for execution. It is the duty of the operating system to allocate these resources judiciously to the processes so that the computer system can run as smoothly as possible.
- The operating system can also work as a **control program**. It manages all the processes and I/O devices so that the computer system works smoothly and there are no errors. It makes sure that the I/O devices work in a proper manner without creating problems.

Operating System Services

Operating system services are fundamental functionalities provided by an operating system (OS) to support the execution of application software.

An Operating System provides services to both the users and to the programs. It provides programs, an environment to execute. It provides users, services to execute the programs in a convenient manner.

Following are the services provided by operating systems to both users and to the programs:

Users

- User Interface
- Program Execution
- I/O Operations
- File-System Manipulation
- Communication
- Error Detection

System/Programs

- Resource Allocation
- Accounting
- Protection and Security

1. User Interface

An interface allows users to interact with the system more easily and intuitively. Many operating systems provide user interfaces, such as command-line interfaces (CLI), graphical user interfaces (GUI) and Batch Based Interface.

A Command-Line Interface (CLI) is a text-based user interface used to interact with software and operating systems. Through a CLI, users can input text commands to perform specific tasks

A GUI usually consists of all the graphical icons displayed on a computer screen, visual indicators like widgets, texts, labels, and text navigation. Thus, a user can directly perform actions with a click of the mouse or keyboard.

A Batch-based interface in computing refers to a method of interaction between the user and the computer system or software where commands or a series of commands are prepared in advance and then executed all at once, rather than being entered and executed one at a time.

2. Program Execution

The OS loads a program into memory and then executes that program. It also makes sure that once started that program can end its execution, either normally or forcefully. The major steps during program management are:

- Loading a program into memory.
- Executing the program.
- Making sure the program completes its execution.

3. I/O Operations

I/O operations are required during the execution of a program. To maintain efficiency and protection of the program, users cannot directly govern the I/O devices instead the OS allows to

read or write operations with any file using the I/O devices and also allows access to any required I/O device when required.

4. File System Manipulation

A program is read and then written in the form of directories and files. These files can be stored on the storage disk for the long term.

Following are the major activities of an operating system with respect to file management.

- Program needs to read a file or write a file.
- The operating system gives the permission to the program for operation on file.
- Permission varies from read-only, read-write, denied and so on.
- Operating System provides an interface to the user to create/delete files.
- Operating System provides an interface to the user to create/delete directories.
- Operating System provides an interface to create the backup of file system.

5. Communication systems

Processes need to swap information among themselves. These processes can be from the same computer system or different computer systems as long as they are connected through communication lines in a network. This can be done with the help of OS support using shared memory or message passing. The OS also manages routing, connection strategies, and the problem of contention and security.

6. Error Detection

Errors may occur in any of the resources like CPU, I/O devices, or memory hardware. The OS keeps a lookout for such errors, corrects errors when they occur, and makes sure that the system works uninterruptedly.

7. Resource Allocation/Resource Management

Resource management in operating systems (OS) is a critical function that involves managing the hardware and software resources of a computer system. These resources include the CPU (central processing unit), memory (both RAM and storage), and networking components.

CPU /Process Management:

The OS is responsible for deciding which processes get to use the CPU when there are multiple processes needing to run. This is handled through CPU scheduling algorithms (e.g., Round-Robin, Shortest Job First, Priority Scheduling) that aim to optimize CPU utilization and ensure a fair and efficient allocation of CPU time to processes.

Memory Management:

Memory management involves allocating and managing the system's primary memory or RAM. The OS must track which parts of memory are in use and by whom, allocate memory to processes when they need it, and deallocate it when they are done. Techniques such as paging and segmentation are used to manage memory efficiently and protect the memory space of processes from each other.

Network Resource Management:

For systems connected to a network, the OS manages the network configuration and communication protocols to enable data exchange between computers. This includes managing network interfaces, routing, bandwidth allocation, and implementing security measures for network connections.

8. Accounting

This keeps a check of which resource is being used by a user and for how long it is being used. This is usually done for statistical purposes.

9. Protection and Security

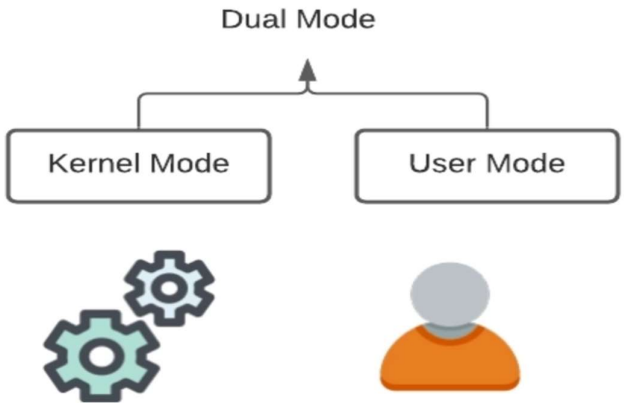
Considering a computer system having multiple users the concurrent execution of multiple processes, then the various processes must be protected from each another's activities. Protection refers to mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer system.

Following are the major activities of an operating system with respect to protection.

- OS ensures that all access to system resources is controlled.
- OS ensures that external I/O devices are protected from invalid access attempts.
- OS provides authentication feature for each user by means of a password

Operating System Operation

In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of operating-system code and user defined code. The approach taken by most computer systems is to provide hardware support that allows us to differentiate among various modes of execution. At the very least, we need two separate modes of operation: **user mode** and **kernel mode** (also called **supervisor mode**, **system mode**, or **privileged mode**)



User mode

When the computer system run user applications like creating a text document or using any application program, then the system is in the user mode. When the user application requests for a service from the operating system or an interrupt occurs or system call, then there will be a transition from user to kernel mode to fulfil the requests.

The mode bit is set to 1 in the user mode. It is changed from 1 to 0 when switching from user mode to kernel mode.

Kernel Mode

When the system boots, hardware starts in kernel mode and when operating system is loaded, it start user application in user mode. To provide protection to the hardware, we have privileged instructions which execute only in kernel mode. If user attempt to run privileged instruction in user mode then it will treat instruction as illegal and traps to OS.

Some of the privileged instructions are:

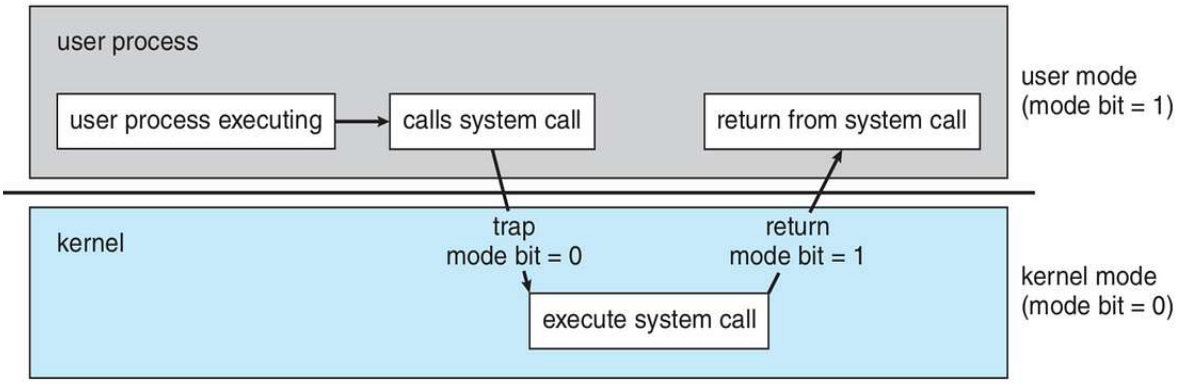
1. Handling Interrupts

2. To switch from user mode to kernel mode.

3. Input-Output management.

The mode bit is set to 0 in the kernel mode. It is changed from 0 to 1 when switching from kernel mode to user mode.

A diagram that illustrates the transition from user mode to kernel mode and back again is as follows



- At system boot time, the hardware starts in **kernel mode.** Thus, whenever the operating system gains control of the computer, it is in **kernel mode**.
- The operating system is then loaded and starts user applications in **user mode.** The system always **switches to user mode** (by setting the mode bit to 1) before passing control to a user program.
- When the computer system is executing on behalf of a user application, **the system is in user mode**. However, when a user application requests a service from the operating system (via a system call), it must **transition from user to kernel mode** to fulfil the request.

Or

- Whenever a trap (Traps are occurred by the user program to invoke the functionality of the OS. Assume the user application requires something to be printed on the screen, and it would set off a trap, and the operating system would write the data to the screen.) or interrupt occurs, the hardware switches from **user mode to kernel mode** (that is, changes the state of the mode bit to 0).

Timer

- At every point, we must ensure that the operating system maintains control over the CPU. We can't allow a user program to get stuck in an infinite loop or to fail to call system services and never return control to the operating system. Now to accomplish this goal, we can use

a **timer**. A timer can be set to interrupt the computer after a specified period. The period may be fixed (1/60 sec) or variable (1ms to 1s).

- A **variable timer** is generally implemented by a fixed-rate clock and a counter. The operating system sets the counter. Every time the clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs.

Operating System Structures

Operating System (OS) structure refers to the way an operating system is designed and organized to manage hardware and software resources, provide services to users and applications, and ensure efficient and secure operation of a computer system. There are several different approaches to operating system structure, each with its own advantages and disadvantages.

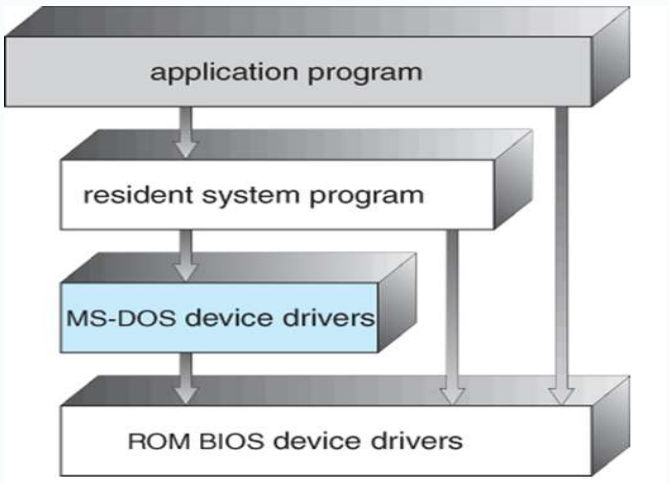
Different types of structures implementing Operating Systems as mentioned below.

1. Simple Structure
2. Layered Structure
3. Micro Kernel Structure
4. Modular Structure
5. Hybrid Structure

Simple Structure

It is the simplest Operating System Structure and is not well defined. It can only be used for small and limited systems. In this structure, the interfaces and levels of functionality are well separated, hence programs can access I/O routines which can cause unauthorized access to I/O routines.

This structure is implemented in MS-DOS operating system. The **MS-DOS operating System is made up of various layers, each with its own set of functions.**



Advantages of Simple Structure

- It is easy to develop because of the limited number of interfaces and layers.
- Offers good performance due to lesser layers between hardware and applications.
- Minimal overhead, suitable for resource-constrained environments.

Disadvantages of Simple Structure

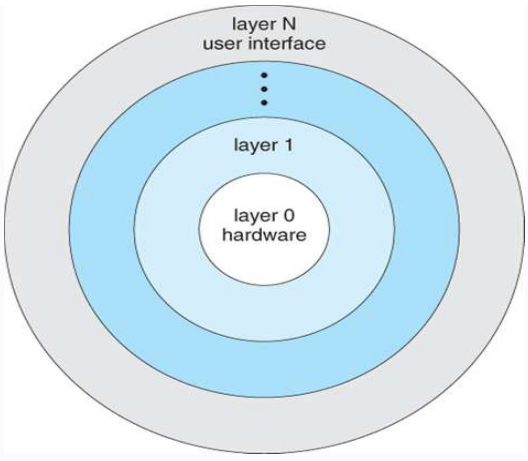
- If one user program fails, the entire operating system crashes.
- Limited functionality.
- Abstraction or data hiding is not present as layers are connected and communicate with each other.
- Layers can access the processes going in the Operating System, which can lead to data modification and can cause Operating System to crash.

Layered Structure

In this type of structure, OS is divided into layers or levels. The hardware is on the bottom layer (layer 0), while the user interface is on the top layer (layer N). These layers are arranged in a hierarchical way in which the top-level layers use the functionalities of their lower-level levels.**Example:Linux**

The following are some of the key characteristics of a layered operating system structure:

- Each layer is responsible for a specific set of tasks. This makes it easier to understand, develop, and maintain the operating system.
- Layers are typically arranged in a hierarchy. This means that each layer can only use the services provided by the layers below it.
- Layers are independent of each other. This means that a change to one layer should not affect the other layers.



Advantages of Layered Structure

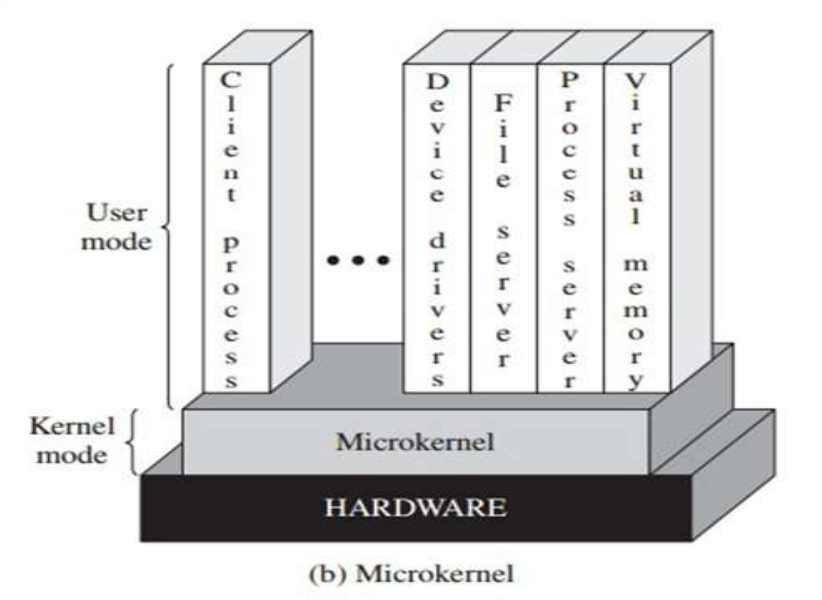
- A layered structure is highly modular, meaning that each layer is responsible for a specific set of tasks. This makes it easier to understand, develop, and maintain the operating system.
- Each layer has its functionalities, so work tasks are isolated, and abstraction is present up to some level.
- Debugging is easier as lower layers are debugged, and then upper layers are checked.

Disadvantages of Layered Structure

- In Layered Structure, layering causes degradation in performance.
- It takes careful planning to construct the layers since higher layers only utilize the functions of lower layers.
- There can be some performance overhead associated with the communication between layers. This is because each layer must pass data to the layer above it.

Micro-Kernel structure

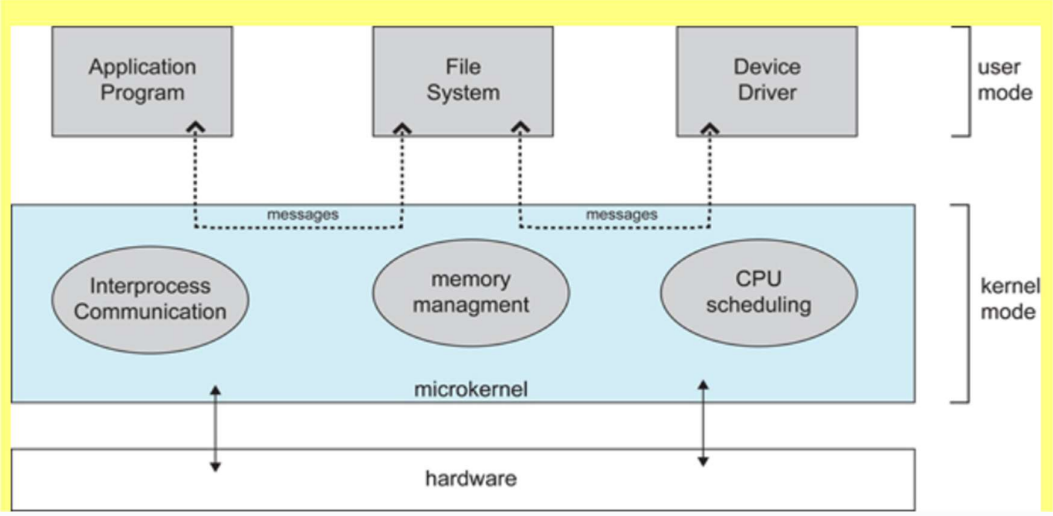
Kernel is the core part of an operating system that manages system resources. It also acts as a bridge between the application and hardware of the computer. It is one of the first programs loaded on start-up (after the Bootloader).



(b) Microkernel

A microkernel is a type of operating system kernel that is designed to provide only the most basic services required for an operating system to function, such as memory management and

process scheduling. Other services, such as device drivers and file systems, are implemented as user-level processes that communicate with the microkernel via message passing. **Ex: Mac OS**



Advantages of Micro-kernel structure:

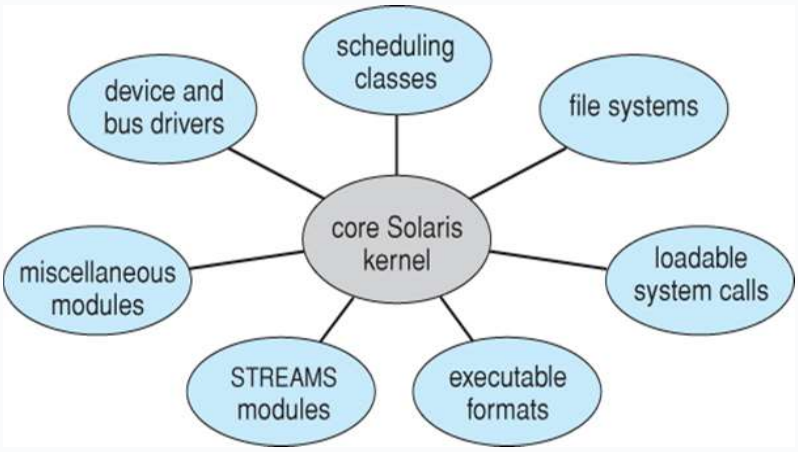
- It allows the operating system to be portable between platforms.
- Enhanced system stability and security.
- As each Micro-Kernel is isolated, it is safe and trustworthy.
- Because Micro-Kernels are smaller, they can be successfully tested.
- If any component or Micro-Kernel fails, the remaining operating System is unaffected and continues to function normally.

Disadvantages of Micro-kernel structure:

- Increased inter-module communication reduces system performance.
- System is complex to be constructed.
- Complexity in managing user-space components.

Modular Structure

In a modular operating system structure, the operating system is divided into a set of independent modules. Each module is responsible for a specific task, such as memory management, process scheduling, or device drivers. Modules can be loaded and unloaded dynamically, as needed**. EX: Solaris**



Advantages of Modular Structure

- A modular structure is highly modular, meaning that each module is independent of the others. This makes it easier to understand, develop, and maintain the operating system.
- A modular structure is very flexible. New modules can be added easily, and existing modules can be modified or removed without affecting the rest of the operating system.

Disadvantages of Modular Structure

- There can be some performance overhead associated with the communication between modules. This is because modules must communicate with each other through well-defined interfaces.
- A modular structure can be more complex than other types of operating system structures. This is because the modules must be carefully designed to ensure that they interact correctly

Hybrid Structure

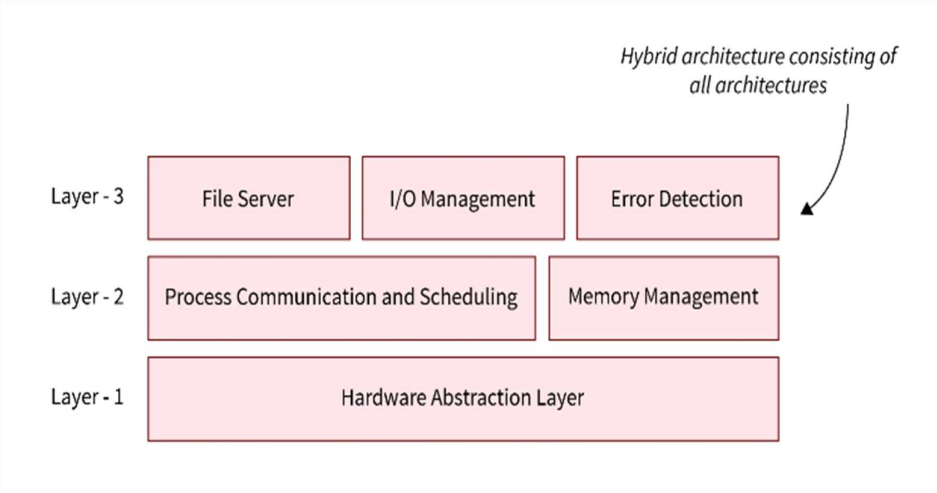
Hybrid Structure as the name suggests consists of a hybrid of all the Structure explained so far and hence it has properties of all of those architectures which makes it highly useful in present-day operating systems. The hybrid-Structure consists of three layers.

1) Hardware abstraction layer: It is the interface between the kernel and hardware and is present at the lowest level.

2) Microkernel Layer: This is the old microkernel that we know and it consists of CPU scheduling, memory management, and inter-process communication.

3) Application Layer: It acts as an interface between the user and the microkernel. It contains functionalities like a file server, error detection, I/O device management, etc.

Example: Microsoft Windows NT kernel implements a hybrid architecture of the operating system.



Advantages:

1. Since it is a hybrid of other structures it allows various structures to provide their services respectively.
2. It is easy to manage because it uses a layered approach.
3. The number of layers is relatively lesser.
4. Security and protection are relatively improved.

Disadvantage:

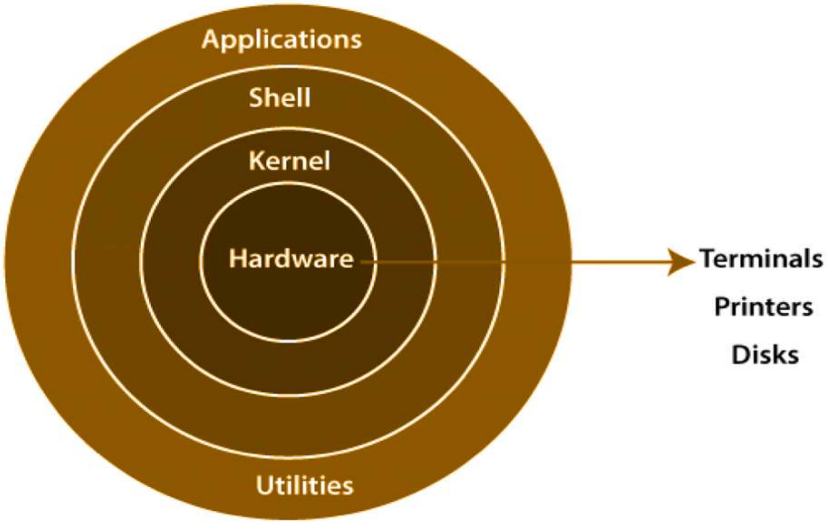
1. It increases overall complexity of system by implementing both structure (monolithic and micro) and making the system difficult to understand.

Linux Architecture

Linux is similar to other operating systems you may have used before, such as Windows, macOS (formerly OS X), or iOS. Like other operating systems, Linux has a graphical interface, and the same types of software you are accustomed to, such as word processors, photo editors, video editors, and so on.

Linux was created in 1991 by Linus Torvalds, a then-student at the University of Helsinki. Torvalds built Linux as a free and open-source alternative to Minix, another Unix clone that was predominantly used in academic settings. He originally intended to name it “Freax,” but later renamed it as “Linux” after a combination of Torvalds’ first name and the word Unix.

The diagram illustrates the structure of the Linux system, according to the layers concept.



The Linux architecture is largely composed of elements such as the Applications, Shell, Kernel, Hardware layer, System Utilities and Libraries.

Applications

Applications are the programs that the user runs on top of the architecture. The applications are the user space element that includes database applications, media players, web browsers, and presentations.

Shell

Shell is the interface that interacts with humans and processes the commands that are given for the execution. One can call it an interpreter because it takes the command from the keyboard and makes it understandable to the kernel.

Shells are categorized into two sections:

Command Line Shell

Graphical User Shell

The command line shell is the user interface where the user types commands in a text form. When the user provides the command in the terminal, the shell interprets the commands for the kernel. The shell also has some built-in commands that help the user to navigate, manage, and change the file system

The graphical user shell is the user interface using the system's peripheral components like a mouse, and keyboard. It is beneficial for users who are not familiar with commands. These shells are used to make the desktop environment easier.

Kernel

In Linux operating systems, the kernel is the core component that acts as the bridge between computer hardware and applications. It is responsible for managing the system's resources and allowing software and hardware to communicate with each other.

System Utilities and Libraries

The system utilities and libraries provide a wide range of functions to manage the system. Low-level hardware complexity to high-level user support is served by the system utilities and libraries.

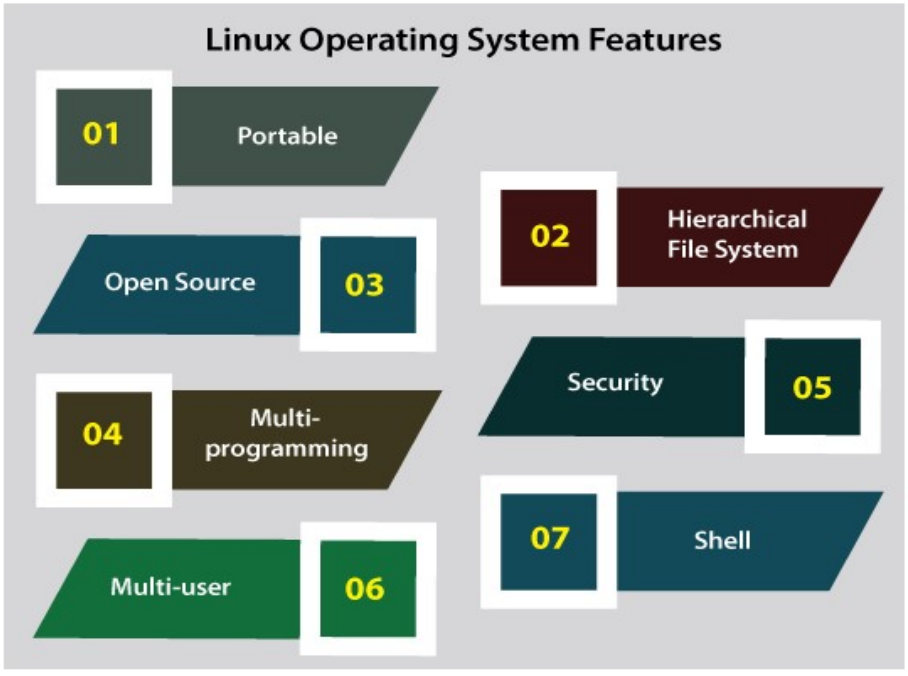
- **System Utilities:** It is the program that performs the task which is given by the users and manages the system. (Antivirus, Winrar, Winzip etc)
- **System Libraries:** Functions through which the system interacts with the kernel and handles all the functionalities without the kernel privileges

Hardware

Hardware layer of Linux is the lowest level of operating system track. It is plays a vital role in managing all the hardware components. It includes device drivers, kernel functions, memory management, CPU control, and I/O operations. This layer generalizes hard complexity, by providing an interface for software by assuring proper functionality of all the components.

Some Linux distributions are: MX Linux, Manjaro, Linux Mint, elementary, Ubuntu, Debian, Solus, Fedora, openSUSE, Deepin.

Linux Operating System Features



1. **Portable:** Linux OS can perform different types of hardware and the kernel of Linux supports the installation of any type of hardware environment.
2. **Hierarchical file system:** Linux OS affords a typical file structure where user files or system files are arranged.
3. **Open source:** Linux operating system source code is available freely and for enhancing the capability of the Linux OS, several teams are performing in collaboration.
4. **Multiprogramming:** Linux OS can be defined as a multiprogramming system. It means more than one application can be executed at the same time.
5. **Security:** Linux OS facilitates user security systems with the help of various features of authentication such as controlled access to specific files, password protection, or data encryption.
6. **Multi-user:** Linux OS can also be defined as a multi-user system. It means more than one user can use the resources of the system such as **application programs, memory,** or **RAM** at the same time.
7. **Shell:** Linux operating system facilitates a unique interpreter program. This type of program can be applied for executing commands of the operating system. It can be applied to perform various types of tasks such as call application programs and others.

Linux Commands

echo command

The **echo** command in Linux and Unix-like operating systems is a built-in command used primarily to display messages or output text to the terminal or into a file.

Syntax

echo [option] [string]

1.How to input a text to get the output using the echo command?

Input: echo “Linux Commands”

Output: -Linux Commands

2. How to print a variable?

x=100

Input: echo “The value of x =$x”

Output: The value of x = 100

The following tabular gives you the possible options in the echo command:

Options	Description
-e	Enable interpretation of backslash escape sequences
-E	Disable interpretation of backslash escape sequences
\a	Alert return
\n	To add a new line
\t	To add a Horizontal tab spaces
\v	To add a Vertical tab spaces
\c	To stop displaying the output from this stage
\r	Carriage Return
\\	To perform as Backslash ('\')
\b	To behave as the backspace
\f	Adding a Form Feed into your texts

3. How to add a new line?

Input: echo -e "Linux \n commands"

Output: Linux
Commands

Note: The Linux "echo" command provides the "-e" option, which instructs the command to interpret backslash escape sequences in the specified text. By utilizing the '\n' sequence, we can add a new line and print the subsequent text on a new line below the previous one. This enables us to format the output and make it more readable by separating it into distinct lines.

5. How to add a horizontal tab to your content?

echo -e "Linux \t commands"

Output: Linux commands

PATH

PATH is an Environment variable in Linux. Environment variables are dynamic values that affect the processes or programs on a computer. It essentially tells the shell which directories to search through to find the executable files (programs or scripts) that match the command names entered

by the user. This search process occurs in the order the directories are listed within the **PATH** variable.

To view the path (echo command)

$echo $PATH

Output

usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games

The **PATH** variable contains a list of directories separated by colons (**:**).When a command is entered, the shell searches for an executable file with the command's name starting from the first directory listed in the **PATH** variable and proceeds to the next if it's not found. This continues until either the executable is found or the list is exhausted. If the executable is not found in any of the directories listed in **PATH**, the shell typically returns a "command not found" error.

Adding a Directory to the PATH Environment Variable **(export Command)**

A directory can be added to PATH in two ways: at the start or the end of a path.

Adding a directory (/the/file/path for example) to the **start of PATH** will mean it is checked first:

export PATH=/the/file/path:$PATH

echo $PATH

Output:

the/file/path:usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games

Adding a directory to the **end of PATH** means it will be checked after all other directories:

export PATH=$PATH:/the/file/path

echo $PATH

Output:

usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games:/ the/file/path

Multiple directories can be added to PATH at once by adding a colon: between the directories:

export PATH=$PATH:/the/file/path:/the/file/path2

echo $PATH

Output:

```
usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/lo
cal/games:/ the/file/path:/the/file/path2
```

man

The "man" is a short term for manual page. In Unix like operating systems such as Linux, man is an interface to view the system's reference manual. A user can request to display a man page by simply typing man followed by a space and then argument. Here its argument can be a command, utility or function. A manual page associated with each of these arguments is displayed.

The basic man command syntax is:

man [option] [section number] [command name]

option – the search result output.

section number – the section in which to look for the man page.

command name – the name of the command which man page you want to see.

But generally [option(s) and section number] are not used. Only command is written as an argument.

some options include

- **-k KEYWORD** (Search for the **KEYWORD** in the whole **manual** page and shows all the **matches**)
- **-f KEYWORD** (Look for a short description of any **KEYWORD** or Command)
- **-d, –default** (Resets the **man** command behavior to **default**)
- **-i, –ignore-case** (**Ignore case sensitivity** of the command)
- **-I, –match-case** (Looking inside the man page with **case sensitivity**)
- **-a, –all** (Shows **all manual** pages that **match** the specific **keyword** or command)

By default, **man** looks in all the available sections of the manual and shows the first match (even if the page exists in several sections). Providing a section number instructs the **man** command to look in a specific section.

There are nine sections of the manual:

1. **General commands:** Commands used in the terminal.
2. **System calls:** Functions the kernel provides.

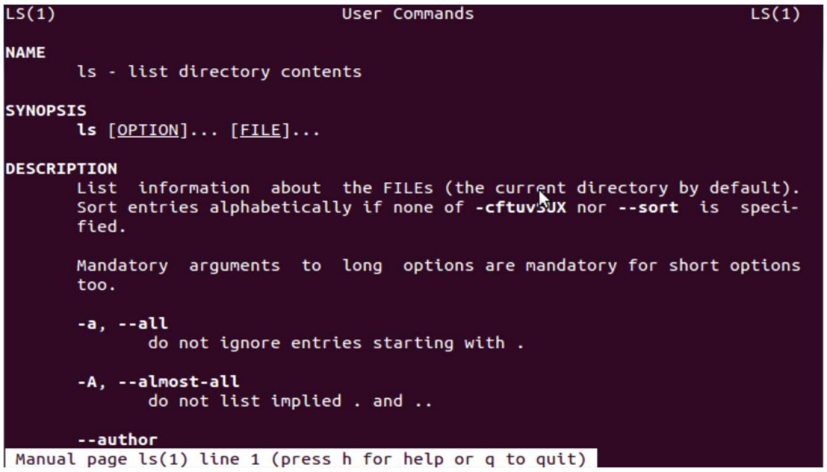
3. **Library functions:** Functions in program libraries.
4. **Special files:** Usually devices found in **/dev** and related drivers.
5. **File formats and conventions:** File formats like **etc/passwd**.
6. **Games:** Descriptions of commands that display database quotes.
7. **Miscellaneous:** Various descriptions, including macro packages and conventions, boot parameters, and others.
8. **System administration commands:** Commands mostly reserved for root.
9. **Kernel Routines:** Information about internal kernel operations.

Example

Syntax of command without option and section

man ls

This command will display all the information about **'ls'** command as shown in the screen shot.



Syntax for a particular section:

man section_number command

Example: man 2 passwd

The **man 2 passwd** command in Linux is intended to show the manual page for the **passwd** system call, which is found in section 2 of the manual. Section 2 of the manual pages typically documents system calls, which are interfaces provided by the Linux kernel.

printf

printf command is used to output a given string, number or any other format specifier. The command operates the same way as *printf* in C, C++, and Java programming languages.

Example

```
$ printf"%s""Hello, Welcome to KMIT”
Output: Hello, Welcome to KMIT
```

script

script command in Linux is used to make typescript or record all the terminal activities. After executing the *script* command, it starts recording everything printed on the screen including the inputs and outputs until exit. By default, all the terminal information is saved in the file *typescript,* if no argument is given.

script

We can specify a filename as an argument to save the output to a different file:

script my_session.log

To end the recording session, type **exit** or press **Ctrl-D**. This will return you to your normal terminal session and stop recording.

Example:

```
script my_session.log
echo "Hello, World!"
ls
exit
```

passwd

The **passwd** command in Linux and Unix-like operating systems is used to change the password of a user account. It can be used by both the system administrators to change other users' passwords and by individual users to update their own passwords.

Basic Usage

For a user to change their own password, they would simply type:

passwd

Upon execution, the system prompts the user to enter their current password (for verification), followed by the new password, and then to retype the new password for confirmation.

uname

The **uname** command in Linux and Unix-like operating systems is used to display system information. It provides details about the kernel name, version, and other system information. By default, without any options, **uname** will print the kernel name.

Options

The **uname** command supports several options that can be used to display specific system information:

- **`-a`**, **`--all`**: Print all available system information (kernel name, nodename, kernel release, kernel version, machine, processor, hardware platform, and operating system).
- **`-s`**, **`--kernel-name`**: Print the kernel name.
- **`-n`**, **`--nodename`**: Print the network node hostname.
- **`-r`**, **`--kernel-release`**: Print the kernel release.
- **`-v`**, **`--kernel-version`**: Print the kernel version.
- **`-m`**, **`--machine`**: Print the machine hardware name (e.g., x86_64).
- **`-p`**, **`--processor`**: Print the processor type or "unknown".
- **`-i`**, **`--hardware-platform`**: Print the hardware platform or "unknown".
- **`-o`**, **`--operating-system`**: Print the operating system.

Example

Displaying the Kernel Name

Simply typing uname without any options will display the kernel name

Example output might be Linux for a Linux system.

who

The **who** command in Linux and Unix-like operating systems is used to display information about users who are currently logged into the system. It provides a list of users, the terminals they are logged in from, the login time, and sometimes the host from which they are accessing the system.

Basic Usage

To run the command, simply type:

who

This will display a list that typically includes the username, terminal name , the date and time of login, and the remote host name or IP address from which the user is accessing the system.

Options

The **who** command supports several options that can alter its output or provide additional information:

- **`-a`**, **`--all`**: Show all information, combining the effects of many other options.

- **-m**: Same as **who am i**, showing information about the current terminal. It's equivalent to running **who** with the **$USER** variable.
- **-q**, **--count**: Display only the names and number of users currently logged on.
- **-H**, **--heading**: Include column headers in the output.
- **-r**, **--runlevel**: Show the current runlevel. This is useful in multi-user environments and for system administrators.
- **-u**, **--users**: Show the current login name, terminal line, login time, idle time and the exit status of a process. The idle time is particularly useful for finding out how long a terminal has been inactive.

Date

The **date** command in Linux and Unix-like operating systems is used to display or set the system's date and time. By default, running the **date** command without any options will display the current date and time according to the system's settings.

Basic Usage

To display the current date and time, simply type:

date

Output: Tue Mar 9 12:34:56 PST 2021

Options

The **date** command supports several options to format the output, set the system's date and time, and more. Some commonly used options include:

- **+%FORMAT**: Allows you to specify the output format of the date/time. For example, **date +"%Y-%m-%d %H:%M:%S"** would output the date and time in the format **YYYY-MM-DD HH:MM:SS**.
- **-u**, **--utc**, **--universal**: Displays or sets the Coordinated Universal Time (UTC).

we can pass the strings like "yesterday", "monday", "last monday" "next monday", "next month", "next year," and many more.

Consider the below commands:

1. date -d now
2. date -d yesterday
3. date -d tomorrow
4. date -d "next monday"

5. date -d "last monday"

The above commands will display the dates accordingly.

Set or Change Date in Linux

To change the system clock manually, use the set command. For example, to set the date and time to *5:30 PM, May 13, 2010*, type:

```
date --set="20100513 05:30"
```

pwd

The **pwd** command in Linux and Unix-like operating systems stands for "Print Working Directory." It is used to display the full pathname of the current working directory. The current working directory is the directory in which the user is currently operating in the shell.

Basic Usage

To use the command, simply type:

pwd

Upon execution, **pwd** outputs the absolute path of the current working directory to the terminal. This path is a full path from the root of the filesystem (/) to the directory you are currently in.

Example

If you are currently in the home directory of a user named **user**, running **pwd** might output something like:/home/user

cd

In Linux, the **cd** command stands for "change directory." It's a shell command used to change the current working directory in the command line interface. The **cd** command is one of the most frequently used commands in Linux, as navigating between directories is a common task.

For example, if you are working in /home/username/Documents and want to go in the Pictures subdirectory of the same directory, you can write *cd Pictures*.

If you want to go to a new directory, you can write cd followed by the absolute path of the directory – *cd /home/username/Music*.

Command	Description
cd	to go to the home folder
cd..	to move one directory up
cd-	move to your previous directory

ls

The **ls** command in Linux is used to list the contents of a directory. It's one of the most commonly used commands in Linux for navigating the filesystem and viewing the files and subdirectories contained within a directory. By default, **ls** will list the contents of the current working directory.

ls command syntax

ls [Options] [File]

Some of the common option tags that you can use with the ls command:

Option	Description
ls –R	lists all the files in the sub-directories as well
ls –S	sorts and lists all the contents in the specified directory by size
ls -al / ls -l	list the files and directories with detailed information
ls -a	shows the hidden files in the specified directory

touch

The **touch** command in Linux is used primarily to create new, empty files and to change the timestamps of existing files. It's a versatile utility for file handling, particularly useful for developers and system administrators for various tasks such as file manipulation, script creation, and timestamp management.

Creating New Files

To create a new file, simply use the **touch** command followed by the name of the file you want to create. If the file does not exist, **touch** will create a new, empty file with that name. If the file already exists, **touch** will update its access and modification times to the current time without altering the file content.

touch newfile.txt

Changing File Timestamps

The primary function of **touch** is to update the timestamps on a file. Every file in Linux has three main timestamps:

- **Access time (atime)**: The last time the file was read.
- **Modification time (mtime)**: The last time the file's content was modified.
- **Change time (ctime)**: The last time the file's metadata (e.g., permissions) or content was changed.

Using **touch**, you can update the access and modification times to the current system time. This can be useful for various purposes, such as triggering processes that are dependent on file timestamps or avoiding archiving based on outdated times.

Options

touch comes with several options that allow you to specify which timestamps to update and to use custom timestamps rather than the current time:

- **-a**: Change only the access time.
- **-m**: Change only the modification time.
- **-t [STAMP]**: Use a specific timestamp instead of the current time. The STAMP argument is in the format **[[CC]YY] MMDDhhmm[. ss]** where each letter represents a component of the date and time (year, month, day, hour, minute, second).

For example, to change the modification time of a file to a specific date and time, you could use:

touch -m -t 202307040930 newfile.txt

This command sets the modification time of **newfile.txt** to July 4, 2023, at 9:30 AM.

Creating Multiple Files

touch can also be used to create multiple files at once by specifying more than one filename:

touch file1.txt file2.txt file3.txt

This command creates three new files named **file1.txt**, **file2.txt**, and **file3.txt** in the current directory, or updates their timestamps if they already exist.

mv

The **mv** command in Linux is used for moving or renaming files and directories. It is a versatile

command that is frequently used for organizing files, updating their locations, or changing their names. The basic syntax for the **mv** command is:

mv [options] source destination

Here are the primary ways to use the **mv** command:

Moving Files

To move a file from one location to another, you specify the current path of the file as the source and the desired path as the destination.

mv /path/to/source/file.txt /path/to/destination/

This command moves **file.txt** from its current location to a different directory.

Renaming Files

To rename a file, you use **mv** with the current filename as the source and the new filename as the destination, within the same directory.

mv oldfilename.txt newfilename.txt

This command renames **oldfilename.txt** to **newfilename.txt**.

Moving Multiple Files

You can also move multiple files to a directory by listing each file as a source before the destination directory.

mv file1.txt file2.txt /path/to/destination/

This command moves **file1.txt** and **file2.txt** to the specified directory.

rm

The **rm** (remove) command in Linux is used to delete files and directories from the filesystem. It's a powerful tool that must be used with caution, as once a file is deleted using **rm**, it typically cannot be recovered easily.

Removing Files: To remove a single file, use the rm command followed by the filename:

rm filename.txt

This command deletes **filename.txt** from the filesystem.

Removing Multiple Files: You can also delete multiple files at once by listing each file as an argument:

rm file1.txt file2.txt file3.txt

Removing Directories: To remove a directory and its contents, you need to use the -r (or --recursive) option, which tells rm to remove directories and their contents recursively.

rm -r directoryname

This command deletes **directoryname** and everything within it, including all files and subdirectories.

To modify the command, add the following options:

- **-i** – prompts a confirmation before deletion.
- **-f** – allows file removal without a confirmation.
- **-r** – deletes files and directories recursively.

mkdir

The **mkdir** command in Linux is used to create new directories. It stands for "make directory," and it allows users to create one or multiple directories at once.

To create a single directory, use the mkdir command followed by the name of the directory you want to create:

mkdir newdirectory

This command creates a new directory named **newdirectory** in the current working directory.

Creating Multiple Directories

You can create multiple directories at once by listing each one as an argument to the **mkdir** command:***mkdir dir1 dir2 dir3***

Sometimes you might want to create a directory and its parent directories at the same time. You can do this using the **-p** (or **--parents**) option, which tells **mkdir** to create any necessary parent directories as well:

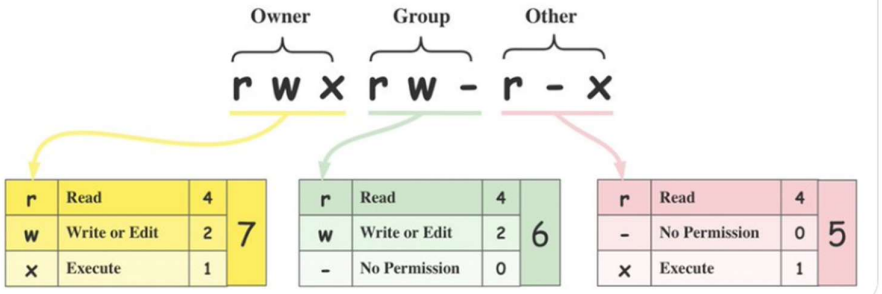
mkdir -p parentdir/subdir/subsubdir

This command creates the directory **subsubdir** and also creates **parentdir** and **subdir** if they don't already exist.

Here are several common **mkdir** command options:

- **-p** – creates a directory between two existing folders. For example, **mkdir -p Music/2024/Songs** creates a new **202**4 directory.
- **-m** – sets the folder permissions. For instance, enter **mkdir -m777 directory** to create a directory with read, write, and execute permissions for all users.

Binary	Octal	String Representation	Permissions
000	0 (0+0+0)	---	No Permission
001	1 (0+0+1)	--x	Execute
010	2 (0+2+0)	-w-	Write
011	3 (0+2+1)	-wx	Write + Execute
100	4 (4+0+0)	r--	Read
101	5 (4+0+1)	r-x	Read + Execute
110	6 (4+2+0)	rw-	Read + Write
111	7 (4+2+1)	rwx	Read + Write + Execute



- **-v** – prints a message for each created directory.

rmdir

The **rmdir** command in Linux is used to remove empty directories. It stands for "remove directory," and its primary purpose is to delete directories that are no longer needed. However, it's important to note that **rmdir** will only remove a directory if it is empty. If the directory contains files or subdirectories, **rmdir** will not delete it and will instead show an error message.

To remove a single empty directory, use the rmdir command followed by the name of the directory you want to delete:

rmdir directoryname

This command removes the directory named **directoryname** if it is empty.

Removing Multiple Directories

You can remove multiple empty directories at once by listing each one as an argument to the **rmdir** command:

rmdir dir1 dir2 dir3

Removing Parent Directories

To remove a directory and its empty parent directories at the same time, you can use the **-p** (or **--parents**) option. This option tells **rmdir** to remove each directory in the specified path, working its way up the path as long as the directories are empty:

rmdir -p parentdir/subdir

This command attempts to remove **subdir** and then **parentdir** if they are both empty. If **parentdir** contains other files or directories, it will not be removed.

tar

The **tar** command in Linux is a highly versatile tool used for creating and manipulating archive files. The name "tar" stands for "Tape Archive," reflecting its original purpose for writing data to tape drives. Over time, it has become a standard utility for file archiving and compression in Unix-like operating systems. **tar** archives multiple files and directories into a single file (often called a tarball), which can be optionally compressed using various compression algorithms. Here's how to use the **tar** command with some of its most common options:

Creating an Archive

To create a **.tar** archive, use the **-c** option (for create), **-f** followed by the archive file name, and then list the files and directories you want to archive:

tar -cf archive_name.tar file1 directory1 file2

This command creates an archive named **archive_name.tar** containing **file1**, **directory1**, and **file2**.

Extracting an Archive

To extract the contents of a **.tar** archive, use the **-x** option (for extract) and **-f** followed by the archive file name:

tar -xf archive_name.tar

This command extracts the contents of **archive_name.tar** into the current working directory.

Viewing the Contents of an Archive

To view the contents of a **.tar** archive without extracting it, use the **-t** option (for list) and **-f** followed by the archive file name:

tar -tf archive_name.tar

This command lists the files and directories contained in **archive_name.tar**.

gzip

gzip command compresses files. Each single file is compressed into a single file. The compressed file consists of a GNU zip header and deflated data. If given a file as an argument, gzip compresses the file, adds a ".gz" suffix, and deletes the original file.

gzip [Options] [filenames]

This syntax allows users to compress a specified file. Now, let's delve into some practical examples to illustrate the usage of the gzip command.

Options	Description
-f	Forcefully compress a file even if a compressed version with the same name already exists.
-k	Compress a file and keep the original file, resulting in both the compressed and original files.
-L	Display the gzip license for the software.
-v	Display the name and percentage reduction for each file compressed or decompressed.
-d	Decompress a file that was compressed using the gzip command

Basic Compression using gzip Command in Linux

To compress a file named "mydoc.txt," the following command can be used:

Example:

gzip mydoc.txt

This command will create a compressed file of mydoc.txt named as mydoc.txt.gz and delete the original file.

How to decompress a gzip file in Linux?

The basic syntax of the gzip command for decompressing a file is as follows:

gzip -d filename.gz

This command decompresses the specified gzip file, leaving the original uncompressed file intact.

Keeping the Original File Using gzip Command in Linux

By default, gzip removes the original file after compression. To retain the original file, use the -k option:

gzip -k example.txt

This command compresses "example.txt" and keeps the original file intact.

Verbose Mode Using gzip Command in Linux

To obtain more details during compression or decompression, the -v option is employed:

gzip -v example.txt

Verbose mode provides information such as file sizes and progress during the compression or decompression process.

Force Compression Using gzip Command in Linux

In cases where the compressed file already exists, the -f option forcefully overwrites it:

gzip -f example.txt

This command compresses "example.txt" and overwrites any existing "example.txt.gz" file

Compressing Multiple Files Using gzip Command in Linux

Gzip can compress multiple files simultaneously by providing their names as arguments:

gzip file1.txt file2.txt file3.txt

This command compresses "file1.txt," "file2.txt," and "file3.txt" individually.

cat

The **cat** command in Linux is a short form for "concatenate." It is one of the most frequently used commands in Unix and Linux operating systems for various purposes. The primary function of **cat** is to read (display), combine, and copy text files. Here are some common uses and examples of how the **cat** command can be utilized:

Display the Contents of a File

To display the contents of a file, simply use **cat** followed by the file name. For example:

cat filename.txt

This command will display the contents of **filename.txt** on the standard output (usually the terminal).

Combine Multiple Files

cat can also be used to combine several files into one. For example, to concatenate the contents of **file1.txt** and **file2.txt** and display the combined content in the terminal, you can use:

cat file1.txt file2.txt

To save the combined content into a new file, you can redirect the output to another file:

cat file1.txt file2.txt > combined.txt

Append Content to a File

You can use **cat** to append content to the end of an existing file. For example, to append **file2.txt** to **file1.txt**, you can use:

cat file2.txt >> file1.txt

This command appends the contents of **file2.txt** to **file1.txt** without overwriting the original content of **file1.txt**.

Create a New File

cat can be used to create a new file by redirecting its output. For example:

cat > newfile.txt

After running this command, the terminal waits for you to input text. You can type the content of the new file, and once done, press **Ctrl+D** to save and exit.

Display Line Numbers

To display the contents of a file with line numbers, you can use **cat** with the **-n** option:

cat -n filename.txt