Concepts of Operating Systems

- Vineela

Session 1 : Introduction to OS Lecture

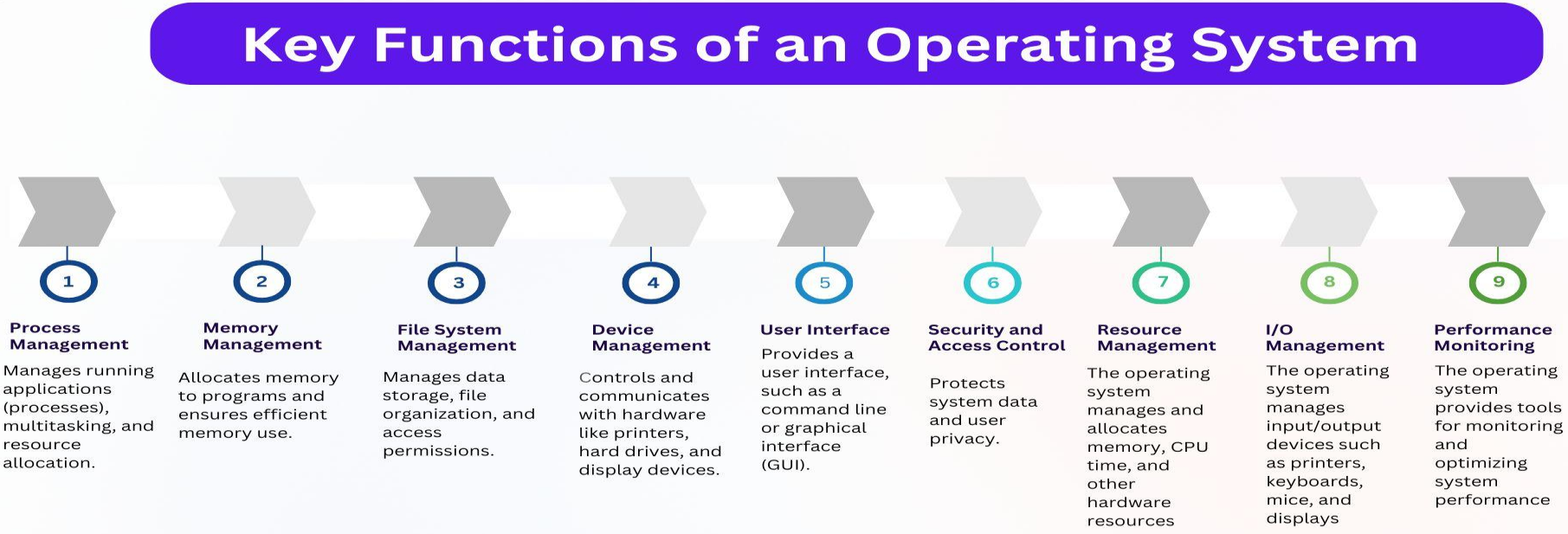
Lecture:

- What is OS; How is it different from other application software; Why is it hardware dependent?
- Different components of OS
- Basic computer organization required for OS.
- Examples of well-known OS including mobile OS, embedded system OS, Real Time OS, desktop OS server machine OS etc. ; How are these different from each other and why
- Functions of OS
- User and Kernel space and mode; Interrupts and system calls

What is OS; How is it different from other application software; Why is it hardware dependent?

- **What is OS?**

  It is **system software that manages computer hardware, software resources and provides common services for computer programs**.



How is OS different from other application software

Aspect	Operating System (OS)	Application Software
Purpose and Functionality:	An OS is system software designed to manage hardware and provide a platform for running other software applications. It controls and coordinates the use of hardware resources (like CPU, memory, and storage), ensuring the efficient and safe operation of a computer. **Examples** - Windows, macOS, Linux, Android, etc.	Application software is designed to perform specific tasks or solve particular problems for users. It **runs on top of an operating system** and interacts with it, but it does not manage hardware or control the system's core functions. **Examples** - Microsoft Word, Google Chrome, Photoshop etc.
Interaction with Hardware:	Directly interacts (**at a system level)** with the hardware and serves as an intermediary between the hardware and application software. It manages hardware resources like memory, processor, storage, and devices (e.g., printers, USB drives,task manager).	It operates **at the user level**, with a direct focus on the specific task or use case for the end-user. **For example**, when you open a document in Word, the application relies on the OS to handle memory and file management.

- **In short, an OS is like the foundation of a house, and application software is the furniture and decor placed on that foundation.**

Is OS is hardware dependant or not?

Is OS is hardware dependant or not?

Yes ,because it directly interacts and manages the hardware components of a computer or device

What are the Aspects of hardware dependency?

Hardware-Specific Drivers:

- Each type of hardware (such as the CPU, memory, storage devices, network interfaces, printer, and input devices (keyboard, mouse) needs **device drivers** tailored for it, so the OS must be designed to support the hardware it's running on.

Resource Management:

- OS allocates and manages **hardware resources** like CPU time, memory, and storage, so the OS must be aware of and adapt to the system's specific hardware resources.

CPU Architecture:

- A program written for an **x86** processor won't run on an **ARM** processor without specific adaptation, and vice versa. This means an **OS must be compiled for the particular type of CPU** used by the system.

Memory and Storage Management:

- Different systems can have different amounts and types of memory or storage (e.g., SSD vs. HDD), so It must be able to **detect, allocate, and deallocate memory** and ensure that different programs don't interfere with each other's memory spaces.

What are the Aspects of hardware dependency?

I/O Management

- The OS handles all **I/O operations** (like reading from or writing to files, input from a keyboard, or output to a monitor). and require specific interfaces and methods for communication, which the OS needs to manage in a hardware-dependent way.

Power Management:

- OS must control **how power is distributed to different components** (CPU, screen, wireless network, etc.) and this depends on the hardware capabilities of the device.

System Architecture and Customization:

- OS has to be **customized for each of the hardware environments of different devices** (like smartphones, desktops, and servers) and different hardware configurations, such as the number of cores in the CPU, types of sensors, and GPU types to take full advantage of the hardware features and optimize performance.

Different Components of OS

Process Management

- Handles the execution of processes.
- Manages CPU scheduling, process creation, and termination.
- Ensures smooth multitasking and prevents deadlocks.

Memory Management

- Allocates and deallocates memory for processes.
- Manages virtual memory and paging.
- Ensures efficient use of RAM.

File System Management

- Organizes and manages files and directories.
- Handles file permissions and access control.
- Provides storage management.

I/O Device Management

- Manages input and output devices like keyboards, printers, and displays.
- Uses device drivers to communicate with hardware.
- Ensures efficient data transfer.

Security & Access Control

- Protects data and system resources from unauthorized access.
- Implements authentication and encryption mechanisms.
- Prevents malware and cyber threats.

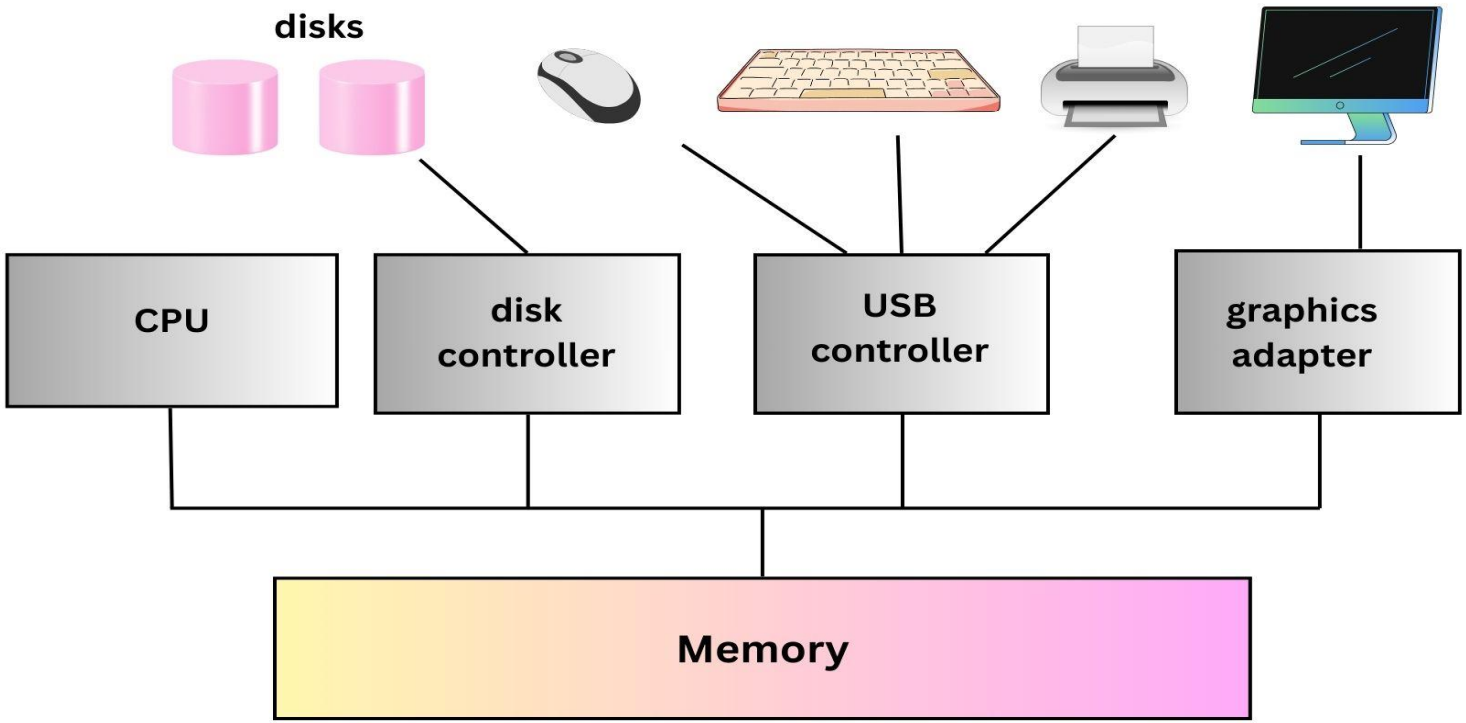
Network Management

- Manages communication between devices over a network.
- Handles protocols, data transmission, and connectivity.
- Supports internet and local networking.

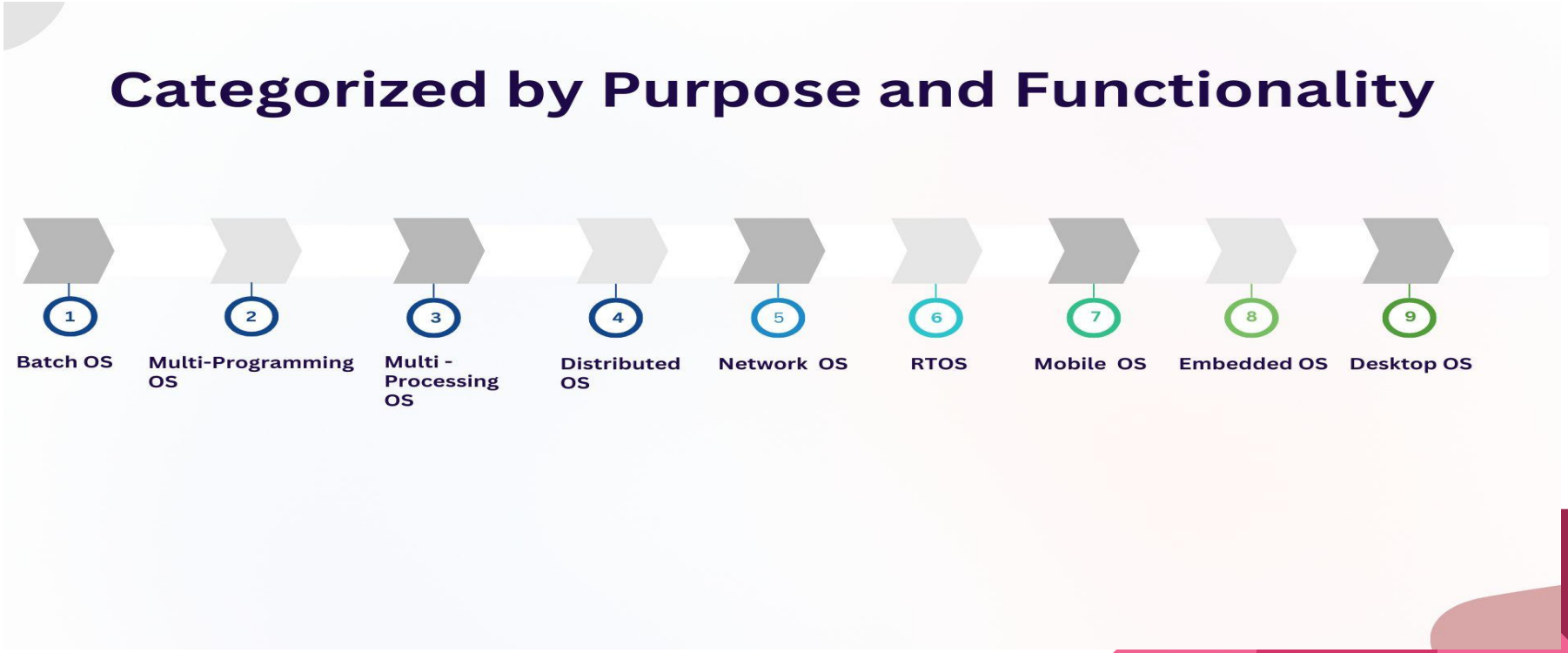
Command Interpreter (Shell)

- Provides an interface for users to interact with the OS.
- Executes commands and scripts.
- Can be graphical (GUI) or command-line (CLI).

Basic computer organization required for OS



Examples of well-known OS including mobile OS, embedded system OS, Real Time OS, desktop OS server machine OS etc. ; How are these different from each other and why

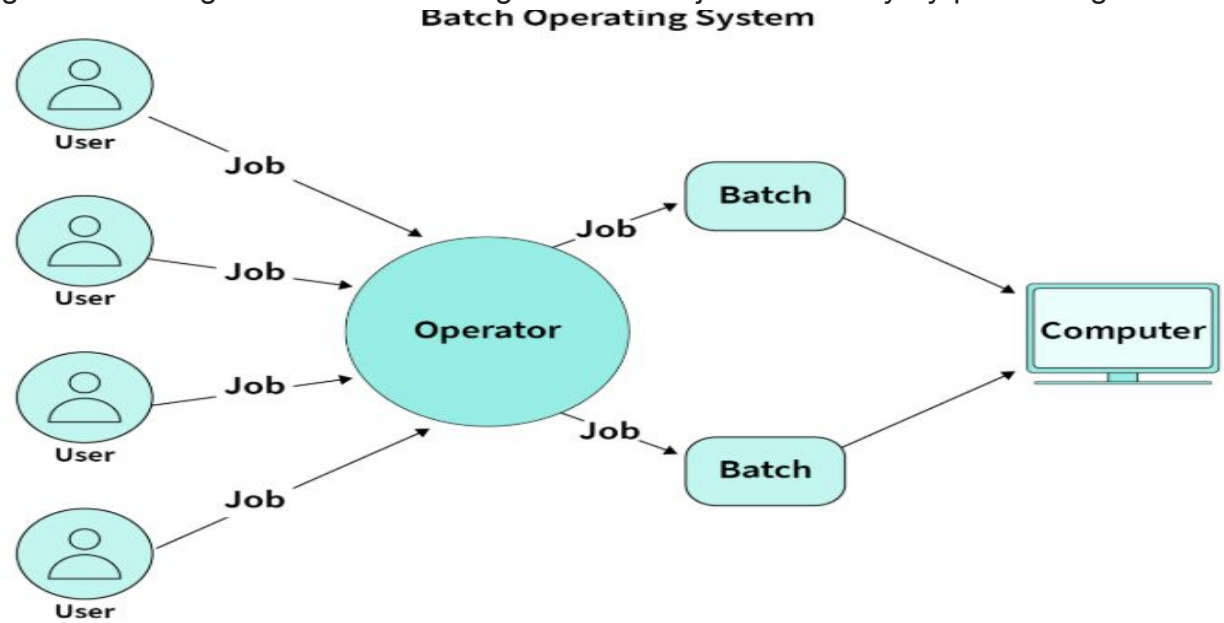


1) Batch Operating System

- It does not interact with the computer directly. There is an operator which takes **similar jobs having the same requirements and groups them into batches**.
- It is the **responsibility of the operator** to sort jobs with similar needs.
- Batch Operating System is designed to manage and execute a large number of jobs efficiently by processing them in groups.

Examples

- Payroll System
- Bank Invoice System
- Transactions Process
- Daily Report
- Research Segment
- Billing System



Advantages and Disadvantages of Batch Operating System

Advantages	Disadvantages
Multiple users can share the batch systems.	CPU is not used efficiently. When the current process is doing IO, the CPU is free and could be utilized by other processes waiting.
The idle time for the batch system is very little.	Other jobs will have to wait for an unknown time if any job fails.
It is easy to manage large work repeatedly in batch systems.	Average response time increases as all processes are processed one by one.

2) Multi - Programming OS

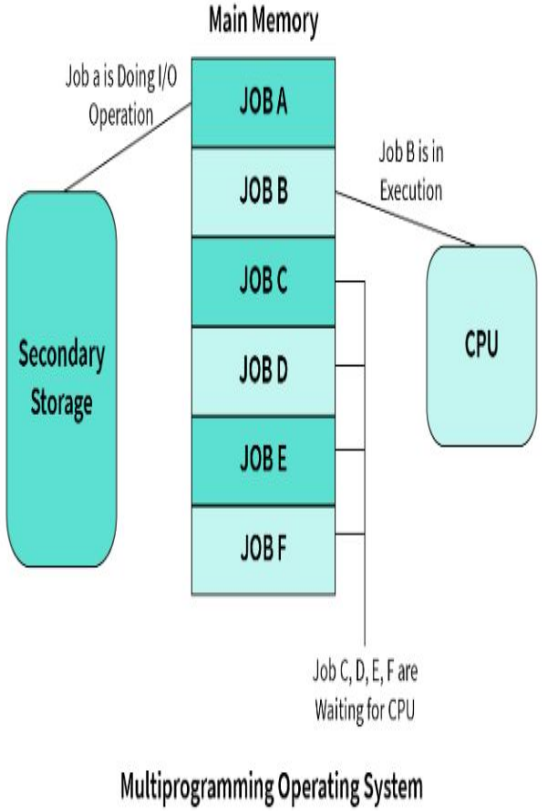
- Multiprogramming Operating Systems can be simply illustrated as, **more than one program is present in the main memory and any one of them can be kept in execution**, so this is used for better utilization of resources.

Examples

- Apps like office, chrome, etc.
- Microcomputers like MP/M, XENIX, and ESQview.
- Windows O/S
- UNIX O/S

A multiprogramming OS is of the following two types:

1. **Multitasking /Time-sharing OS**: Enables execution of multiple programs at the same time, by swapping each program in and out of memory one at a time.
2. **Multiuser Operating System**: This allows many users to share processing time on a powerful central computer from different terminals, by rapidly switching between terminals.



Multiprogramming Operating System

Advantages and Disadvantages of Multi - Programming OS

Advantages

- Great Reliability

  Reliability refers to the probability that the system will perform its intended functions correctly and without failure for a specified period under given conditions

- Improve Throughput

  Throughput refers to the amount of work or data processed within a specific time frame

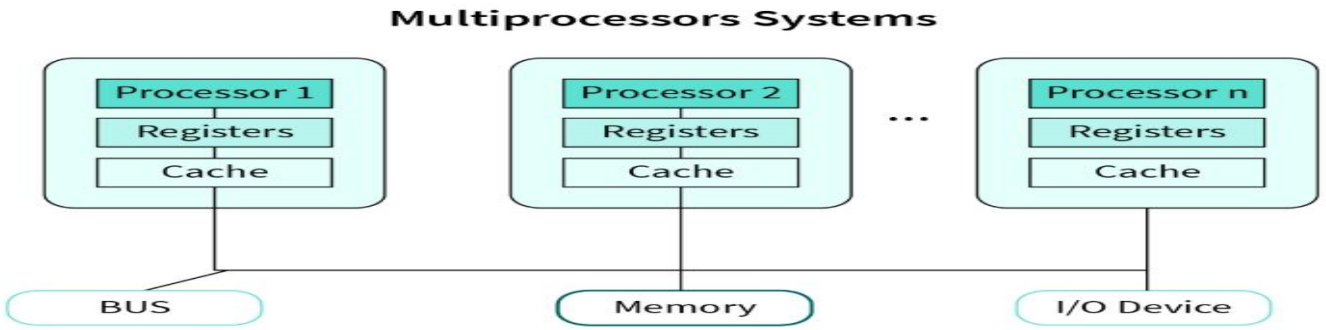
- Cost-Effective System
- Parallel Processing

Disadvantages

- It is more expensive due to its large architecture.
- Its speed can get degraded due to failing any one processor.
- It has more time delay when the processor receives the message and takes appropriate action.
- It has big challenges related to skew and determinism
- It needs context switching which can impact its performance.

3) Multi - Processor OS

- A Multi-Processing Operating System is a type of Operating System **in which more than one CPU is used for the execution of resources**. It betters the throughput of the System.
- The following are four major components, used in the Multiprocessor Operating System:
  1. **CPU** – capable of accessing memories as well as controlling the entire I/O tasks.
  2. **Input Output Processor** – The I/P processor can access direct memories, and every I/O processor has to be responsible for controlling all input and output tasks.
  3. **Input/Output Devices** – These devices are used for inserting the input commands, and producing output after processing.
  4. **Memory Unit** – Multiprocessor system uses two types of memory modules - shared memory and distributed shared memory.



Advantages and Disadvantages of Multi - Processor OS

Advantages

- Failure of one processor does not affect the functioning of other processors.
- It divides all the workload equally to the available processors.
- Makes use of available resources efficiently.

Disadvantages

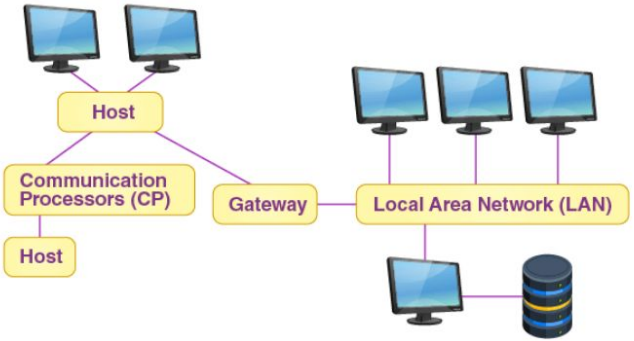
- Symmetrical multiprocessing OS are more complex.

Symmetric multiprocessing is a computer architecture where two or more identical processors share the same memory and input/output (I/O) devices, and are controlled by a single operating system

- They are more costlier.
- Synchronization between multiple processors is difficult.

4) Distributed Operating Systems

- The Distributed OS is separated into sections and **loaded on different machines rather than being placed on a single machine**
- All processors are **connected by valid communication mediums** such as high-speed buses and telephone lines, LAN/WAN lines and in which every processor contains its local memory along with other local processors



A typical view of a distributed System

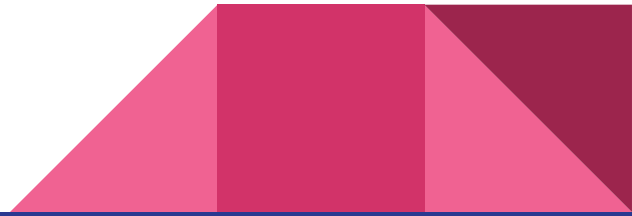
Advantages and Disadvantages of Distributed OS

Advantages:

- Increased Reliability
- Scalability
- Resource Sharing
- Improved Performance

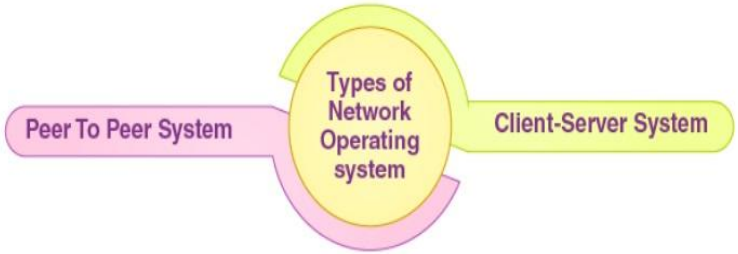
Disadvantages:

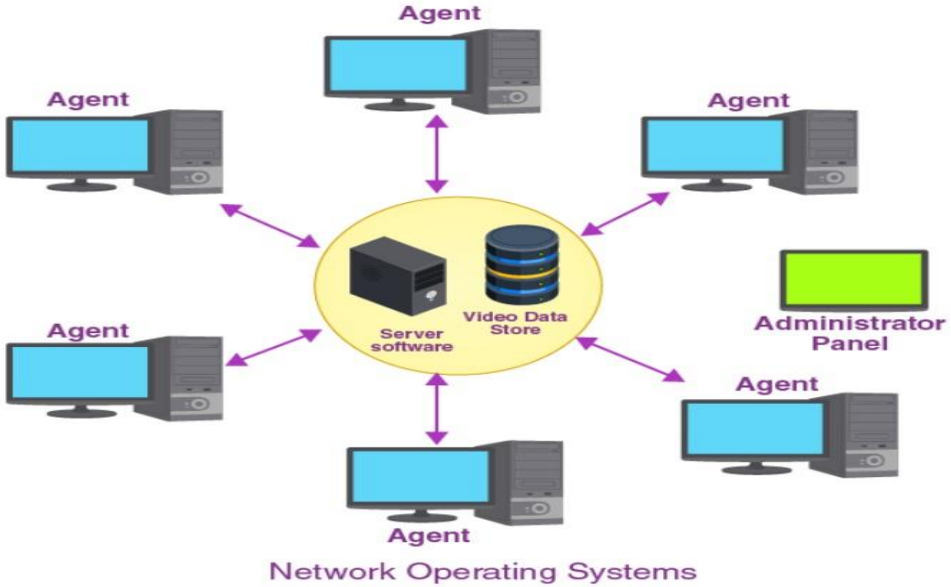
- Complex
- Security Concerns
- Network Dependency
- Consistency and Data Integrity
- Difficulty in Troubleshooting



5) Network Operating Systems

- Network Operating System has **special functions for connecting computers and devices into a local-area network or Inter-network.** Some popular network operating systems are Novell Netware, Linux, IBM OS/2, etc.
- There are two basic types of network operating systems:
    1. **Peer-to-Peer Network Operating Systems:** Allow users to **share network resources** saved in a common, accessible network location.
    2. **Client/Server Network Operating Systems**: Provide users with **access to resources through a server**.





Network Operating Systems

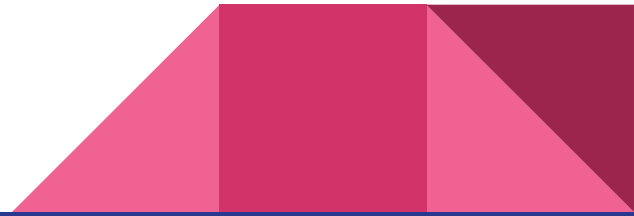
Advantages and Disadvantages Network Operating Systems

Advantages

- Centralized Management
- Enhanced Security
- Resource Sharing
- Cost-Effectiveness

Disadvantages

- Dependency and Potential for Failure
- High Setup and Maintenance Costs
- Performance Issues
- Security Risks



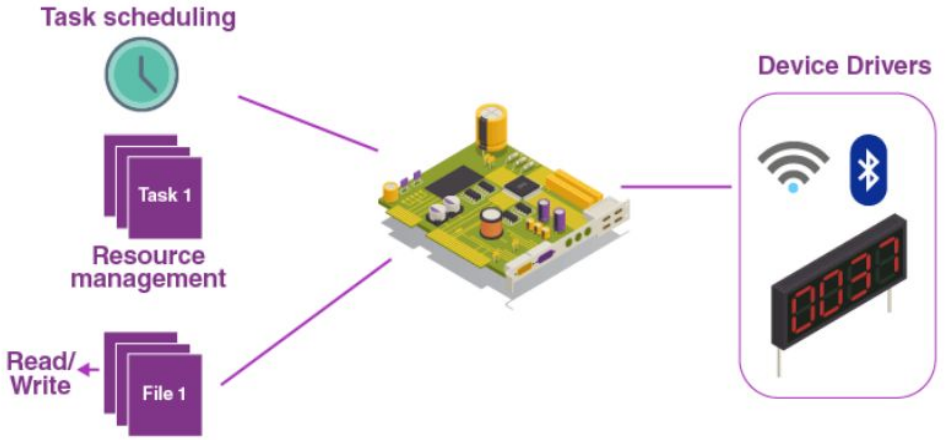
6) Real Time Operating Systems

- In this type of system, **each job has a deadline by which it must be completed; otherwise, there will be a significant loss**, or even if the output is provided, it will be utterly useless.

Example - In military applications, if you wish to drop a missile, the missile must be dropped with a specific degree of precision

Examples

- Airline traffic control systems
- Command Control Systems
- Airlines reservation system
- Heart Pacemaker
- Network Multimedia Systems
- Robotics



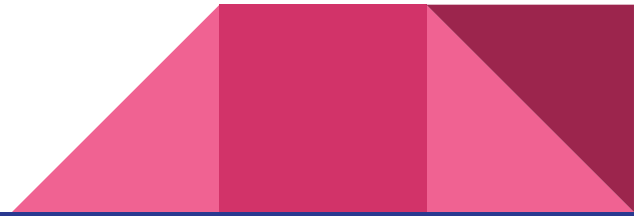
Advantages and Disadvantages of RTOS

Advantages

- Maximum utilization of devices and systems
- Error Free
- Best Memory allocation
- Time assigned for shifting tasks in these systems is very less

Disadvantages

- Usage of expensive system resources
- Complex Algorithms
- Device Driver And Interrupt Signals



7) Mobile Operating Systems

- It helps run application software on mobile devices
- The operating systems found on smartphones include Symbian OS, IOS, BlackBerryOS, Windows Mobile, Palm WebOS, Android, and Maemo
- Android, WebOS, and Maemo are all derived from Linux
- iPhone OS originated from BSD and NeXTSTEP, which are related to Unix

Examples

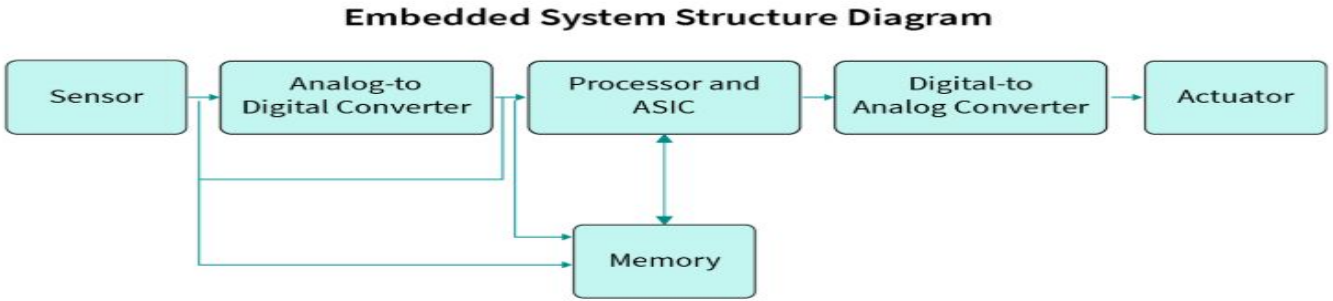
- Android
- IOS
- HarmonyOS
- PalmOS

8) Embedded Operating Systems

- It is built on Internet of Things devices
- It aims to perform with certainty specific tasks regularly that help the device operate.
- An embedded operating system often has limited features and functions.

Examples

- Windows Mobile/CE (handheld Personal Data Assistants)
- Symbian (cell phones)
- Linux-based OSes.



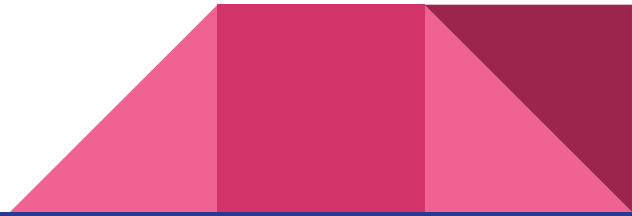
Advantages and Disadvantages of Embedded OS

Advantages

- The OS is often low-cost
- The OS tends to use few resources, including minimal power
- The performance is generally trouble-free.

Disadvantages

- Usually only run a single or very few applications.
- It is difficult to modify the OS
- Trouble-shooting can be difficult
- Inconsistent and timely execution of action
- Limited power and memory

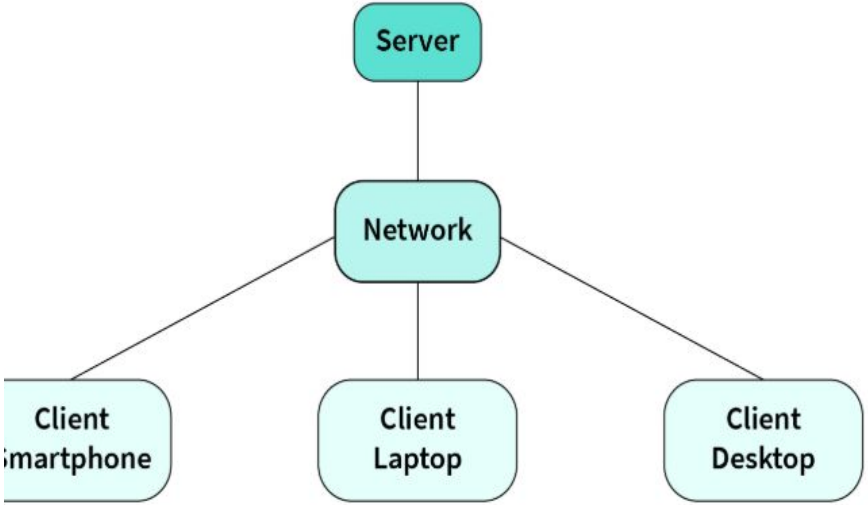


9) Desktop Operating System

- The Client System can be said as a computer in a network where the user performs some task or activity over the network. Such OS **do not have complete control** over the resources but **use the network to access.**
- The processing power remains in the hands of the server OS, which is developed in such a way that it can fulfill all the requirements of the client or the desktop operating system.

Examples

- Windows
- Linux
- Unix
- MAC OS
- MS-DOS
- Solaris
- Ubuntu
- Fedora
- QNX



Advantages and Disadvantages of Desktop OS

Advantages

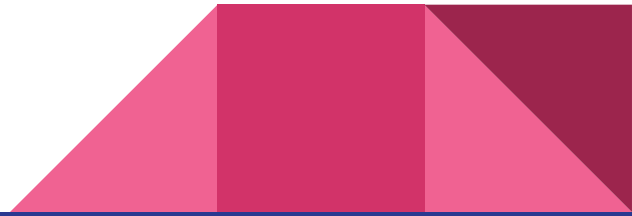
- Centralization of resources are present at a common location.
- Better management of resources as the files are stored in a single place.
- Remote access to the server gives processing power to every user.
- High security as only the server needs to be secured from threats and attacks.
- The server can play different roles for the different

Disadvantages

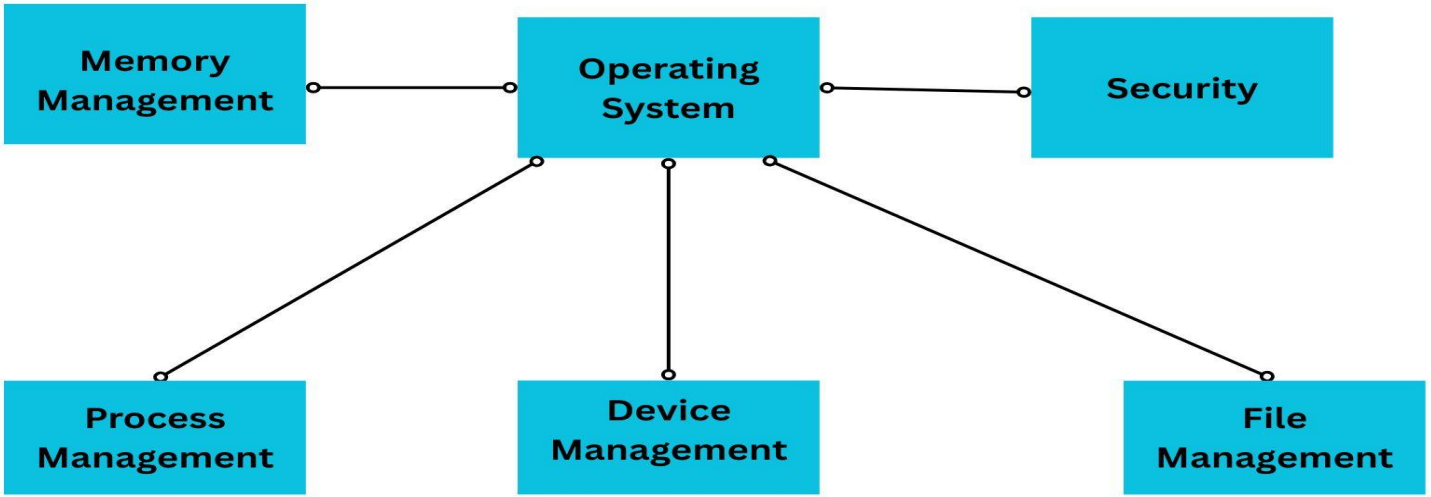
- Network congestion as multiple requests from the clients can block the network traffic.
- The architecture of request and response is not robust enough for heavy processing.
- If the server fails, all the desktop systems connected over the network fail.
- If the service interrupts, the task has to be started from scratch. For instance, if a desktop system requests a file download that gets interrupted, the file becomes corrupt, and the entire process needs to be carried out from the start.
- The operating system architecture is highly costly.
- A professional IT personnel is needed to manage and maintain such an operating environment.

Functions of OS

- The main goal of an operating system is to make the computer environment more convenient to use and to utilize resources most efficiently.
- Operating System handles the following responsibilities:
  - Controls all the computer resources.
  - Provides valuable services to user programs.
  - Coordinates the execution of user programs.
  - Provides resources for user programs.
  - Provides an interface (virtual machine) to the user.
  - Hides the complexity of software.
  - Supports multiple execution modes.
  - Monitors the execution of user programs to prevent errors.

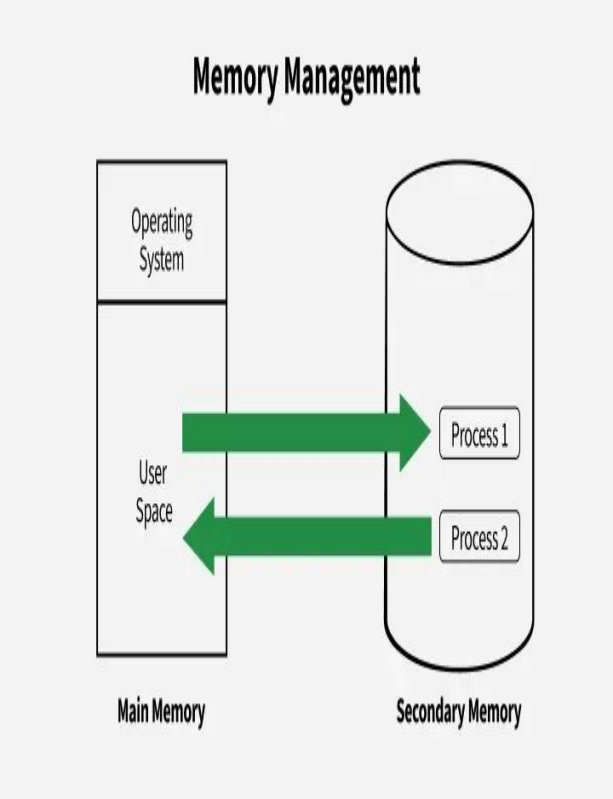


Functions of OS



Memory Management

- OS handles the storage and organization of data in both main (primary) memory and secondary storage.
- It ensures that memory is allocated and deallocated properly to keep programs running smoothly.
- It also manages the interaction between volatile main memory and non-volatile secondary storage.



Key Activities in Memory Management:

Main Memory Management

- **Memory Allocation:** Assigns memory to processes using techniques like paging and segmentation.
- **Memory Deallocation:** Frees memory when no longer needed.
- **Memory Protection:** Prevents processes from accessing each other's memory.
- **Virtual Memory:** Uses disk space as extra memory to run larger processes.
- **Fragmentation:** Manages wasted memory space (internal/external) through compaction.

Secondary Memory Management

- **Disk Space Allocation**: Organizes how files are stored on the disk (contiguous, linked, indexed).
- **File System Management**: Manages files and directories for efficient data access.
- **Free Space Management**: Tracks available space on the disk.
- **Disk Scheduling**: Organizes the order of disk read/write requests.
- **Backup and Recovery**: Ensures data is backed up and can be restored after failure.

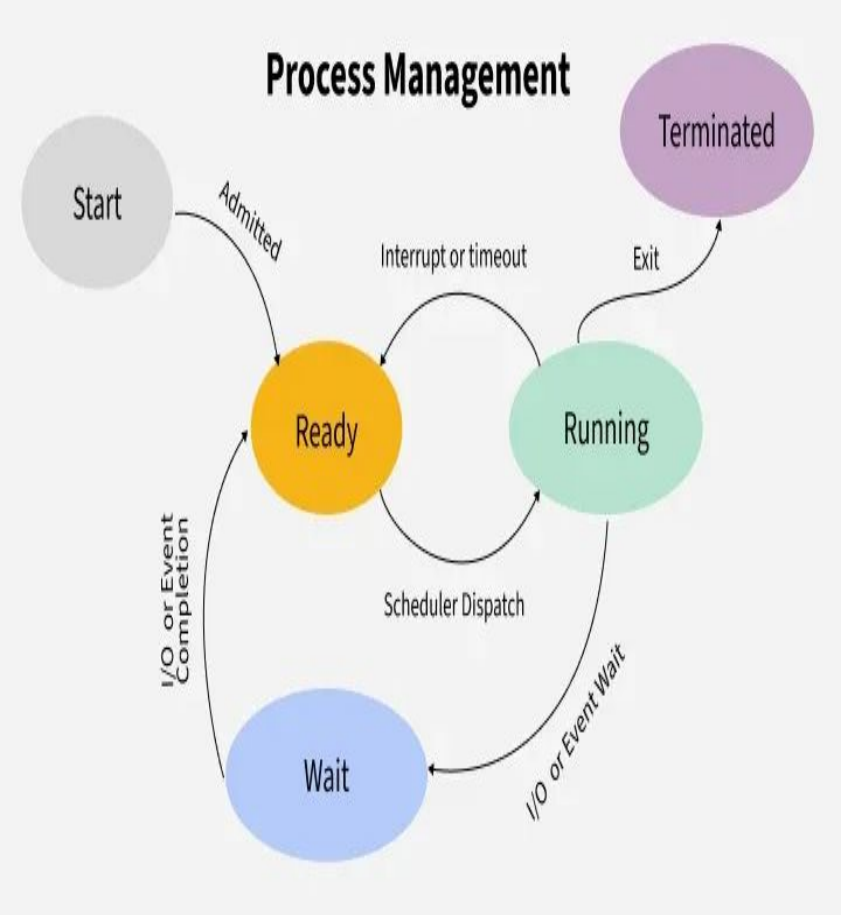
Process Management

- A Process is a running program from the moment program start and until it finishes.

  Operating system makes sure each process:

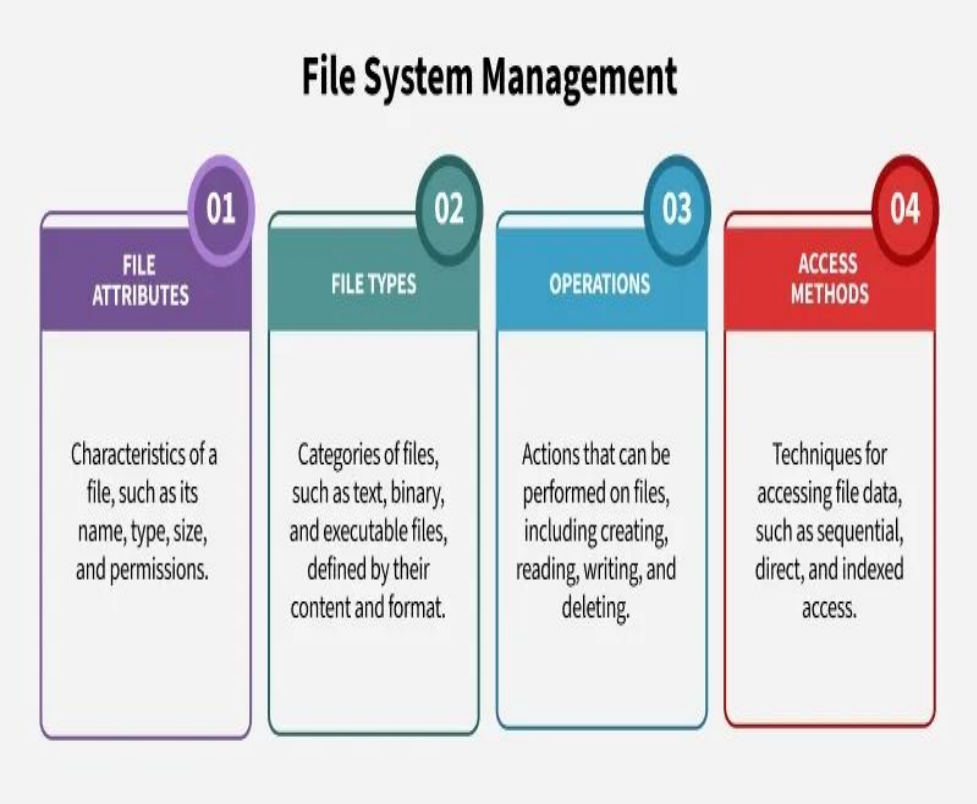
  - gets its turn to use the CPU
  - synchronized when needed
  - has access to the resources it needs, like memory, files, and input/output devices.
- It also handles issues like process coordination and communication, while preventing conflicts such as deadlocks. This way, the OS ensures smooth multitasking and efficient resource use.

- **Core Functions in Process Management:**
  Process Scheduling,Process Synchronization,Inter-Process Communication (IPC), Deadlock Handling



File System Management

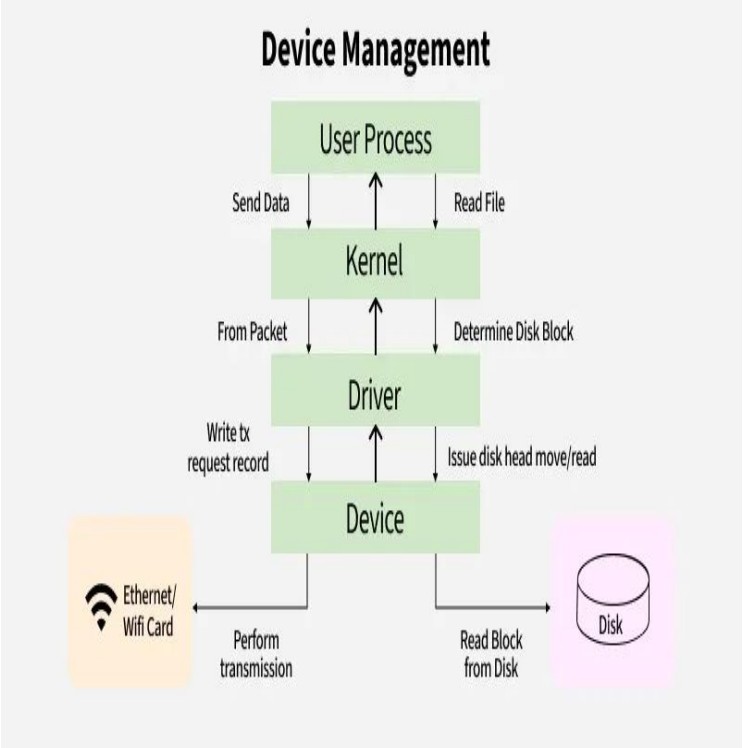
- File management in the operating system ensures the organized storage, access and control of files.
- The OS **abstracts the physical storage details to present a logical view of files**, making it easier for users to work with data.
- It manages how files are stored on different types of storage devices (like hard drives or SSDs) and ensures smooth access through directories and permissions.

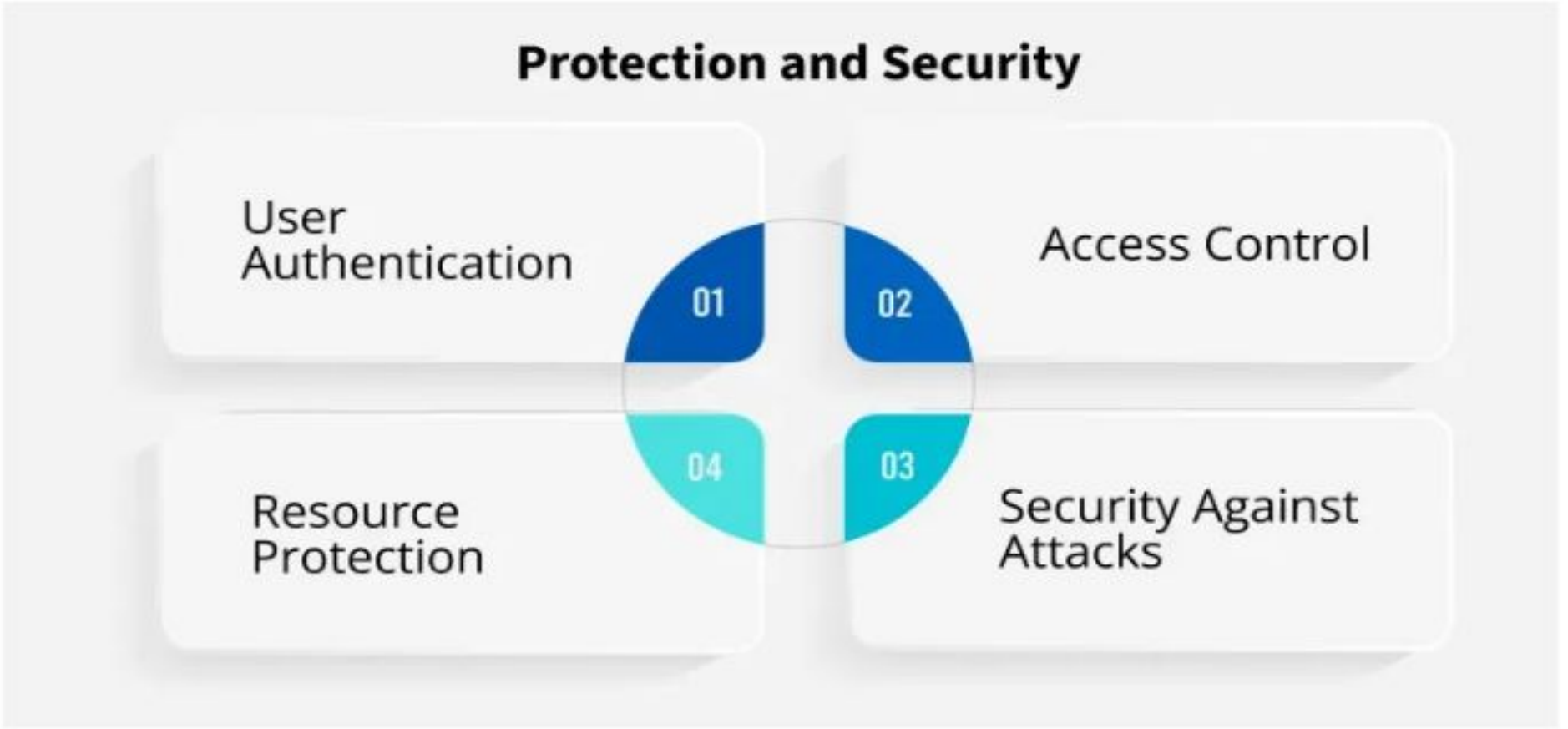


Device Management (I/O System)

- Device management of an operating system handles the communication between the system and its hardware devices, like printers, disks or network interfaces.

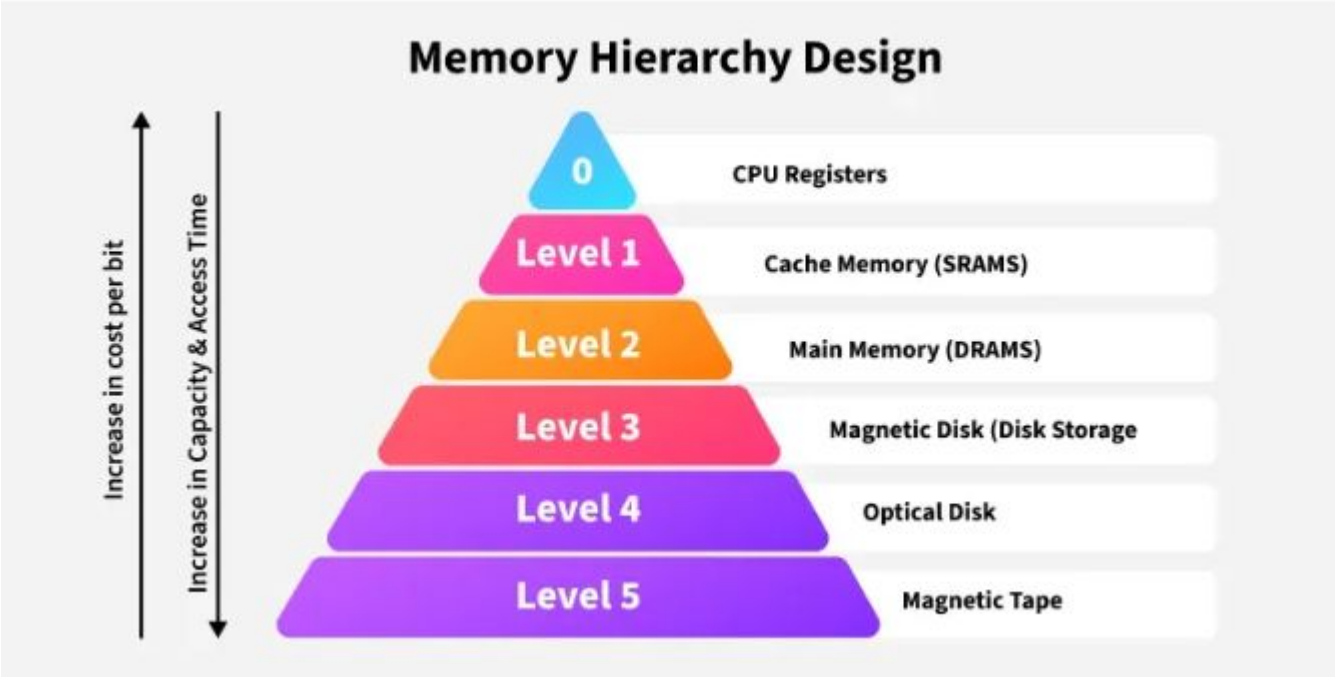
- OS provides device drivers to control these devices, using techniques like Direct Memory Access (DMA) for efficient data transfer and strategies like buffering and spooling to ensure smooth operation.




Protection and Security
User Authentication
01
Access Control
02
Resource Protection
04
Security Against Attacks
03

Kernel

- **Core of the OS**, responsible for managing hardware and resources.
- Kernels are the **heart of operating systems** , managing how hardware and software communicate and ensuring everything runs smoothly



User and Kernel space and mode

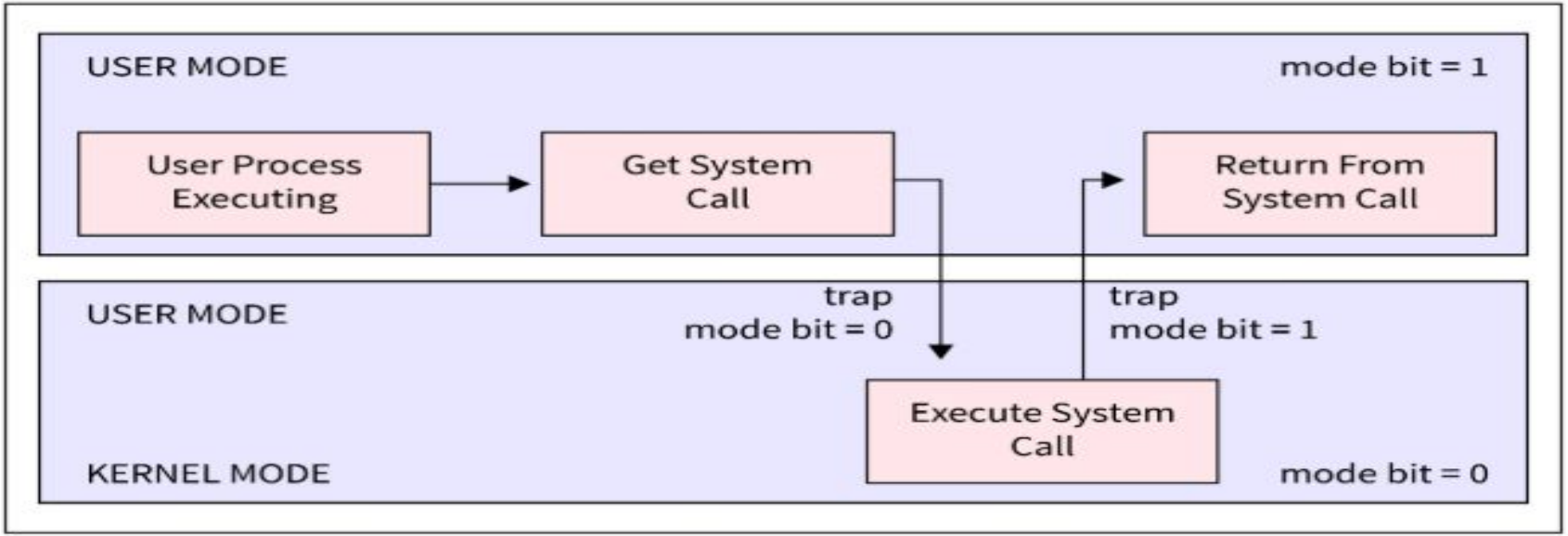
- **User mode and kernel mode are two working states** inside working system that determine the level of access and control.

  For example, if you are running MS Word, or watching some video using the VLC Player, all these software applications are running in the user mode.

- When opening the program in user mode, it is not allowed to access the RAM and hardware directly.
- To access the hardware and RAM in user mode, it sends a request to the kernel. That is the reason user mode is also known as **slave mode** or **restricted mode**.
- Running a program in user mode does not have its own address space and thus also it is unable to access the address space of the kernel.
- That is why, if there is any program failure in user mode, it does not affect the other processes. It only affects that particular process where an interrupt occurs.
- If an application running under user mode and it wants to access system resources and hardware, it will have to first go through the Operating system Kernel by using syscalls (system calls).
- The mode bit is set to 1 in user mode and while switching from user mode to kernel mode, this mode bit is set to 0.

What is Kernel Mode?

- **When we start our system, it boots in Kernel mode**. The kernel can access the hardware and RAM of the system directly but there are some privileged instructions that can run in the kernel mode only.
- These instructions are interrupt instructions, input-output management, etc. If these privileged instructions get executed under the user mode, it is not legal and it may **cause an illegal trap**.



What is Kernel Mode?

- In kernel mode, the mode bit is set to 0. And when switching from kernel mode to user mode, the bit mode is changed from 0 to 1. When the mode changes from user mode to kernel mode or vice-versa, it is known as **Context Switching**.

- Kernel is the **central module** of the operating system and it works as the **middle layer** between the operating system and the hardware of a system.

- OS maintain control over the computer system by utilizing the kernel of an operating system as a means of communication.

- The kernel handles the remaining system functions on behalf of the operating system, hence it is the **first software to load into memory** when a system boots up after the bootloader.

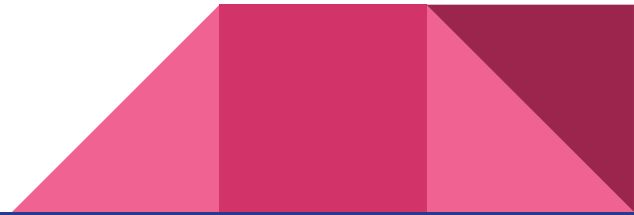
- The kernel is in the **memory of the system** until the operating system shuts down the system

Difference between User mode and Kernel mode

Aspect	User Mode	Kernel Mode
Privilege Level	Lower-privileged	Higher-privileged
Access to Hardware	Restricted	Unrestricted
Access to System Memory	Limited	Full access
Execution Environment	User-level applications	Operating System and Kernel components
Error Isolation	Processes in User Mode are isolated	Kernel manages process isolation
Purpose	Run user applications	Manage system resources and hardware
Exception Handling	Limited exception handling capabilities	Comprehensive exception handling
Stability	Application crashes do not crash OS	Kernel issues can crash the entire OS

Interrupts and System Calls

- The interrupt is a **signal emitted by hardware or software** when a process or an event needs immediate attention.
- In I/O devices one of the bus control lines is dedicated for this purpose and is called the **Interrupt Service Routine** (ISR).
- When a device raises an interrupt at let's say process i,e., the processor first completes the execution of instruction i.
- Then it loads the **Program Counter (PC)** with the address of the first instruction of the ISR. Before loading the Program Counter with the address, the address of the interrupted instruction is moved to a temporary location.
- Therefore, after handling the interrupt the processor can continue with process i+1.
- The amount of time between the generation of an interrupt and its handling is known as **Interrupt latency**



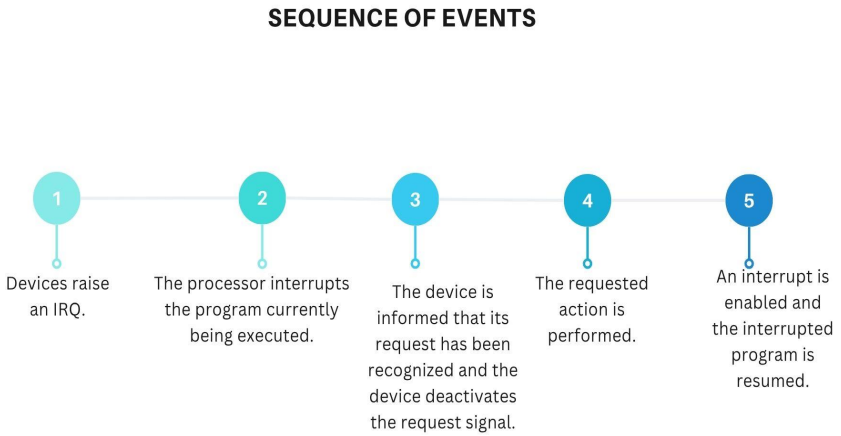
Types of Interrupt

Software Interrupt

- Traps and exceptions are other names
- They serve as a signal to carry out a certain function or respond to an error condition.
- A particular instruction known as an "**Interrupt Instruction**" is used to create software interrupts
- Generated by programs or the operating system itself.
- Used to request OS services (via system calls).
- **Example**: A program may use a software interrupt to ask the OS to read from a file.

Hardware Interrupt

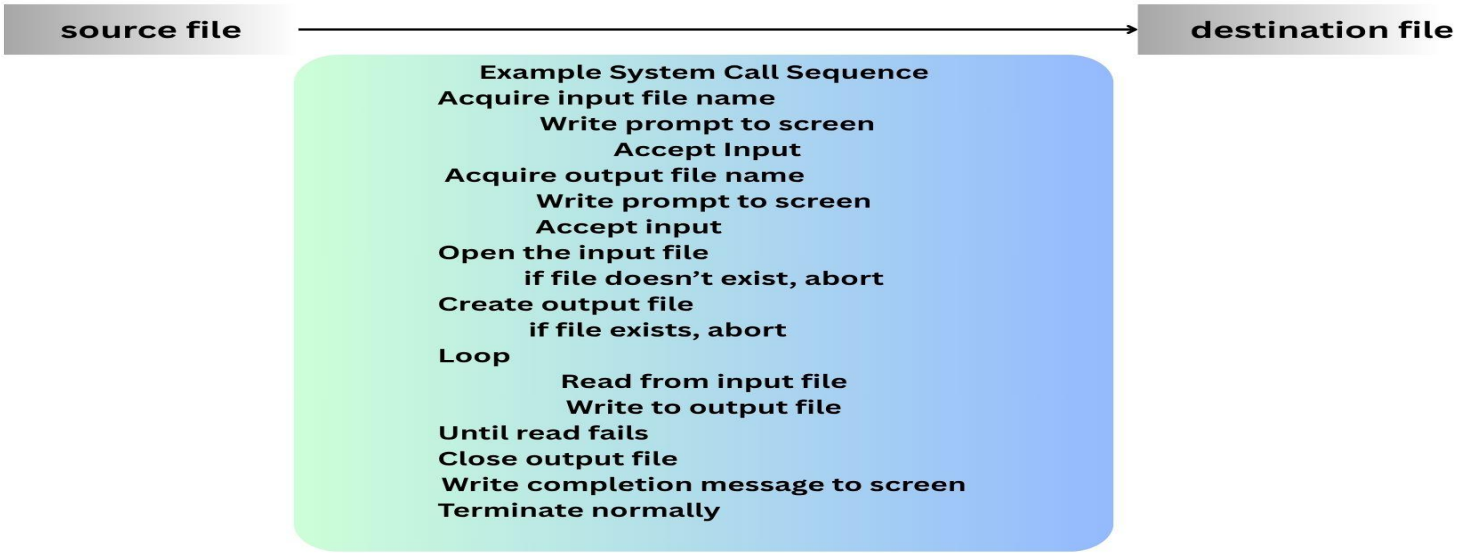
- Triggered by external devices (e.g., keyboard, mouse, timer).
- **Example**: Pressing a key sends a signal to the CPU,



System Calls

- **A System Call** is a programmatic way in which a computer program requests a service from the kernel of the operating system on which it is executed.
- A system call is a way for programs to **interact with the operating system**. A computer program makes a system call when it requests the operating system's kernel.
- System call **provides** the services of the operating system to the user programs via the Application Program Interface(API).
- **System calls are the only entry points into the kernel system and are executed in kernel mode.**

Example of System Call Sequence



System call	Description
fork()	Create process
exit()	Terminate current process
wait()	Wait for a child process to exit
kill(pid)	Terminate process pid
getpid()	Return current process's id
sleep(n)	Sleep for n seconds
exec(filename, *argv)	Load a file and execute it
sbrk(n)	Grow process's memory by n bytes
open(filename, flags)	Open a file; flags indicate read/write
read(fd, buf, n)	Read n byes from an open file into buf
write(fd, buf, n)	Write n bytes to an open file
close(fd)	Release open file fd
dup(fd)	Duplicate fd
pipe(p)	Create a pipe and return fd's in p
chdir(dirname)	Change the current directory
mkdir(dirname)	Create a new directory
mknod(name, major, minor)	Create a device file
fstat(fd)	Return info about an open file
link(f1, f2)	Create another name (f2) for the file f1
unlink(filename)	Remove a file

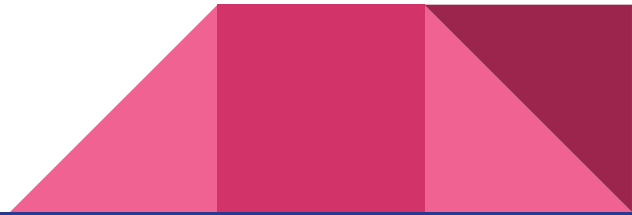
Session 2: Introduction to Linux

Lecture:

- Working basics of file system
- Commands associated with files/directories & other basic commands. Operators like redirection, pipe
- What are file permissions and how to set them?
- Permissions (chmod, chown, etc); access control list; network commands (telenet, ftp, ssh, sftp, finger)
- System variables like – PS1, PS2 etc. How to set them

Shell Programming

- What is shell; What are different shells in Linux?
- Shell variables; Wildcard symbols
- Shell meta characters; Command line arguments; Read, Echo



Introduction to Linux

- Linux is one of popular **version of UNIX**
- **Its** development began in 1991 by Linus Torvalds.

Basic Structure of a Linux System

- **Kernel** – Core of the system; manages CPU, memory,

  and devices.

- **Shell** – Interface that interprets user commands.
- **Libraries & Utilities** – Support programs for system

  functionality.

- **Applications** – End-user software (e.g., browsers,

  editors).

Popular Linux Distributions (Distros)

- A **distribution** is a complete Linux system including the kernel, tools, and applications.

Distro	Description
Ubuntu	Beginner-friendly, desktop and server
Debian	Very stable, used as a base for others
Fedora	Cutting-edge, developer-focused
Arch Linux	Lightweight, DIY approach
Kali Linux	For ethical hacking and cybersecurity
Red Hat/CentOS	Enterprise-grade, used in servers

Features of LINUX

Stable
-rarely crashes

Multi-use OS
-multiple users can access the system

Secure
-provide encryption

Compatible
-large number of file formats

Portable
-work on different types of hardware

Open Source
-free to use

Multi Programming
-multiple application can be run at same time

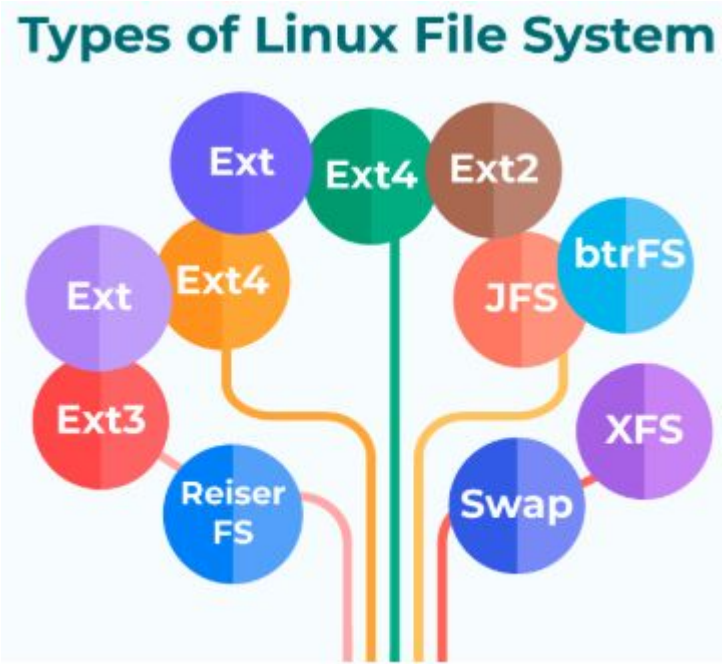
Working basics of File System

- A **File System** is a method for storing and organizing data on a computer
- One of the key features of the Linux Operating System is that everything in Linux is a file, including devices, programs, and system information
- Linux File System is a **hierarchical, tree-like and organized structure** that starts with the root directory (/), which contains all other directories and files and branches out into subdirectories as needed and makes complicated systems can be structured logically and organized
- Even the most **basic commands** such as ls and cat are also files, which lies inside the /bin directory, which **itself is also a file**
- The Linux file system structure also **provides an API (Application Programming Interface)** that allows applications to interact with the file system.

What does API do?

- The API provides a set of functions and commands that allow applications to create, modify, and delete files and directories, as well as to read and write data to and from storage devices.

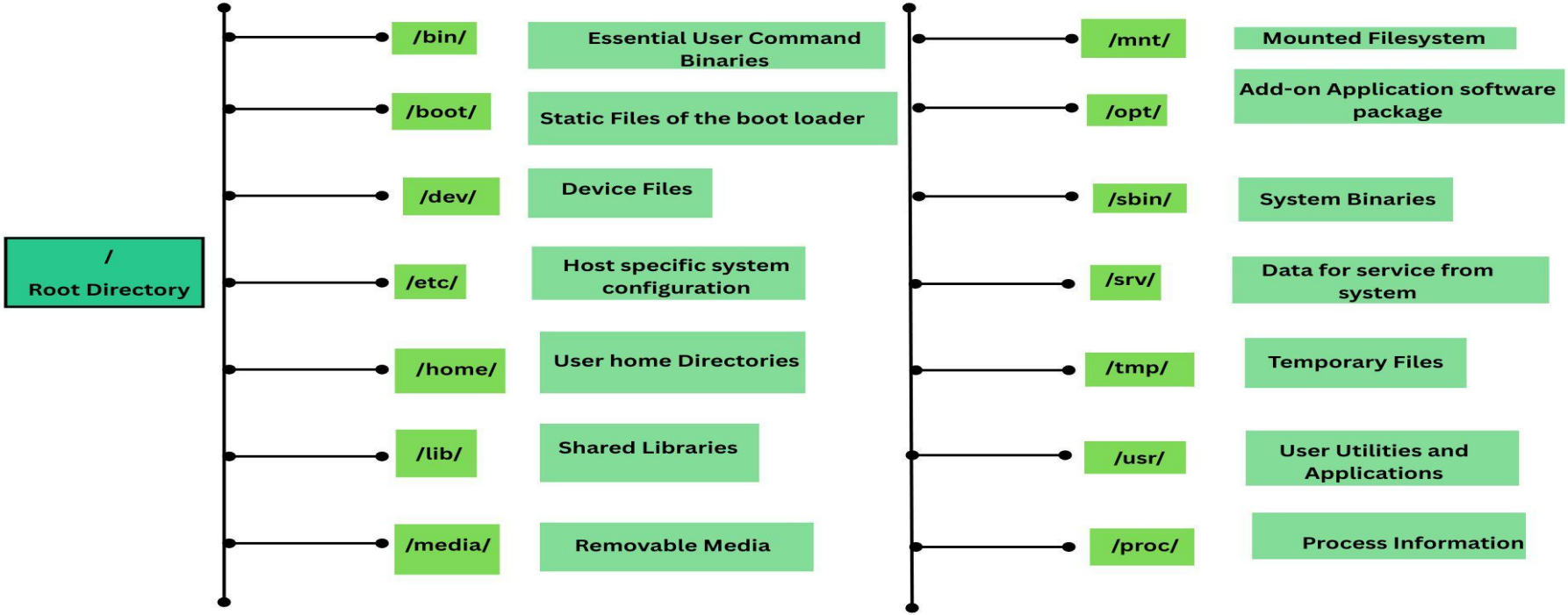
- Linux uses different file systems such as ext4, XFS, Btrfs, JFS, and ZFS to manage and store data on storage devices.
    - ext2 - USB drives, legacy systems
    - ext3 - Older Linux systems
    - ext4 - Default on modern Linux
    - XFS - Servers, large files
    - btrfs - Backups, snapshots, containers



Types of File System in Linux

```
cdac@cdac-HP-ProBook-440-G8-Notebook-PC:~$ cd ..
cdac@cdac-HP-ProBook-440-G8-Notebook-PC:/home$ cd ..
cdac@cdac-HP-ProBook-440-G8-Notebook-PC:/$ ls
 bin    cdrom   etc    lib     lib64    lost+found   mnt   proc   run    snap   swapfile  'System Volume Information'   usr
 boot   dev     home   lib32   libx32   media        opt   root   sbin   srv    sys        tmp                          var
cdac@cdac-HP-ProBook-440-G8-Notebook-PC:/$ cd /dev
cdac@cdac-HP-ProBook-440-G8-Notebook-PC:/dev$ ls
acpi_thermal_rel  gpiochip0  kvm      loop26        mcelog     ptmx      tty13  tty32  tty51     ttyS11  ttyS30  vcs2          vhci
autofs            hidraw0    log      loop27        media0     ptp0      tty14  tty33  tty52     ttyS12  ttyS31  vcs3          vhost-net
block             hpet       loop0    loop28        mei0       pts       tty15  tty34  tty53     ttyS13  ttyS4   vcs4          vhost-vsock
btrfs-control     hugepages  loop1    loop29        mem        random    tty16  tty35  tty54     ttyS14  ttyS5   vcs5          video0
bus               hwrng      loop10   loop3         mqueue     rfkill    tty17  tty36  tty55     ttyS15  ttyS6   vcs6          video1
char              i2c-0      loop11   loop30        mtd0       rtc       tty18  tty37  tty56     ttyS16  ttyS7   vcsa          vmci
console           i2c-1      loop12   loop31        mtd0ro     rtc0      tty19  tty38  tty57     ttyS17  ttyS8   vcsa1         vsock
core              i2c-10     loop13   loop32        net        shm       tty2   tty39  tty58     ttyS18  ttyS9   vcsa2         zero
cpu               i2c-11     loop14   loop33        ng0n1      snapshot  tty20  tty4   tty59     ttyS19  udmabuf vcsa3         zfs
cpu_dma_latency   i2c-12     loop15   loop34        null       snd       tty21  tty40  tty6      ttyS2   uhid    vcsa4
cuse              i2c-2      loop16   loop35        nvme0      stderr    tty22  tty41  tty60     ttyS20  uinput  vcsa5
disk              i2c-3      loop17   loop36        nvme0n1    stdin     tty23  tty42  tty61     ttyS21  urandom vcsa6
dma_heap          i2c-4      loop18   loop37        nvme0n1p1  stdout    tty24  tty43  tty62     ttyS22  userfaultfd vcsu
dri               i2c-5      loop19   loop4         nvme0n1p2  tpm0      tty25  tty44  tty63     ttyS23  userio  vcsu1
drm_dp_aux0       i2c-6      loop2    loop5         nvme0n1p3  tpmrm0    tty26  tty45  tty7      ttyS24  v4l     vcsu2
drm_dp_aux1       i2c-7      loop20   loop6         nvme0n1p4  tty       tty27  tty46  tty8      ttyS25  vboxdrv vcsu3
ecryptfs          i2c-8      loop21   loop7         nvme0n1p5  tty0      tty28  tty47  tty9      ttyS26  vboxdrvu vcsu4
fb0               i2c-9      loop22   loop8         nvram      tty1      tty29  tty48  ttyprintk ttyS27  vboxnetctl vcsu5
fd                initctl    loop23   loop9         port       tty10     tty3   tty49  ttyS0     ttyS28  vboxusb vcsu6
full              input      loop24   loop-control  ppp        tty11     tty30  tty5   ttyS1     ttyS29  vcs     vfio
fuse              kmsg       loop25   mapper        psaux      tty12     tty31  tty50  ttyS10    ttyS3   vcs1    vga_arbiter
cdac@cdac-HP-ProBook-440-G8-Notebook-PC:/dev$ cd input/
cdac@cdac-HP-ProBook-440-G8-Notebook-PC:/dev/input$ ls
 by-path  event1   event11  event13  event15  event3  event5  event7  event9  mouse0
event0    event10  event12  event14  event2   event4  event6  event8  mice    mouse1
cdac@cdac-HP-ProBook-440-G8-Notebook-PC:/dev/input$
```

Linux Directory



Commands associated with files/directories & other basic commands, Operators like redirection, pipe

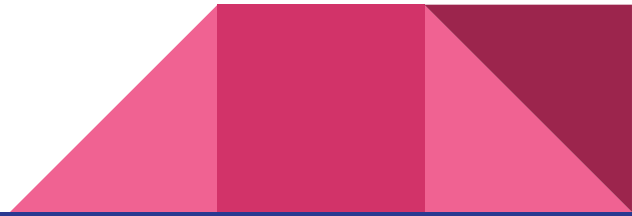
A directory can be thought of as a **virtual container that holds files and other directories within it**.

- **/ (root directory):**
  The root directory is the top-level directory in the Linux file system. All other directories and files are contained within the root directory.
- **/bin:**
  The /bin stands for binaries. This directory contains essential command-line tools and programs that are required for basic system administration tasks.
- **/etc:**
  The /etc directory contains system configuration files that are used by various applications and services on the system.
- **/home:**
  The /home directory contains the home directories of users on the system. Each user has their own subdirectory within /home where they can store their personal files and settings.
- **/opt:**
  The /opt directory is used to store additional software packages that are not part of the core system.
- **/tmp:**
  The /tmp directory contains temporary files that are created by applications and services running on the system.
- **/usr:**
  The /usr directory contains user-level programs, libraries, documentation, and shared data files.
- **/var:**
  The /var directory contains variable data files that change frequently, such as log files and system databases.

Commands associated with files/directories & other basic commands

Command	Description	Example
ls	List files and directories	ls -l, ls -a
cd	Change directory	cd Documents/
pwd	Print current directory path	pwd
mkdir	Create a new directory	mkdir my_folder
rmdir	Remove an empty directory	rmdir old_folder
rm	Delete a file or directory	rm file.txt, rm -r folder/
cp	Copy file or directory	cp a.txt b.txt, cp -r dir1 dir2

Command	Description	Example
echo	Print text to terminal or file	echo "Hello"
man	Show manual/help for commands	man ls



Command	Description	Example
mv	Move or rename file/directory	mv old.txt new.txt
touch	Create a new empty file	touch notes.txt
stat	Show file or directory details	stat file.txt
cat	Display file content	cat file.txt
head	Show first 10 lines of a file	head file.txt
tail	Show last 10 lines of a file	tail file.txt
chmod	Change file permissions	chmod 755 script.sh
chown	Change file ownership	sudo chown user file.txt
find	Find files/directories by name or type	find /home -name "*.txt"
locate	Fast search using database	locate myfile.pdf

Redirection Operators		
Command	**Description**	**Example**
> (Output Redirection)	Redirects standard output to a file, overwriting its contents	echo "Hello" > file.txt -file.txt is replaced with the text “Hello”
>>(Append Output)	Redirects standard output to a file, appending to its contents	echo "Hello again" >> file.txt -Adds “more text ” to the end of file.txt
< (Input Redirection)	Redirects standard input from a file	sort < file.txt
Pipe Operator (|)	Used to chain multiple commands together, passing the output of one as input to another. ls -l | grep "txt"	ls -l | grep "txt"
cat file.txt | sort | uniq > sorted.txt		Reads file.txt Sorts the lines Sorts the lines Saves to sorted.txt

What are file permissions and how to set them?

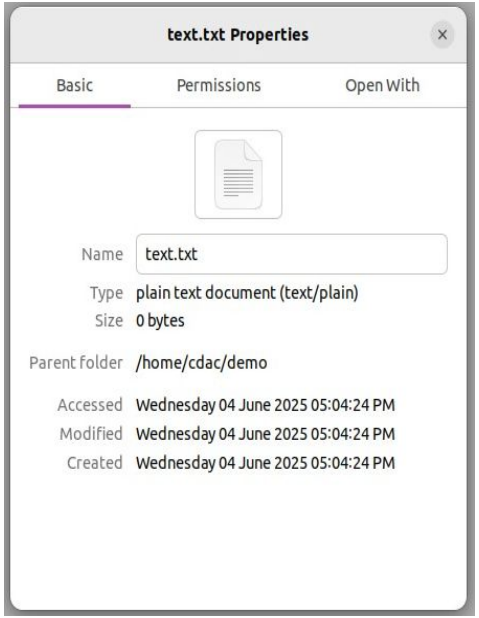
There are **two ways to check the permissions**:

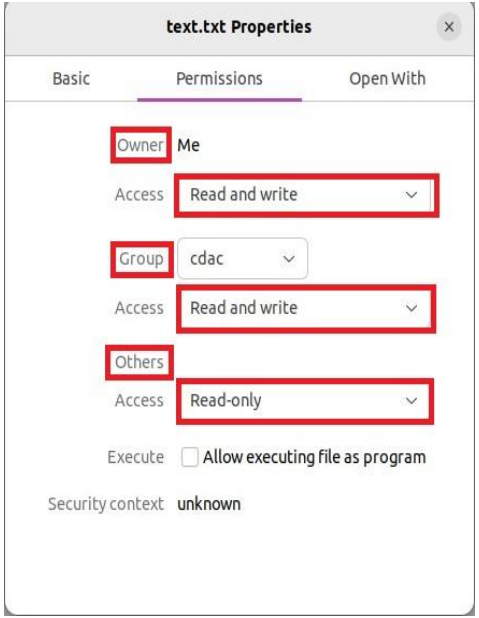
1) Using the graphical user interface (GUI)
2) The command-line interface (CLI)

Permissions tab shows the permissions for each file **divided into three categories**:

- Owner (the user who created the file/directory).
- Group (which the owner belongs to).
- Others (all other users).

1) Check Permissions Using GUI





2) The command-line interface (CLI)

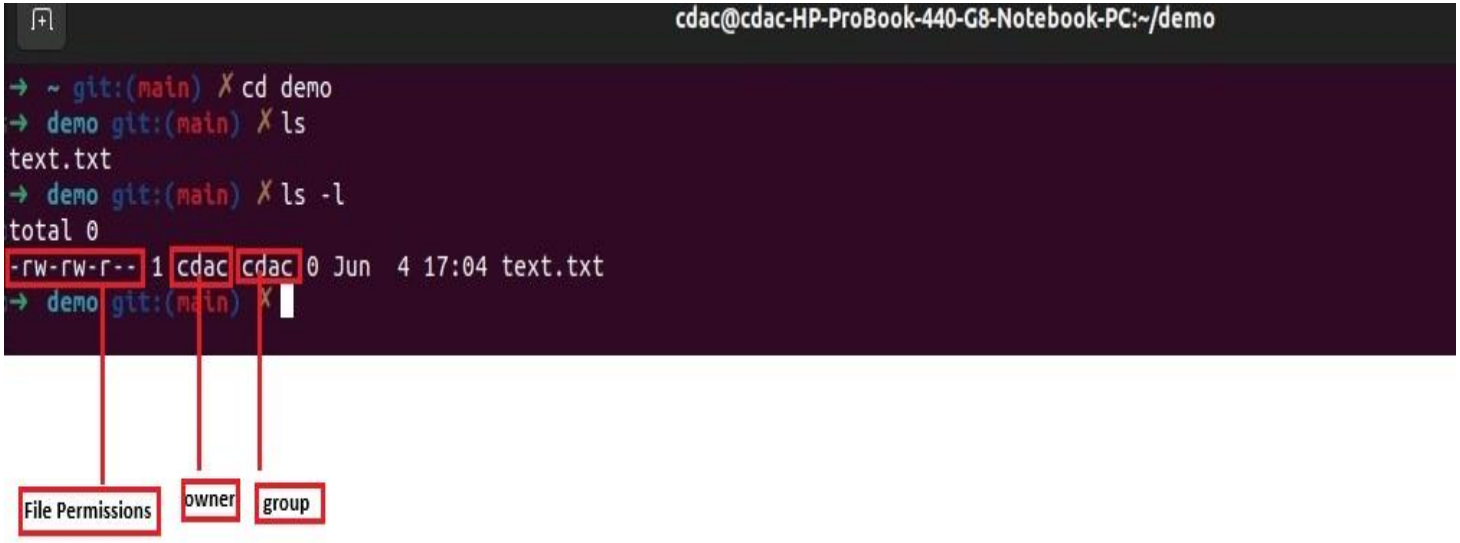
- Use the **ls command to list information about files/directories**
- Each category has three permission types: read (r), write (w), and execute (x)

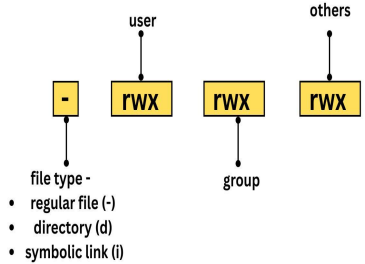
Permission Types

- **Read. (r) -** The read permission allows users to view the contents of a file or list the contents of a directory.
- **Write. (w) -** The write permission allows users to modify a file's contents or add, remove, or rename files within a directory.
- **Execute. (x) -** The execute permission allows users to execute a file or traverse (i.e., enter) a directory. For files, execute permission is required to run the file as a **program** or **script**. For directories, execute permission is required to access the contents of the directory.

Check Permissions in Command-Line with ls Command

- Use the **ls command** to list information about files/directories.
- You can also add the -l option to the command to see the information in a long list format.





Permissions

How to give permissions to file?

- **Syntax** - chmod permissions filename

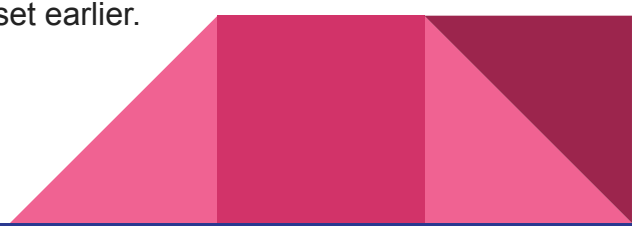
  Where permissions can be read, write, execute or a combination of them. filename is the name of the file for which the permissions need to change.

How to change permissions?

- We can change permissions using two modes:
  1. **Symbolic mode**: This method uses symbols like u, g, o to represent users, groups, and others. Permissions are represented as r (read), w (write), x (execute). You can modify permissions using +, - and =.

Example: chmod u+x filename

+ → Adds a permission to a file or directory
– → Removes the permission
= → Sets the permission if not present before. Also overrides the permissions if set earlier.



2. Absolute mode:

- This method represents permissions as 3-digit octal numbers ranging from 0-7 to represent permissions and mathematical operators to modify.

  Ex - chmod ugo+rwx file_name → chmod 777 file_name

  In above, both of them provide full read ,write and execute permission to all the group

Permissions (chmod, chown, etc)

- **chmod** - change file/directory permissions

  Symbolic Mode

  Ex - chmod u+x file.sh → Add execute to user
  chmod g-w report.txt→Remove write from group
  chmod 0=r file.txt → Set others to read-only

  chmod 775 file.sh →User: rwx (7), Group: r-x (5) , Others: r-x (5)

- **chown** - **Change Ownership**

  Ex - chown username file.txt
  chown user:group file.txt

- **chgrp** - **Change Group**

  Ex - chgrp developers file.txt

Octal	Binary	File Mode
0	000	- - -
1	001	- - x
2	010	- w -
3	011	- w x
4	100	r - -
5	101	r- x
6	110	rw-
7	111	rwx

Access Control List (ACL)

- ACL allows **setting individual permissions for multiple users/groups**

  Set ACL - setfacl -m u:john:rwx file.txt → Give ‘john’ full access

  setfacl -m g:staff:rw file.txt → Group ‘staff’ read-write

  View ACL - getfacl file.txt

  Remove ACL - setfacl -x u:john file .txt

  setfacl -b file.txt → Remove all ACLs

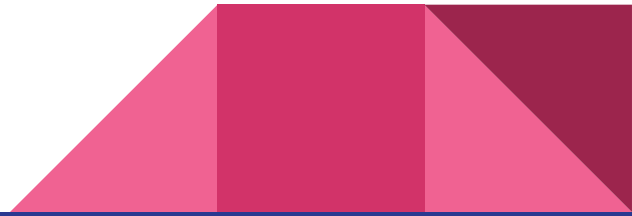
Network Commands

Command	Purpose	Example
telnet	Connect to remote system (insecure)	telnet example.com 23
ftp	Transfer files(unsecure)	ftp ftp.example.com
ssh	Secure remote login (encrypted)	ssh user@host
sftp	Secure FTP over SSH	sftp user@host
finger	View user info (login, shell,etc.,)	finger username

System variables like – PS1, PS2 etc.

- In Linux, **System variables** like PS1, PS2, and others are **environment variables** used to configure various aspects of the shell environment.

Variable	Description
PS1	Primary prompt string (default command prompt)
PS2	Secondary prompt string (used for multi-line commands)
PS3	Prompt for select command (used in shell scripts)
PS4	Used for debugging (shown when running with set -x)



How to set them System Variables?

Modify Temporarily (Only for Current Session)

You can set or change them directly in the shell:

```
PS1="[\u@\h \W]\$ "     # Example custom prompt
PS2="> "                # Change secondary prompt
```

Make the Change Permanent:

Edit the ~/.bashrc

```
                        nano ~/.bashrc
Add or modify lines like:   export PS1="[\u@\h \W]\$ "
```

After saving, apply changes with: source ~/.bashrc

Ex - export PS1="\[\e[1;32m\]\u@\h:\w\$ \[\e[0m\]"

The above will make the prompt green with

user@host:path format.

Sequence	Meaning
\u	Username
\h	Hostname
\w	Current working directory
\W	Basename of the current directory
\t	Current time
\d	Date
\$	$ for normal user, # for root

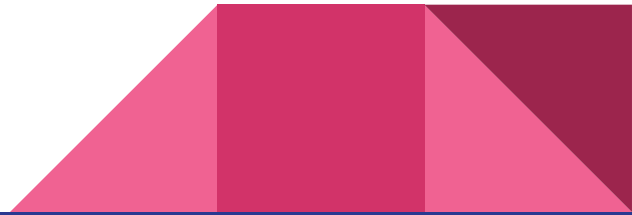
Shell Programming

What is Shell?

- If we are using any major operating system, we are indirectly interacting with the **shell**.
- While running any Linux distribution, we are interacting with the shell by using the terminal
- A **shell** in Linux is a **command-line interpreter** that allows users to interact with the operating system.
- It takes input from the user, processes it and passes it to the kernel for execution.

It can be used to:

- Run programs
- Manage files
- Execute system commands
- Automate tasks with shell scripts



What are different shells in Linux?

- Linux supports multiple shells. Each has its own features and syntax, but they all serve the same core purpose.

Bourne shell − If you are using a Bourne-type shell, the **$** character is the default prompt.		
Shell	**Description**	**Command to use**
Bash (Bourne Again Shell)	Most commonly used shell; default in most Linux distros	bash
Sh (Bourne Shell)	Original Unix shell; simple and portable	sh
Zsh (Z Shell)	Advanced shell with better scripting, plugins, and completion	zsh
Ksh (Korn Shell)	Combines features of sh and csh, used in enterprise systems	ksh

C shell − If you are using a C-type shell, the % character is the default prompt		
Shell	**Description**	**Command to use**
Csh (C Shell)	C-like syntax; less commonly used today	csh
(TENEX C Shell) tcsh	Enhanced version of csh with command-line editing	tcsh

Shell Variables

- In shell scripts, variables act as containers for holding strings and they do not possess memory addresses.
- Variables in shell scripts are mostly used for referring and altering data within the script.

Examples

Variable Names:

- A variable name could contain any alphabet (a-z, A-Z), any digits (0-9), and an underscore (_).
- However, a variable name must start with an alphabet or underscore.
- It can never start with a number.
- Shell variables are named in UPPERCASE by convention.

Note: It must be noted that no other special character such as !,*,- except underscore can be used in a variable name because all other special characters have special meanings in Shell Scripting

Ex: **Valid Variable Names -** ABC, !ABD, $ABC **Invalid variable names -** 2_AN, _AV_3, AV232, &QAID

Defining Variables:

- We use the equals symbol (=) to declare a variable in Linux.

Syntax: variable_name=<variable data>

Ex - my_message=”Hello World”

- Note that there must be no spaces around the **"="** sign

Accessing Variable

- Variable data could be accessed by appending the variable name with '$' as follows:

```
VAR_1="Devil"
VAR_2="OWL”
echo "$VAR_1$VAR_2"
```

Output: DevilOWL

Unsetting Variables

- The unset command directs a shell to delete a variable and its stored data from list of variables.
- It can be used as follows:

```
var1="Devil"
var2=23
echo $var1 $var2
unset var1
echo $var1 $var2
```

Output:

```
DEVIL 23
23
```

Read only Variables

- These variables are read only i.e., their values could not be modified later in the script

```
var1="Devil"
var2=23
readonly var1
echo $var1 $var2
var1=23
echo $var1 $var2
```

Output : Devil 23

```
./bash1: line 8: var1: readonly variable
Devil 23
```

```
#!/bin/bash
#variable definitions
Var_name="Devil"
Var_age=23

# accessing the declared variables using $
echo "Name is $Var_name, and age is $Var_age."

# read-only variables
var_blood_group="O-"
readonly var_blood_group
echo "Blood group is $var_blood_group and read only."
echo "Error for read only variables, if trying to \
modify them."
echo
var_blood_group="B+"
echo

# unsetting variables
unset Var_age
echo "After unsetting var_age..."
echo
echo "Name is $Var_name, blood group is $var_blood_group\
 and age is $Var_age..."
```

Variable Types

1) Local Variable:

- A local variable is a variable that is present within the current instance of the shell.
- Local variables is temporary storage of data within a shell script.
- It is not available to programs that are started by the shell. They are set at the command prompt.
  For Example: `name=Jayesh`

  In this case the local variable is (name) with the value of Jayesh.

2) **Environment Variables**:

- These variables are commonly used to configure the behavior script and programs that are run by shell.
- Environment variables are only created once, after which they can be used by any user.

  For Example: `export PATH=/usr/local/bin:$PATH` would add `/usr/local/bin` to the beginning of the shell's search path for executable programs.

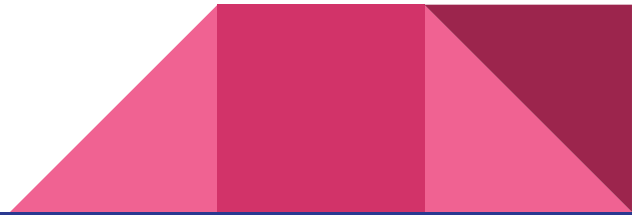
Shell Variables −It is a special variable that is set by the shell and is required by the shell in order to function correctly.

Some of these variables are environment variables whereas others are local variables.

For Example: `$PWD` = Stores working directory

`$HOME` = Stores user's home directory

`$SHELL` = Stores the path to the shell program that is being used.



Wildcard Symbols in Linux Shell

- Wildcards are special characters used in the shell to represent **one or more characters** in file and directory names.
- They're especially useful in commands like ls, cp, mv, rm, etc.
- Wildcards are also called **globs**.

Wildcard	Meaning / Matches	Example
*	Matches **zero or more** characters	ls *.txt (all .txt files)
?	Matches **exactly one** character	ls file?.txt (e.g., file1.txt, fileA.txt)
[]	Matches **any one character** inside brackets	ls file[123].txt (matches file1.txt, file2.txt, file3.txt)
[^]	Matches **any one character not** in brackets	ls file[^1].txt (matches all except file1.txt)
{}	Matches **a comma-separated list** of strings	ls {file1,file2}.txt (matches both file1.txt and file2.txt)
~	Represents the **home directory**	cd ~ (goes to home folder)
\	**Escapes** the next character (treat as normal)	echo * prints *

Shell Meta Characters in Linux

- **Shell metacharacters** are special characters that the shell interprets in a specific way to control input, output, command chaining, wildcard expansion, etc

Read & Echo

- The read and echo commands are fundamental tools for **interacting with users** in a shell script.
- read – takes **input** from the user.

Syntax: read [variable_name]

Ex - echo "Enter your name:"

read name

echo "Hello, $name!"

Output: Enter your name: Vimal

Hello, Vimal!

- **read with Multiple Variables:** echo "Enter two values:"

read a b

echo "First: $a, Second: $b"

- **read with Prompt (Using -p flag)**

  read -p "Enter your course name: " course

  echo "You are learning $course"

- **Silent Input (Password style) with -s**

  read -sp "Enter password: " password

  echo

  echo "Password received."

  echo – displays output to the terminal.

  Newline (-e) - echo -e "Line1\nLine2"

  No newline (-n) - echo -n "Same line "