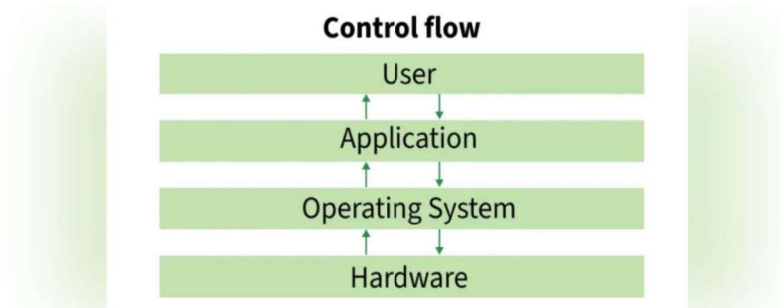


Introduction to Operating System: An operating system acts as an intermediary between the user of a computer and computer hardware. In short, it serves as an interface between computer hardware and user.



This flow shows how **commands, actions, and responses** move through the layers of a computer system — **from the user to the hardware** and back. The arrows represent **control, data flow, and interaction** between each layer.

User:

1. The **user** is a human interacting with the computer system through a **user interface** (e.g., keyboard, mouse, touchscreen, etc.).
2. The user gives input or commands — such as opening a file, typing a message, or clicking a button.
3. The input moves **down** to the **Application** layer. Eg. A user clicks the “Save” button in a text editor.

Application: This is the **software** that users interact with directly — like Microsoft Word, Google Chrome, Photoshop, etc.

1. The application interprets the user's actions and decides what it needs from the underlying system to fulfill the request.
2. The application communicates with the **Operating System (OS)** to get access to system resources like memory, disk, network, etc.
3. Once the OS completes the task, results go **back up** to the application, which then shows the result to the user. Eg: The text editor sends a request to the OS to write the contents of a document to disk.

Operating System: The **OS** (like Windows, Linux, macOS) is a **middleman** between applications and hardware. It manages resources and provides essential services.

1. It receives requests from applications and decides **how to allocate system resources** (e.g., CPU time, memory, file access). It then interacts with the hardware to complete tasks.
2. The OS sends instructions to the **Hardware**, such as writing data to a disk or sending information over the network.
3. The hardware sends a status or result back to the OS, which then relays it to the application.
4. The OS writes the file to the hard drive and confirms the file has been saved successfully.

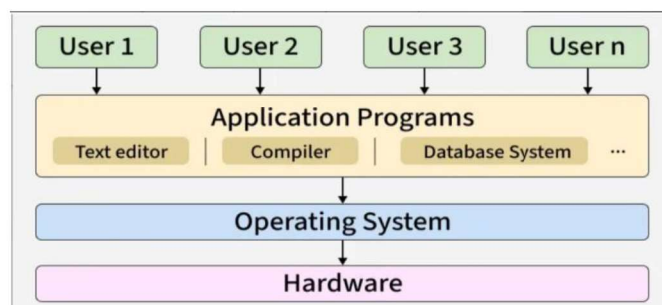
Hardware: This is the **physical layer** — your actual computer parts: CPU, RAM, hard drive, GPU, motherboard, etc.

1. The hardware executes the **low-level operations** requested by the OS — such as storing data, displaying pixels, or processing computations.
2. Hardware doesn't initiate action — it **responds** to commands from the OS.
3. Once the task is done, the hardware sends signals back to the OS confirming the operation.
4. The hard disk physically writes the document data to a specific sector and signals completion.

Operating System and User Interaction

Every general-purpose computer consists of hardware, an operating system, system programs and application programs.

- The hardware includes CPU, ALU, memory, I/O devices, peripheral devices and storage devices.
- The system programs include compilers, loaders, editors and the OS itself.
- The application programs are user-level software that perform specific tasks.



An operating system is always running in the background to manage and co-ordinate the use of hardware resources among system and application programs for multiple users. The OS is not itself just a user interface. Instead, it provides an interface for user interaction, typically through:

- Command-Line Interface (CLI), via shells and terminal commands.
- Graphical User Interface (GUI), through windows, icons and menus.

Goals of Operating System

Primary Goals

The primary goals of an operating system (OS) are to provide a easy to use and convenient environment for executing user programs.

1. **User Convenience** :It should be easy to use, providing a user-friendly interface and making it simple to interact with the system.
2. **Program Execution**: It facilitates the execution of user programs, providing the necessary environment and services for them to run.
3. **Resource Management**: The OS manages and allocates the computer's resources, including the CPU, memory, disk storage and input/output devices, to ensure fair utilization.
4. **Security**: The OS protects the system and user data from unauthorized access, ensuring the confidentiality, integrity and availability of information.

Secondary Goals

1. **Efficient Resource Utilization**: It should aim to maximize the performance and utilization of computer resources like CPU, Memory and IO devices, ensuring that the system runs smoothly and efficiently.
2. **Reliability**: It should be robust and reliable, able to handle errors and exceptions gracefully, ensuring that the system continues to operate smoothly. It should be modular in design and easy to debug.

Components of an Operating Systems

There are two basic components of an Operating System.

- Shell is the outermost layer of the Operating System and it handles the interaction with the user. It interprets input for the OS and handles the output from the OS.
- Kernel is the core component of operating system. The kernel is the primary interface between the Operating system and Hardware.

List of Common Operating Systems

There are multiple types of operating systems each having its own unique features:

- **Windows OS**: Developed by Microsoft. Used in personal computing, Business environment, Gaming.
- **macOS**: Developed by Apple. Used for creative industries (design, video editing, music production), Personal computing and Professional environments.

- **Linux:**Community-driven (various distributions). There are large selection of distributions (e.g., Ubuntu, Fedora, Debian). It is free to use and distribute. Used in servers and data centers, Development and programming, Personal computing for tech enthusiasts.
- **Unix:**Developed originally AT&T Bell Labs, various commercial and open-source versions available. Used in servers and workstations, Development environments, Research and academic settings

List of Common Operating Systems

There are multiple types of operating systems each having its own unique features:

- **Windows OS:** Developed by Microsoft. Used in personal computing, Business environment, Gaming.
- **macOS :**Developed by Apple. Used for creative industries (design, video editing, music production), Personal computing and Professional environments.
- **Linux:**Community-driven (various distributions). There are large selection of distributions (e.g., Ubuntu, Fedora, Debian). It is free to use and distribute. Used in servers and data centers, Development and programming, Personal computing for tech enthusiasts.
- **Unix:** Developed originally AT&T Bell Labs, various commercial and open-source versions available. Used in servers and workstations, Development environments, Research and academic settings

History of Operating System: The operating system has been evolving through the years. The following table shows the history of OS.

Era	Key Developments	Examples
1956	The first Operating System was GM-NAA I/O in 1956 by General Motors.	GM-NAA I/O (1956)
1960s	IBM developed a time sharing system TSS/360	OS/360, DOS/360 and TSS/360
1970s	Unix popularized simplicity and multitasking; rise of personal computers with basic OSs.	Unix (1971), CP/M (1974)
1980s	GUI-based OSs gained traction; networking features became standard.	Apple Macintosh (1984), Windows (1985)

1990s	Open-source Linux emerged; GUIs in Windows and Mac OS improved.	Linux (1991), Windows 95 (1995)
2000s- Present	Mobile OSs dominated; cloud and virtualization technologies advanced computing.	iOS (2007) android (2008)

Q) System Components?

An Operating system is an interface between users and the hardware of a computer system. It is a system software that is viewed as an organized collection of software consisting of procedures and functions, providing an environment for the execution of programs. The operating system manages system software and computer hardware resources. It allows computing resources to be used in an efficient way. Programs interact with computer hardware with the help of operating system. A user can interact with the operating system by making system calls or using OS commands.

Important Components of the Operating System:

- Process Management
- File Management
- Command Interpreter
- System Calls
- Signals
- Network Management
- Security Management
- I/O Device Management
- Secondary Storage Management
- Main Memory Management

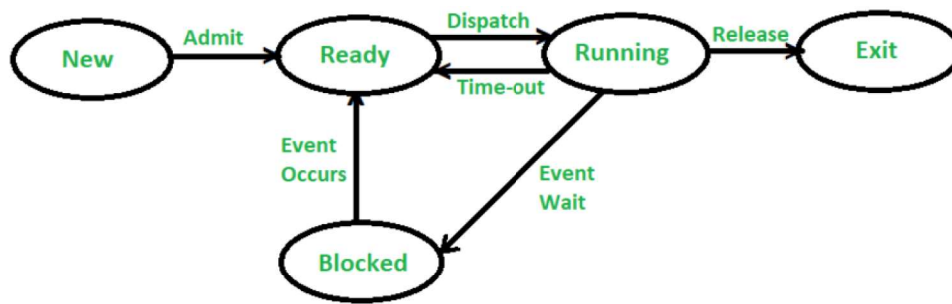
Process Management: A process is a program in execution. It consists of the followings:

Executable program
 Program data
 Stack and stack pointer
 Program counter and other CPU registers
 Details of opened files

A process can be temporarily paused (suspended) so that the operating system can run another process. The suspended process can be started again later. Before suspending a process, the OS saves its details in a special table called the process table, so it can continue later from where it stopped.

The operating system provides two important **system calls** to manage processes:

- **Create** – to start a new process
- **Kill** – to stop or end a process



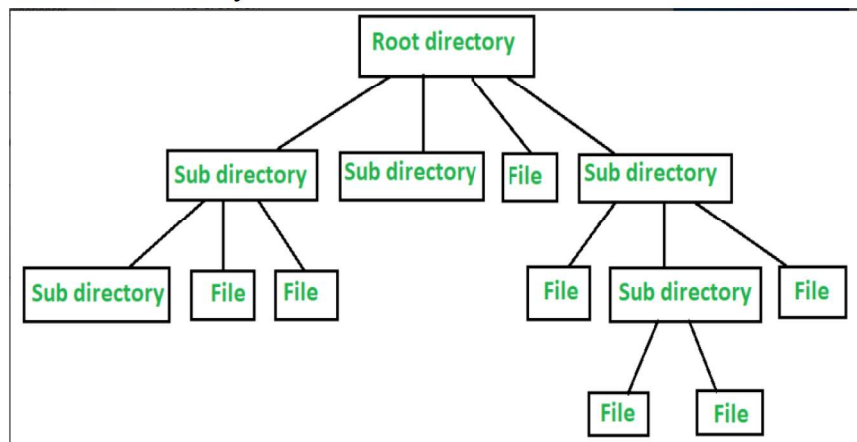
Files Management: Files are used for long-term storage. Files are used for both input and output. Every operating system provides a file management service. This file management service can also be treated as an abstraction as it hides the information about the disks from the user. The operating system also provides a system call for file management. The system call for file management includes:

File creation

File deletion

Read and Write operations

Files are stored in a directory. System calls provide to put a file in a directory or to remove a file from a directory. Files in the system are protected to maintain the privacy of the user. Below shows the Hierarchical File Structure directory.



Command Interpreter: There are several ways for users to interface with the operating system. One of the approaches to user interaction with the operating system is through commands. Command interpreter provides a command-line interface. It allows the user to enter a command on the command line prompt (cmd). The command interpreter accepts and executes the commands entered by a user. For example, a shell is a command interpreter under UNIX. The commands to be executed are implemented in two ways:

- ✓ The command interpreter itself contains code to be executed.
- ✓ The command is implemented through a system file. The necessary system file is loaded into memory and executed.

System Calls: System calls provide an interface to the services made by an operating system. The user interacts with the operating system programs through System calls. These calls are normally made available as library functions in high-level languages such as C, Java, Python etc. It provides a level of abstraction as the user is not aware of the implementation or execution of the call made. Details of the operating system is hidden from the user. Different hardware and software services can be availed through system calls.

System calls are available for the following operations:

- ✓ Process Management
- ✓ Memory Management
- ✓ File Operations
- ✓ Input / Output Operations.

Signals: Signals are used in the operating systems to notify a process that a particular event has occurred. Signals are the software or hardware interrupts that suspend the current execution of the task. Signals are also used for inter-process communication. A signal follows the following pattern:

A signal is generated by the occurrence of a particular event it can be the clicking of the mouse, the execution of the program successfully or an error notifying, etc.

A generated signal is delivered to a process for further execution.

Once delivered, the signal must be handled.

A signal can be synchronous and asynchronous which is handled by a default handler or by the user-defined handler.

The signal causes temporarily suspends the current task it was processing, saves its registers on the stack, and starts running a special signal handling procedure, where the signal is assigned to it.

Network Management

The definition of network management is often broad, as network management involves several different components. Network management is the process of managing and administering a computer network. A computer network is a collection of various types of computers connected with each other.

Network management comprises fault analysis, maintaining the quality of service, provisioning of networks, and performance management.

Following are the features of network management:

- Network administration
- Network maintenance

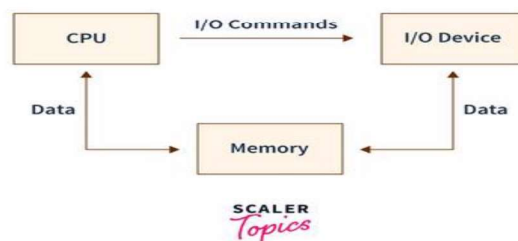
- Network operation
- Network provisioning
- Network security

Security Management: The operating system is primarily responsible for all task and activities happen in the computer system. The various processes in an operating system must be protected from each othersactivities. For that purpose, various mechanisms which can be used to ensure that the files, memory segment, CPU and other resources can be operated on only by those processes that have gained proper authorization from the operating system.

Security Management refers to a mechanism for controlling the access of programs, processes, or users to the resources defined by a computer controls to be imposed, together with some means of enforcement.

I/O Device Management

In computer I/O, the operating system is responsible for managing and controlling I/O processes and devices. Because the functions and speeds of devices attached to the computer (I/O devices) vary so considerably, multiple ways for controlling them are required. These methods make up the operating system's I/O subsystem, which shields the rest of the system from the intricacies of controlling I/O devices.



Secondary Storage Management

The main purpose of a computer system is to execute programs. These programs, together with the data they access, must be in main memory during execution. Since the main memory is too small to permanently accommodate all data and program, the computer system must provide secondary storage to backup main memory.

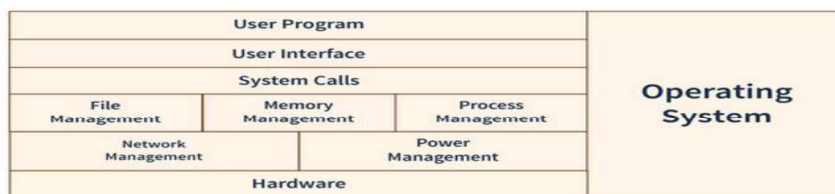
Most modern computer systems use disks as the principle on-line storage medium, for both programs and data. Most programs, like compilers, assemblers, sort routines, editors, formatters, and so on, are stored on the disk until loaded into memory, and then use the disk as both the source and destination of their processing.

The operating system is responsible for the following activities in connection with disk management:

- Free space management
- Storage allocation
- Disk scheduling

Main Memory Management

A huge array of storage or bytes with an address is referred to as main memory. A series of reads or writes of specified memory addresses is utilized in the memory management process.

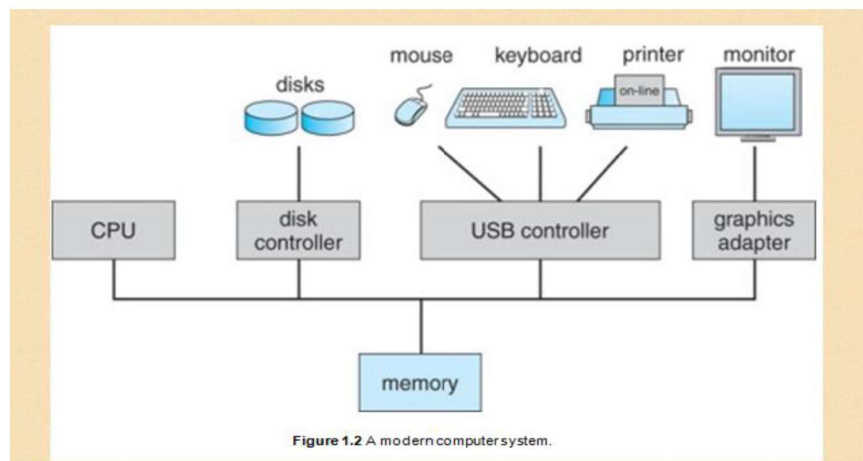


SCALER
Topics

To run programs, it must be mapped to absolute addresses and then loaded into memory. Several aspects influence memory management strategy selection. Because each algorithm requires hardware support, memory management is mostly determined by the system's hardware design. It ensures quick storage so that the CPU has direct access to it. It is expensive and has limited storage space. Regardless, a Programme must be stored in main memory in order to be executed.

Q) System Organization?

Computer Organization is a core area of computer science and engineering that deals with how the components of a computer system work together to execute programs efficiently. It focuses on the internal structure and functional units of a computer system and how they interact.



The figure shows the major components of a computer system and their interconnections. Each block represents a hardware component that plays a specific role in the functioning of the system.

1. CPU (Central Processing Unit)

- The **brain of the computer**.
- Responsible for **executing instructions, performing calculations, and controlling other components**. It consists of
 - **Arithmetic Logic Unit (ALU)** – performs arithmetic and logical operations.
 - **Control Unit (CU)** – directs the operation of all other components by sending control signals.
- **Registers** – high-speed storage for temporary data and instructions. Communicates with memory and I/O controllers through the **system bus**.

2. Memory (Main Memory or RAM) :

- Temporarily stores data, instructions, and intermediate results during program execution.
- The CPU fetches instructions and data directly from memory.
- Acts as a bridge between the CPU and secondary storage (disks).
- Faster than disk storage but volatile (data is lost when power is off).

3. Disk Controller

- A specialized circuit that manages communication between the CPU and storage disks.
- Handles data transfer, read/write operations, and error checking.
- Allows the CPU to issue simple commands like “read block” or “write block” without worrying about physical disk details.
- Connected to disks (e.g., Hard Disk Drive or SSD).

4. Disks (Secondary Storage)

- Used for **permanent data storage**.
- Store the **operating system, applications, and user data**.
- Slower than main memory but provide **large storage capacity**.
- Data is transferred between disk and memory through the **disk controller**.

5. USB Controller

- Manages **input and output devices** connected via USB ports (Universal Serial Bus).
- Controls devices such as **mouse, keyboard, printer**, and other USB peripherals.
- Transfers data between these devices and the CPU or memory.
- Handles communication protocols, buffering, and device detection.

6. Mouse

- An **input device** used to control the pointer or cursor on the screen.
- Sends signals to the CPU through the **USB controller**.
- Converts user hand movements into electrical signals that the system interprets.

7. Keyboard

- Another **input device** that allows the user to enter text and commands.
- Connected via the **USB controller**.
- Each key press generates a signal that is sent to the CPU for processing.

8. Printer

- An **output device** used to produce hard copies of documents or images.
- Communicates through the **USB controller**.
- The **Operating System** manages printing through print queues and device drivers.

9. Graphics Adapter (or Graphics Card)

- Converts data from the CPU into **visual signals** for display on the **monitor**.
- Contains its own **processor (GPU)** and **memory** for image rendering.
- Connected to the **monitor** and communicates with the CPU through the **system bus**.

10. Monitor

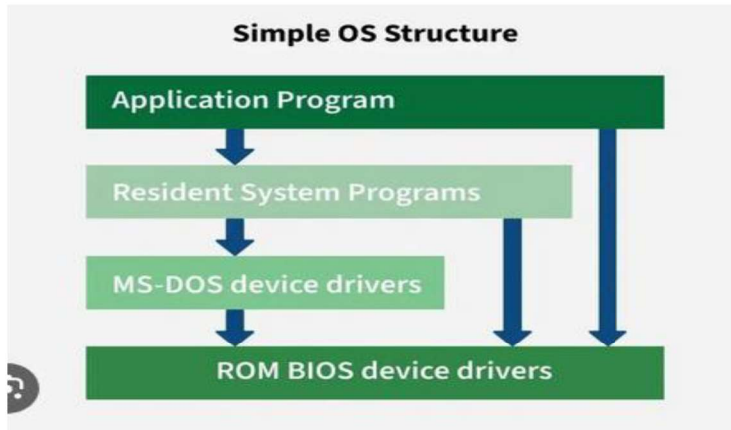
- A **display device** that shows the output of the computer system.
- Works in coordination with the **graphics adapter**.
- Displays text, images, videos, and user interface elements.

11. System Bus (Interconnection Pathway)

- The **communication channel** that connects the CPU, memory, and all controllers.
- Consists of three types of buses:
 - **Data Bus** – carries actual data.
 - **Address Bus** – carries memory addresses.
 - **Control Bus** – carries control signals.
- Ensures smooth data transfer between all components.

Q) Structure and operation and Protection and Security?

OS Structure:



This diagram illustrates a Simple Operating System (OS) Structure, commonly used to describe the architecture of MS-DOS (Microsoft Disk Operating System). It represents how different layers of software interact — from application programs at the top to the hardware-level BIOS at the bottom. Let's go through each layer in detail:

1. Application Program: These are the user-level programs that perform specific tasks, such as word processors, spreadsheets, games, or any other software.

- **Function:**
 - They request OS services (e.g., reading a file, printing a document) through system calls.
 - Communicate with the OS (and sometimes directly with the hardware in simple systems like MS-DOS).
- **Examples:**
 - Microsoft Word
 - Lotus 1-2-3
 - Turbo C compiler
- **Interaction:**
 - Sends commands to the **Resident System Programs** or directly to **MS-DOS/BIOS** for execution.

2. Resident System Programs: This layer consists of the essential parts of the operating system that are loaded into memory during boot time and remain resident.

- **Function:**
 - Provides a command interpreter (like `COMMAND.COM` in MS-DOS).
 - Offers basic system utilities (copy, delete, rename, etc.).
 - Manages memory and file systems.
- **Examples:**

- Command-line interpreter (CLI)
- File management routines
- **Interaction:**
 - Receives user or application commands.
 - Passes low-level hardware requests to **Device Drivers**.

3. MS-DOS Device Drivers: These are software modules that control and manage specific hardware devices (e.g., keyboard, disk drives, printers).

- Translate general I/O commands from the system into device-specific operations.
- Provide a consistent interface to higher layers.
- **Examples:**
 - HIMEM.SYS (memory manager)
 - ANSI.SYS (keyboard/screen control)
 - CONFIG.SYS (device configuration)
- **Interaction:**
 - Communicate directly with the **ROM BIOS device drivers** for hardware-level access.

4. ROM BIOS Device Drivers: BIOS (Basic Input/output System) is stored in the ROM (Read-Only Memory) of the computer.

- **Function:**
 - Provides the most basic hardware control functions.
 - Contains low-level routines to communicate with physical hardware (keyboard, display, disk, etc.).
 - Initializes and tests hardware components during boot-up (POST - Power On Self Test).
- **Examples:**
 - Keyboard interrupt handler
 - Disk sector read/write routines
 - Display drivers (VGA text mode)
- **Interaction:**
 - Directly interacts with **hardware**.
 - Provides services to **device drivers** and sometimes directly to **applications** (in MS-DOS).

Flow of Communication

1. **Application Program → Resident System Programs:**
User applications request operating system services.
2. **Resident System Programs → MS-DOS Device Drivers:**
OS utilities call the appropriate device drivers for hardware I/O.
3. **MS-DOS Device Drivers → ROM BIOS Device Drivers:**
Device drivers use BIOS routines for direct hardware interaction.

4. **ROM BIOS Device Drivers → Hardware:**

BIOS executes the actual hardware-level operations.

Operation

The fundamental operations of an OS are to manage system resources efficiently and securely:

- **Process Management:** Creating, scheduling, and terminating processes.
- **Memory Management:** Allocating and deallocating memory, and using virtual memory to extend physical memory.
- **File and Disk Management:** Storing and retrieving data on disk drives and managing file systems.
- **I/O Management:** Controlling input and output devices.

Protection

Protection ensures that processes and users do not interfere with each other's operations by controlling access to resources.

Principle of Least Privilege: Processes are given only the minimum permissions needed to complete their tasks.

- **Memory Protection:** Prevents a process from accessing the memory of another process.
- **CPU Protection:** Uses interrupts to prevent a process from consuming the CPU indefinitely.
- **I/O Protection:** Restricts access to I/O devices to prevent hardware damage or misuse.

Security

Security is the broader concept of protecting the system from external and malicious threats, ensuring confidentiality, integrity, and availability of data.

Authentication: Verifies a user's identity (e.g., through passwords or biometrics).

Authorization: Grants specific permissions to authenticated users for accessing resources.

Access Control: Implements rules (like Access Control Lists) to enforce authorization policies.

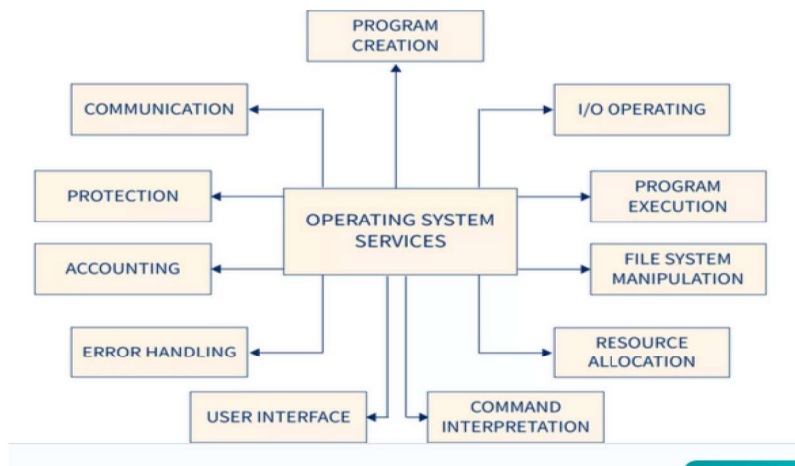
Encryption: Encodes data to protect it from unauthorized viewing, even if accessed.

Malware Protection: Uses antivirus software to detect and remove viruses, worms, and other malicious code.

System Updates: Applies patches and updates to fix vulnerabilities that attackers could exploit.

Q) Operating system Services?

Operating System (OS) services refer to the **basic functions and features provided by an operating system** to make the computer system convenient, efficient, and secure for both users and applications. In simple terms — these are the **services or facilities the OS offers** to users and application programs to perform various tasks.



1. Program Creation: The operating system provides services to create and load programs into memory for execution. It helps allocate necessary resources such as CPU time, memory space, and input/output devices. This service ensures that programs can be easily written, compiled, assembled, and linked for execution without users needing to manage hardware-level details.

2. Program Execution: Once a program is created, the OS manages its execution. It handles tasks such as loading the program into main memory, initializing execution, and monitoring the process until it completes. The OS ensures proper sequencing and execution of instructions and also handles process termination when execution finishes or an error occurs.

3. Input/output (I/O) Operations: I/O services manage input and output operations between programs and peripheral devices like keyboards, printers, or disks. The OS acts as an intermediary, providing a uniform interface so that programs do not need to know the hardware details of I/O devices. This simplifies I/O handling for application developers.

4. File System Manipulation: The OS provides file-related services such as creating, deleting, reading, writing, and organizing files. It manages directories for efficient data access and storage. Additionally, it controls permissions to ensure that only authorized users or programs can access or modify specific files.

5. Communication: This service enables processes to communicate with each other, either within the same computer or across networked systems. The OS supports communication through mechanisms like message passing, shared memory, or sockets. It ensures data is transmitted securely and reliably between processes or devices.

6. Error Handling: The OS continuously monitors the system for possible errors in hardware or software. When an error occurs—such as a memory overflow or device failure—it takes corrective actions or reports the issue to the user or process. Proper error handling ensures system stability and smooth operation.

7. Resource Allocation: Since multiple processes may need the same resources at the same time, the OS allocates CPU time, memory space, and I/O devices efficiently among them. It ensures fair distribution and optimal use of system resources, preventing conflicts or deadlocks between processes.

8. Accounting: The accounting service keeps track of system resource usage by users and processes. It maintains records of CPU usage, memory consumption, and I/O operations. This information helps in performance analysis, cost calculation, and identifying areas for system optimization.

9. Protection: Protection services ensure that all access to system resources is controlled and authorized. The OS enforces security policies to prevent unauthorized access or interference between processes. It uses mechanisms like user authentication, access control lists, and encryption to safeguard data and operations.

10. User Interface: The operating system provides an interface that allows users to interact with the computer system. This interface may be a command-line interface (CLI) or a graphical user interface (GUI). It simplifies user interaction with the system, allowing them to issue commands or manage files easily.

11. Command Interpretation: The command interpreter, also known as the shell, is responsible for interpreting and executing user commands. It acts as a bridge between the user and the operating system kernel. It interprets commands entered by the user, translates them into system calls, and displays the results or errors.

Q) Operating system interface?

There are mainly **three types** of interfaces provided by operating systems:

1. Command-Line Interface (CLI): The **Command Line Interface (CLI)** in Windows is a **text-based environment** where users type commands to perform tasks such as managing files, running programs, or configuring system settings — instead of using graphical menus and icons.

It's one of the most powerful tools in Windows for **administration, troubleshooting, and automation**.

Common Uses:

Task	Command	Description
View files	dir	Lists files and folders
Change directory	cd folder name	Opens a folder
Create folder	mkdir folder name	Makes a new directory
Delete file	del filename.txt	Deletes a file
Copy file	copy source.txt destination.txt	Copies files
Clear screen	cls	Clears terminal output
View IP details	ipconfig	Displays network info

2. Graphical User Interface (GUI): A Graphical User Interface allows users to interact with the system using windows, icons, menus, and pointers (WIMP). It is visually appealing and easy for beginners to use.

- **How it works:**
Users click on icons or buttons to perform actions such as opening files, copying text, or browsing the web. The OS translates these actions into low-level system calls.
- **Advantages:**
 - Easy to use and learn
 - More intuitive and user-friendly
 - Supports multitasking visually
- **Disadvantages:**
 - Requires more memory and processing power
 - Limited control compared to CLI
- **Examples:**
 - **Windows OS (Windows 10, 11)**
 - **macOS**
 - **Linux distributions with GUI desktops** (like Ubuntu GNOME, KDE Plasma)

Example: In Windows, when you right-click a file and choose “Delete”, the OS executes system calls to remove the file from the file system.

3. Menu-Driven Interface: In a **Menu-Driven Interface**, users interact through a list of options or menus displayed on the screen. They can choose desired operations using keyboard shortcuts or arrow keys.

- **How it works:** The system presents a set of options. The user selects one, and the OS performs the related operation.
- **Advantages:**
 - Simple and easy to navigate
 - No need to remember commands
 - Suitable for devices with limited input (like ATMs or kiosks)

- **Disadvantages:**
 - Slower for frequent tasks
 - Limited flexibility
- **Examples:**
 - **ATM machine screens**
 - **Mobile phone settings menus**
 - **Old DOS-based software installers**

Example:

When using an ATM, you select options like “Withdraw Money” or “Check Balance” from menus — this is a menu-driven interface. Other Interfaces (Advanced Types)

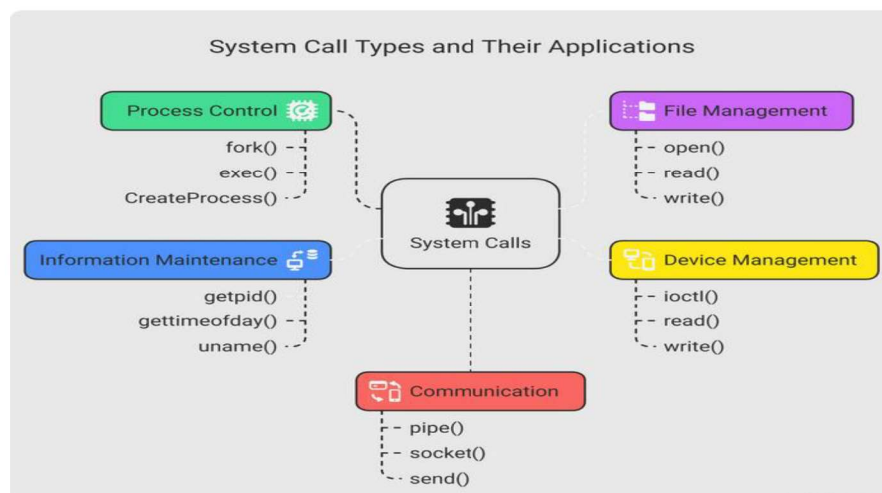
- **Touch User Interface (TUI):** Found in smart phones and tablets where users interact through touch gestures. (Example: Android, iOS)
- **Voice User Interface (VUI):** Allows voice commands. (Example: Siri, Alexa)
- **Application Programming Interface (API):** Interface for application developers to communicate with the OS using predefined functions or system calls. (Example: Windows API, POSIX API in Linux)

Q) System calls and types?

System calls are the interface between user programs and the operating system, allowing applications to request services like file access, process control, or device handling. They are the only way for a program to switch from user mode to kernel mode and safely use system resources managed by the kernel.

Types of System Calls

Services provided by an OS are typically related to any kind of operation that a user program can perform like creation, termination, forking, moving, communication, etc. Similar types of operations are grouped into one single system call category. System calls are classified into the following categories:



1. Process Control: Process control system calls are essential for managing processes — which are instances of running programs in an operating system. They enable creating new processes, managing their execution, and terminating them.

- **fork():** This system call creates a new process by duplicating the existing one (called the parent process). The new process is called the child process. Both processes run concurrently, allowing multitasking. This is fundamental in Unix-like systems for creating new processes.
- **exec():** This call replaces the current process image with a new program. After executing `exec()`, the new program starts running in the process's place. It's useful for running a different program within the same process without creating a new one.
- **Create Process():** Used primarily in Windows operating systems, this system call creates a new process and its primary thread. It also loads a specified executable into the process. It is a more comprehensive function than `fork()` and `exec()`, handling multiple aspects of process creation in one call.

2. File Management: File management system calls allow programs to interact with files stored on disk or other storage devices. These calls provide ways to create, access, read, write, and close files.

- **Open ():** This call opens a file, returning a file descriptor (or handle) that programs use to refer to the file in subsequent operations. It can create the file if it does not exist, depending on parameters.
- **Read ():** After opening a file, this call reads data from it into a buffer in memory. It transfers a specified number of bytes from the file to the program, enabling data retrieval.
- **Write ():** This call writes data from a buffer in memory to the file. It allows the program to save or modify data stored on disk, updating the file's contents.

3. Device Management: Device management system calls handle communication between the operating system and hardware devices such as printers, disk drives, keyboards, and network adapters. These calls provide abstraction so software doesn't have to manage hardware specifics.

- **ioctl() (Input/Output Control):** This versatile call sends device-specific commands to hardware devices. It is often used for operations like setting device parameters or controlling behavior that doesn't fit into standard read/write operations.
- **read():** Similar to file management, this call can also read data from devices (e.g., reading input from a keyboard or data from a network card).
- **write():** This call sends data to devices (e.g., sending data to a printer or output device). Both read and write here are generalized for devices, not just files.

4. Information Maintenance: These system calls allow the operating system and programs to retrieve or update system-related information, useful for managing processes and system status.

- **getpid():** Returns the Process ID (PID) of the calling process. This unique identifier is useful for process management, signaling, or debugging.
- **gettimeofday():** Retrieves the current system time and date, often used for time-stamping or measuring time intervals within applications.
- **uname():** Returns system information such as the OS name, version, and hardware details. This helps programs adapt to different environments or collect system diagnostics.

5. Communication: Communication system calls provide mechanisms for data exchange between processes, which may be on the same machine or across a network. They are key for implementing inter-process communication (IPC).

- **pipe():** Creates a unidirectional communication channel between two related processes. One process writes data to the pipe, and the other reads it, allowing simple data sharing.
- **socket():** Creates an endpoint for network communication. Sockets enable sending and receiving data over network protocols like TCP/IP, fundamental for client-server communication.
- **send():** Sends data through a socket to another process, either locally or over a network connection. It is used to transmit messages, files, or commands in distributed applications.

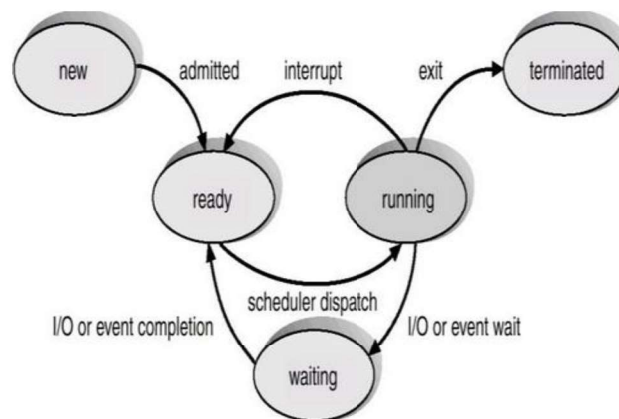
Q) Process in Operating System:

A process is a program in execution. The execution of a process progresses in a sequential fashion. A program is a passive entity while a process is an active entity. A process includes much more than just the program code. A process includes the text section, stack, data section, program counter, register contents and so on. The text section consists of the set of instructions to be executed for the process. The data section contains the values of initialized and uninitialized global variables in the program. The stack is used whenever there is a function call in the program. A layer is pushed into the stack when a function is called. The arguments to the function and the local variables used in the function are put into the layer of the stack. Once the function call returns to the calling program, the layer of the stack is popped. The text, data and stack sections comprise the address space of the process. The program counter has the address of the next instruction to be executed in the process.

It is possible to have two processes associated with the same program. For example, consider an editor program, say Microsoft Word. The program has the same text section. But, the data section will be different for each file that is opened in Microsoft Word, that is, each file has a different data section.

Process States: As a process executes, it changes state. The state of a process refers to what the process currently does. A process can be in one of the following states during its lifetime:

- **New:** The process is being created.
- **Running:** Instructions are being executed.
- **Waiting:** The process is waiting for some event to occur.
- **Ready:** The process is waiting to be assigned to a processor.
- **Terminated:** The process has finished execution.



1. New State: When a process is first created, it enters the new state. In this state, the operating system performs all the necessary setup tasks, such as creating the process control block (PCB), assigning memory, and allocating other required resources. The process is not yet ready to run at this point. Once the system admits it into the pool of executable processes, it moves to the ready state. This admission depends on system load and scheduling policies.

2. Ready State: In the ready state, the process has been fully prepared to run and is waiting for the CPU to be assigned to it. All the resources except the CPU are available to the process. It stays in the ready queue, waiting for its turn based on the CPU scheduling algorithm (like FCFS(First-Come, First-Served), Round Robin, etc.). When the scheduler selects the process, it is dispatched to the CPU and transitions into the running state.

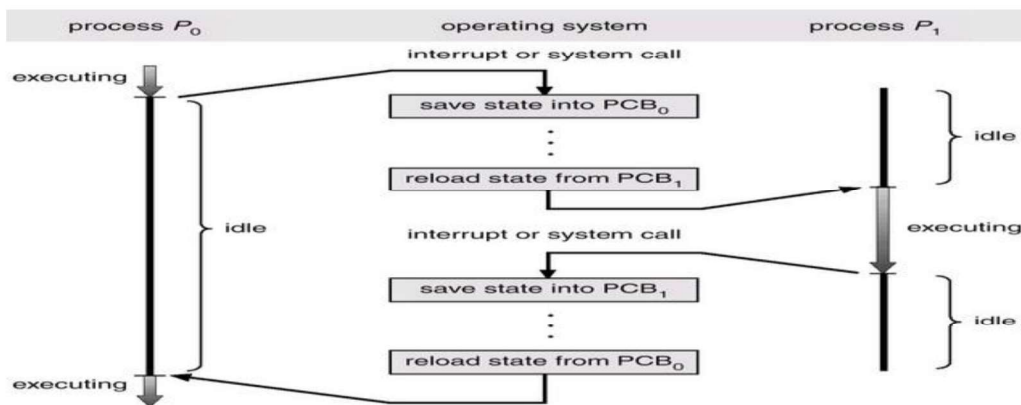
3. Running State: The running state is where the actual execution of the process takes place. The CPU executes the instructions of the process, performing computations, accessing memory, and interacting with I/O devices. A process remains in this state until it either completes execution, needs to wait for some I/O or event, or is preempted by the scheduler. If the process finishes its task, it moves to the terminated state; if it requests I/O or an event, it moves to the waiting state; and if interrupted, it returns to the ready state.

4. Waiting (or Blocked) State: When a process cannot continue execution until some external event occurs, such as the completion of an I/O operation or the availability of a resource, it enters the waiting state. In this state, the process is temporarily inactive and does not compete for the

CPU. Once the event or I/O operation it was waiting for is completed, the process is moved back to the ready state, where it again waits for CPU scheduling.

5. Terminated (or Exit) State: After a process finishes its execution or is forcefully terminated due to an error or external command, it enters the terminated state. In this final state, the operating system performs cleanup operations such as deallocating memory, closing files, and removing the process control block from the system. Once this cleanup is done, the process is completely removed from the system and no longer exists.

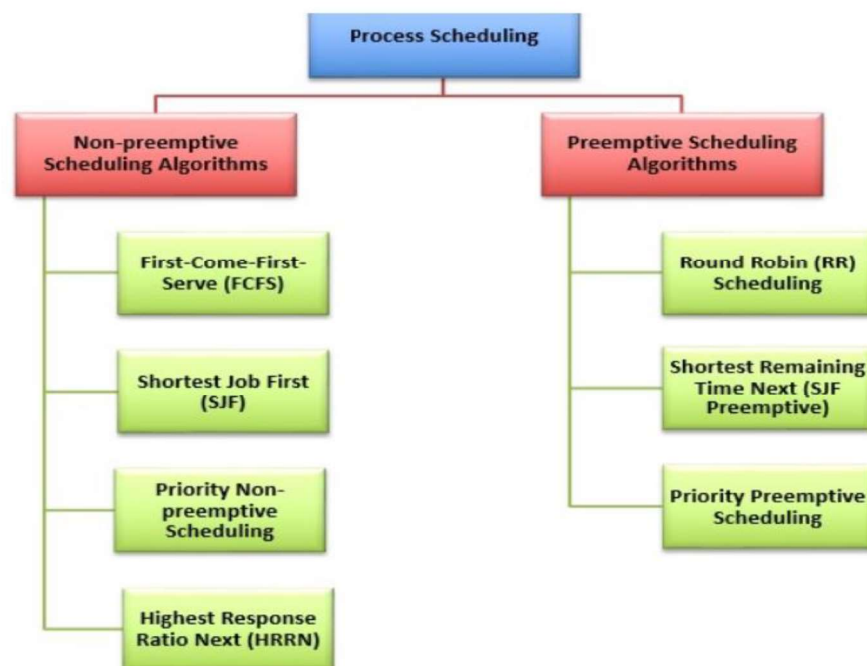
Process control Block:



1. **Definition:** The Process Control Block (PCB) is a data structure maintained by the operating system to store all information about a specific process.
2. **Purpose:** It acts as a repository for information needed by the OS to manage and control process execution efficiently.
3. **Creation:** A PCB is created when a process is admitted into the system (in the "new" state) and deleted when the process terminates.
4. **Unique Identification:** Each PCB has a unique **Process ID (PID)** which distinguishes it from other processes in the system.
5. **Process State Information:** The PCB stores the current state of the process — such as new, ready, running, waiting, or terminated.
6. **Program Counter:** It contains the **program counter (PC)**, which holds the address of the next instruction to be executed for that process.
7. **CPU Register Information:** The PCB saves the contents of all CPU registers (like stack pointer, general-purpose registers) during context switching.
8. **Memory Management Information:** It includes memory-related details like base and limit registers, page tables, and segment tables that define the process's memory area.
9. **Accounting Information:** The PCB holds process accounting data such as CPU usage, process priority, job number, and execution time for scheduling and monitoring purposes.
10. **I/O Status Information:** It records the list of I/O devices allocated to the process, open files, and pending I/O operations.

11. **Parent and Child Process Links:** The PCB may store references to the parent process and child processes to manage process hierarchies.
12. **Scheduling Information:** It contains details like process priority, scheduling queue pointers, and other scheduling parameters used by the CPU scheduler.
13. **Protection Information:** Some PCBs store access control information like permissions and privileges to maintain process and memory protection.
14. **Context Switching Role:** During context switching, the OS saves the current process's CPU state in its PCB and loads another process's state from its PCB — allowing multitasking.
15. **Storage Location:** All PCBs are typically maintained in a **process table** within the kernel, which the OS uses to keep track of all active processes.

Q) Process Scheduling:



- **Process Scheduling** is one of the most important functions of an operating system. It determines the order in which processes are executed by the CPU. Since many processes may be ready to execute at the same time, the operating system uses different scheduling algorithms to decide which one to run next. The main goal of process scheduling is to maximize CPU utilization, reduce waiting time, improve system responsiveness, and ensure fairness among processes. Process scheduling is generally divided into two main categories — **Non-preemptive Scheduling** and **Preemptive Scheduling**.

- **Non-preemptive Scheduling Algorithms** are those in which once a process starts execution, it cannot be interrupted until it finishes or voluntarily releases the CPU (for example, when waiting for I/O). The CPU is allocated to a process for its entire burst time, ensuring simple management but possibly causing inefficiencies.
- The **First Come First Serve (FCFS)** algorithm is the simplest form of scheduling. In this method, the process that arrives first in the ready queue is executed first. It works just like a queue in a ticket counter — first-come, first-served. Once the CPU is assigned to a process, it runs to completion. Although FCFS is easy to understand and implement, it may lead to a problem known as the “convoy effect,” where short processes get stuck waiting behind long ones, leading to poor average waiting time and response time.
- The **Shortest Job First (SJF)** algorithm selects the process with the smallest CPU burst time for execution next. This algorithm provides the minimum average waiting time and is optimal for batch systems. However, its major drawback is that the CPU burst time of each process must be known in advance, which is often not possible in real systems. Another problem with SJF is starvation, where long processes may never get executed if short ones keep arriving.
- The **Priority Non-preemptive Scheduling** algorithm assigns each process a priority value, and the CPU is allocated to the process with the highest priority. If two processes have the same priority, the one that arrived first is chosen (FCFS rule). This type of scheduling is useful when some processes are more important than others. However, lower-priority processes might suffer from starvation if high-priority processes keep arriving continuously.
- The **Highest Response Ratio Next (HRRN)** algorithm is an improved version of SJF that prevents starvation. In HRRN, the next process to be executed is chosen based on the highest response ratio, which is calculated as $(\text{Waiting Time} + \text{Burst Time}) / \text{Burst Time}$. The longer a process waits, the higher its response ratio becomes, ensuring that even long processes eventually get CPU time. This makes HRRN fairer than SJF while still providing good average waiting time.
- **Preemptive Scheduling Algorithms** allow the operating system to take the CPU away from a process before it completes execution. This means that if a higher-priority or shorter process arrives, or the time quantum expires, the currently running process can be paused and moved back to the ready queue. Preemptive scheduling provides better responsiveness and is ideal for time-sharing systems, but it increases the overhead due to frequent context switching.
- The **Round Robin (RR)** scheduling algorithm is one of the most widely used preemptive methods. Each process is assigned a fixed time slice or quantum to execute. When a process's time slice expires, it is preempted and placed at the end of the ready queue, allowing the next process to use the CPU. This method ensures fairness, as all processes get equal CPU time in a cyclic manner. Round Robin works best for time-sharing and interactive systems. However, the performance of this algorithm heavily depends on the choice of time quantum. If it's too short, it causes too many context switches; if it's too long, it behaves like FCFS.

- The **Shortest Remaining Time Next (SRTN)** scheduling algorithm is the preemptive version of SJF. In SRTN, the process with the shortest remaining CPU burst time is always selected for execution. If a new process arrives with a shorter remaining time than the currently running one, the CPU is preempted, and the new process starts execution immediately. This algorithm gives excellent average waiting and turnaround times but can cause starvation for long processes. It also requires accurate prediction of burst times, which is not always feasible.
- The **Priority Preemptive Scheduling** algorithm is similar to the non-preemptive version, except that the CPU can be taken away from a running process if a new process with a higher priority arrives. This ensures that important processes get CPU attention immediately. It is very effective in real-time systems where certain tasks must be executed quickly. However, just like the non-preemptive version, it may cause starvation for low-priority processes. Techniques like **aging** are often used to gradually increase the priority of waiting processes and prevent starvation.
- Finally, **non-preemptive scheduling algorithms** like FCFS, SJF, Priority, and HRRN are simple and predictable but may cause inefficiencies due to lack of preemption. On the other hand, **preemptive scheduling algorithms** like Round Robin, SRTN, and Priority Preemptive provide better responsiveness and fairness, especially in multi-user or real-time systems, but at the cost of higher overhead and complexity. The choice of algorithm depends on the specific goals of the system — such as response time, throughput, fairness, or efficiency.

Q) Inter process communication:

Inter process communication is the mechanism provided by the **operating system** that allows processes to communicate with each other. This communication could involve a process letting another process know that some event has occurred or the transferring of data from one process to another.



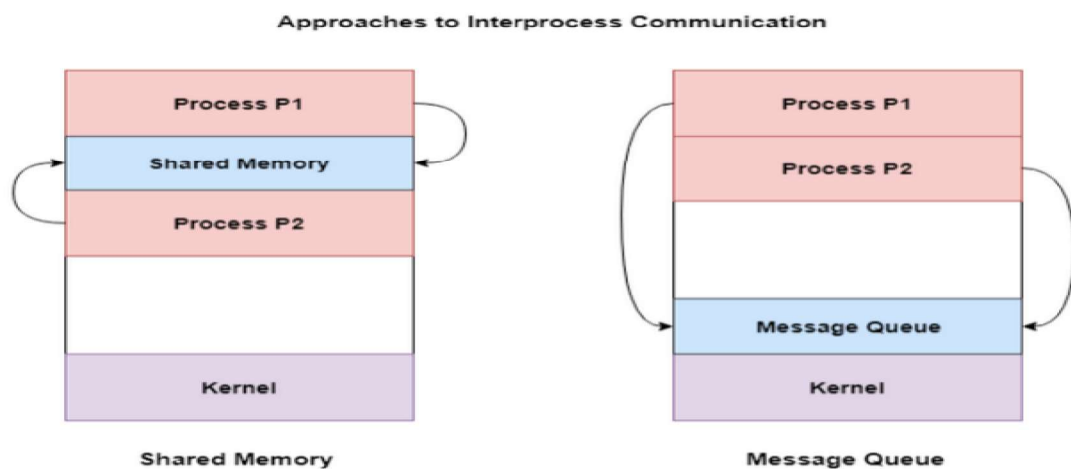
Synchronization in Inter process Communication:

Synchronization is a necessary part of inter process communication. It is either provided by the inter process control mechanism or handled by the communicating processes. Some of the methods to provide synchronization are as follows?

- **Semaphore:** A [semaphore](#) is a variable that controls the access to a common resource by multiple processes. The two types of semaphores are binary semaphores and counting semaphores.

- **Mutual Exclusion:** Mutual exclusion requires that only one process thread can enter the critical section at a time. This is useful for synchronization and also prevents race conditions.
- **Barrier:** A barrier does not allow individual processes to proceed until all the processes reach it. Many parallel languages and collective routines impose barriers.
- **Spinlock:** This is a type of lock. The processes trying to acquire this lock wait in a loop while checking if the lock is available or not. This is known as busy waiting because the process is not doing any useful operation even though it is active.

Approaches to Inter process Communication: The different approaches to implement inter process communication are given as follows?



- **Pipe :**A pipe is a data channel that is unidirectional. Two pipes can be used to create a two-way data channel between two processes. This uses standard input and output methods. Pipes are used in all POSIX systems as well as Windows operating systems.
- **Socket:** The socket is the endpoint for sending or receiving data in a network. This is true for data sent between processes on the same computer or data sent between different computers on the same network. Most of the operating systems use sockets for inter process communication.
- **File:** A file is a data record that may be stored on a disk or acquired on demand by a file server. Multiple processes can access a file as required. All operating systems use files for data storage.
- **Signal:** Signals are useful in inter process communication in a limited way. They are system messages that are sent from one process to another. Normally, signals are not used to transfer data but are used for remote commands between processes.
- **Shared Memory:** Shared memory is the memory that can be simultaneously accessed by multiple processes. This is done so that the processes can communicate with each other. All POSIX systems, as well as Windows operating systems use shared memory.

- **Message Queue:** Multiple processes can read and write data to the message queue without being connected to each other. Messages are stored in the queue until their recipient retrieves them. Message queues are quite useful for interprocess communication and are used by most operating systems.

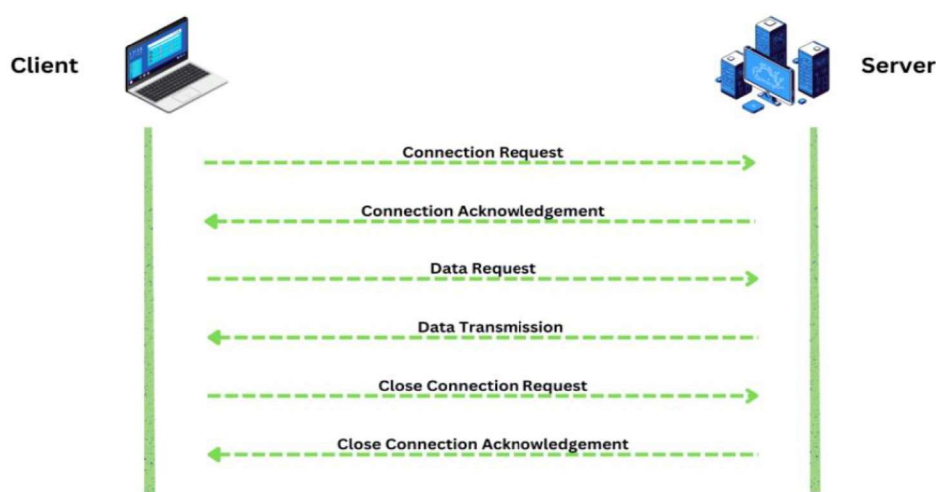
Q) Communication in Client server systems?

In an Operating System, Client Server Communication refers to the exchange of data and Services among multiple machines or processes. In Client server communication System one process or machine acts as a client requesting a service or data, and another machine or process acts like a server for providing those Services or Data to the client machine. This Communication model is widely used for exchanging data among various computing environments like **Distributed Systems, Internet Applications, and Networking Application communication.** The communication between Server and Client takes place with different Protocols and mechanisms.

In Client Server Communication we can use different ways.

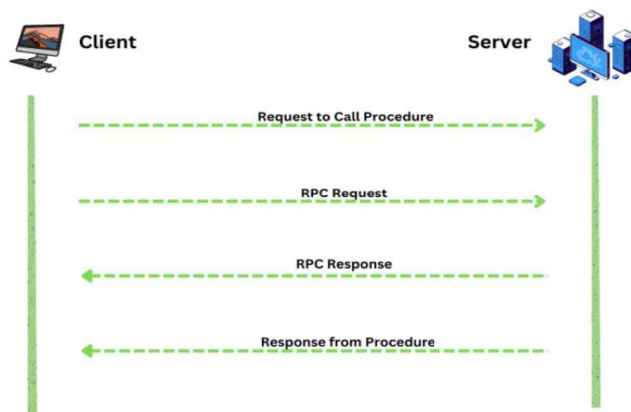
1. Sockets Mechanism
2. Remote Procedure Call
3. Message Passing
4. Inter-process Communication
5. Distributed File Systems

➤ **Sockets Mechanism:** The Sockets are the End Points of Communication between two machines. They provide a way for processes to communicate with each other, either on the same machine or over through Internet also possible. The Sockets enable the communication connection between Server and the client to transfer data in a bidirectional way.



Remote Procedure Call (PRC):

Remote Procedure Call is a Protocol. A Protocol is set of Instructions. It allows a client to execute a procedure call on remote server, as if it is local procedure call. PRC is commonly used in Client Server communication Architecture. PRC Provide high level of abstraction to the programmer. In This The client Program issues a procedure call , which is translated into message that is sent over the network to the Server, The Server execute the call and send back to the Client Machine.



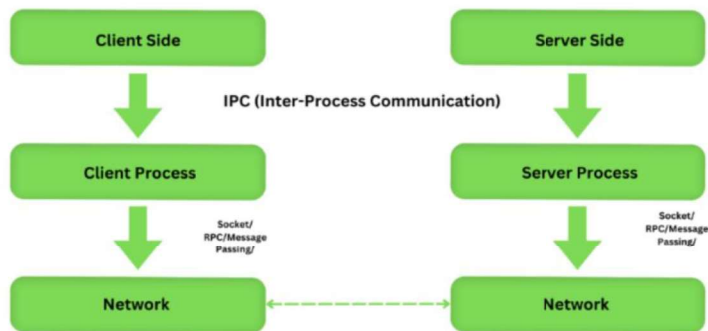
Message Passing

Message Passing is a communication Method. Where the machines communicated with each one by send and receiving the messages. This approach is commonly used in Parallel and Distributed Systems, This approach enables data exchange among the System.



Inter process Communication

The Inter Process Communication also called **IPC**. It allows communication between processes within the same Machine. The IPC can enable data sharing and Synchronous between different processes running concurrently on an operating system. And it includes Sharing Memory, message queues, semaphores and pipes among others.



Distributed File Systems

Distributed File Systems provide access to files from multiple machines in network. Client can access and manipulate files stored on Remote Server, Through Standard Interface Example Network File System and Server Message Block.



Q) Multithreaded Programming:?

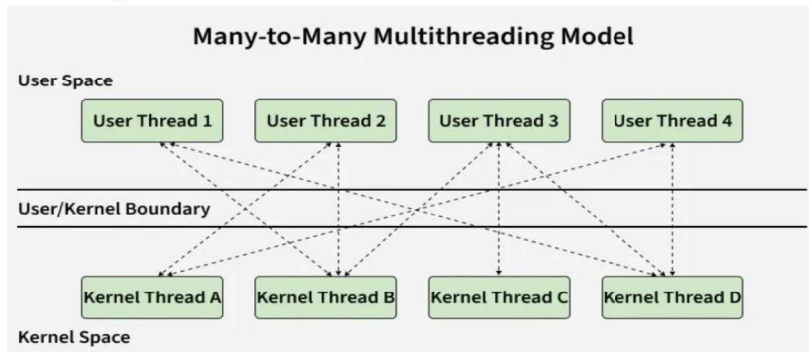
Multithreading is a technique where a process is divided into smaller execution units called threads that run concurrently.

- A thread is also called a lightweight process. Concurrency or Parallelism within a process is achieved by dividing a process into multiple threads.
- Multithreading improves system performance and responsiveness by allowing multiple threads to share CPU, memory and I/O resources of a single process.
- **Example:** In a browser, each tab can be a thread. In MS Word, one thread formats text while another processes inputs.

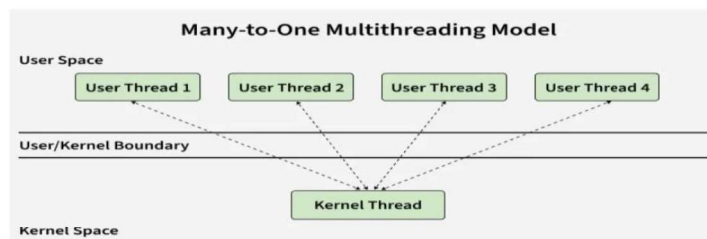
Multithreading model are of three types.

- **Many to many model.**
- **Many to one model.**
- **One to one model.**

Many to Many Model: In this model, we have multiple user threads multiplex to same or lesser number of kernel level threads. Number of kernel level threads are specific to the machine, advantage of this model is if a user thread is blocked we can schedule others user thread to other kernel thread. Thus, System doesn't block if a particular thread is blocked. It is the best multi threading model.

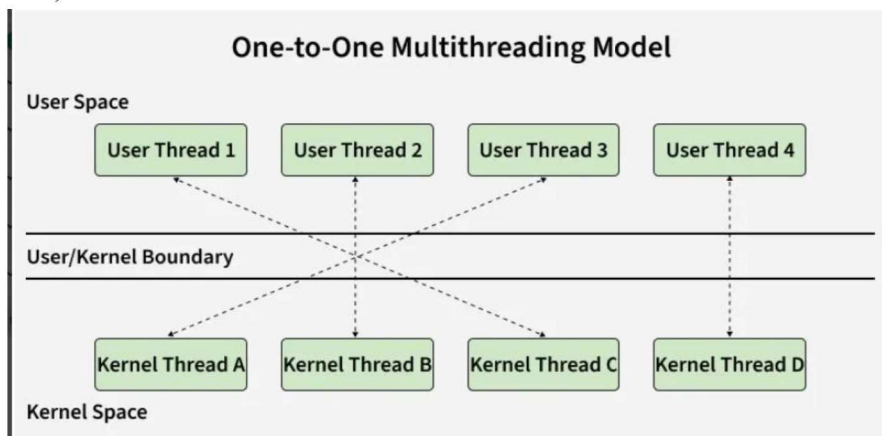


Many to One Model : In this model, we have multiple user threads mapped to one kernel thread. In this model when a user thread makes a blocking system call entire process blocks. As we have only one kernel thread and only one user thread can access kernel at a time, so multiple threads are not able access multiprocessor at the same time. The thread management is done on the user level so it is more efficient.



One to One Model : In this model, one to one relationship between kernel and user thread. In this model multiple thread can run on multiple processor. Problem with this model is that creating a user thread requires the corresponding kernel thread.

As each user thread is connected to different kernel , if any user thread makes a blocking system call, the other user threads won't be blocked.



Q) Threading Issues ?

Threading in operating systems brings **powerful concurrency**, but it also introduces **several issues and challenges** that must be carefully managed. These are often called **threading issues** or **multithreading problems**. Some of the threading issues are defined as follows.

1. Synchronization Issues

- When multiple threads access **shared resources** (like variables, files, or memory) simultaneously, inconsistent results can occur.
- Example: Two threads updating a shared counter at the same time may overwrite each other's updates. (The data may be override due to two threads update the data at a time).

Solution: Use synchronization mechanisms — mutexes, semaphores, monitors, condition variables.

2. Race Conditions

- Occur when the **output depends on the order or timing** of thread execution.
 - Example:
 - Thread 1: $x = x + 1$
 - Thread 2: $x = x + 1$
- If both read $x = 5$ at the same time, the result could incorrectly be $x = 6$ instead of $x = 7$.

Solution: Synchronize critical sections so only one thread modifies shared data at a time.

3. Deadlocks

- A situation where two or more threads are **waiting indefinitely** for resources locked by each other.

Example:

- Thread A holds Resource 1 and waits for Resource 2.
- Thread B holds Resource 2 and waits for Resource 1.

Solution: Avoid circular waits, acquire locks in a consistent order, use timeout mechanisms.

4. Starvation (Livelock)

- Some threads **never get CPU time or resources** because others always take priority.
- Can happen if a scheduling policy always favors certain threads.

Solution: Fair scheduling algorithms (like round-robin or aging).

5. Thread Interference

- Occurs when threads interleave their operations in unpredictable ways, leading to incorrect results.
- Similar to race conditions but focuses more on the **interleaving of operations**.

6. Context Switching Overhead

- Frequent switching between threads causes CPU overhead. (**Thread states are continuously changing**)
- Each context switch requires saving and restoring thread states, affecting performance.

7. Portability Issues

- Thread behavior can vary across operating systems and compilers. (**Thread can change based on the operating system**)
- Example: Thread libraries like **Pthreads** vs **Windows threads** have different APIs and synchronization primitives.

8. Memory Consistency Errors

- Threads might see **stale or inconsistent values** of shared variables due to caching and compiler optimization. (**We can see variation of thread like before and after**)
- Happens when changes by one thread are not immediately visible to others.

Solution: Use **memory barriers** or **volatile variables** to ensure visibility of shared data.

9. Debugging Difficulty

- Multithreaded programs are **hard to test and debug** because bugs like race conditions are timing-dependent and nondeterministic. (**That means the errors are continuously changing, we can't able to provide solutions for such type of errors.**)

UNIT-2

Introduction about CPU Scheduling:

CPU scheduling in an Operating System (OS) is the process of deciding which process or thread should use the CPU (Central Processing Unit) at any given time.

Since a CPU can usually execute only one process at a time (per core), and there are often many processes waiting, the OS uses a CPU scheduler to efficiently manage how CPU time is distributed among them.

Purpose of CPU Scheduling: The main goals are:

- **Maximize CPU utilization** (keep the CPU busy as much as possible)
- **Improve system throughput** (To increase the number of processes per unit time)
- **Minimize waiting time** (time spent in the ready queue)
- **Minimize turnaround time** (total time to execute a process)
- **Minimize response time** (time until a process first gets CPU)
- **Ensure fairness** (every process gets a chance to run)

Types of Scheduling:

1. Preemptive Scheduling: Preemptive scheduling is a **CPU scheduling method** in which the **operating system can interrupt (preempt)** a currently running process and **assign the CPU to another process**.

This usually happens when:

- A **higher-priority process** arrives, or
- The **running process's time slice (quantum)** expires.

2. Non-preemptive Scheduling: Non-preemptive scheduling is a **CPU scheduling method** in which **once a process gets the CPU, it cannot be interrupted** until it **finishes execution** or **voluntarily releases the CPU** (e.g., for I/O operations).

In other words, the CPU is **occupied by the process until it completes**, and no other process can take it away.

Q) Scheduling criteria?

CPU scheduling criteria are the **standards or parameters** used to **evaluate and compare** the performance of various CPU scheduling algorithms in an operating system. In formal terms, these criteria serve as quantitative measures to determine the **efficiency, effectiveness, and fairness** of a scheduling method in managing CPU resources and process execution.

There are many different criteria to check when considering the **"best"** scheduling algorithm, they are

1. CPU Utilization: To make out the best use of the CPU and not to waste any CPU cycle, the CPU would be working most of the time (Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

Example – CPU Utilization: Suppose a computer system is running several applications like a web browser, a word processor, and a media player.

- If the CPU is **busy 85% of the time** and idle only 15% of the time, it indicates **high CPU utilization**, meaning the CPU is being effectively used.
- On the other hand, if the CPU is **busy only 30% of the time** and idle 70% of the time, the utilization is **low**, meaning the CPU is underused and many CPU cycles are wasted.

2. Throughput: It is the total number of processes completed per unit of time or rather says the total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

Example – Throughput:

Suppose a printing server handles multiple print jobs:

- In **Scenario 1**, the server completes **10 small print jobs every second**. Here, the throughput = **10 jobs/second**.
- In **Scenario 2**, the server processes **1 large, complex report per hour**. Here, the throughput = **1 job/hour**.

Throughput measures **how much work the system completes in a given time**. Higher throughput indicates the system is handling tasks efficiently, while lower throughput shows slower processing or heavier tasks.

3. Turnaround Time: It is the amount of time taken to execute a particular process, i.e. The interval from the time of submission of the process to the time of completion of the process (Wall clock time).

Example: Suppose three processes are submitted to a CPU scheduler:

Process	Arrival Time	Burst Time
P1	0 ms	5 ms
P2	1 ms	3 ms
P3	2 ms	8 ms

Let's assume **First-Come, First-Served (FCFS)** scheduling.

Execution Order: P1 → P2 → P3

Completion Times:

- P1 finishes at 5 ms
- P2 finishes at 8 ms (started at 5 ms, burst 3 ms)
- P3 finishes at 16 ms (started at 8 ms, burst 8 ms)

- **Turnaround Time Calculation:**

P1: $5 - 0 = 5 \text{ ms}$

- P2: $8 - 1 = 7 \text{ ms}$

- P3: $16 - 2 = 14 \text{ ms}$

Average Turnaround Time:

$5 + 7 + 14 / 3 = 8.67$.

4. Waiting Time: The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

Example: The total time a process spends waiting in the ready queue before it gets the CPU.

- Formula: $\text{Waiting Time} = \text{Turnaround Time} - \text{Burst Time}$

5. Load Average: It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

Example :Load Average represents the average number of processes waiting in the ready queue over a period of time.

- It measures how “busy” the CPU is and indicates the system load.
- Formula (simplified for a period):

$$\text{Load Average} = \frac{\text{Total time all processes spend in ready queue}}{\text{Total observation time}}$$

6. Response Time: Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution (final response).

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

- Formula (simple):

$$\text{Response Time} = \text{Time when process first gets CPU} - \text{Arrival Time}$$

Q) Scheduling Algorithms in Operating System

To decide which process to execute first and which process to execute last to achieve maximum CPU utilization, computer scientists have defined some algorithms, they are:

These algorithms are either non-preemptive or preemptive. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

1Jobs are executed on first come, first serve basis.

- It is a non-preemptivescheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Example: Let's say we have:

Process	Arrival Time	Burst Time
P1	0	5
P2	2	3

Even though P2 arrives while P1 is running, the CPU will **not stop P1** to run P2.

P2 must wait until P1 finishes.

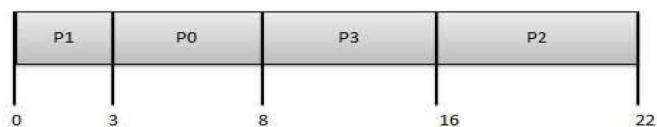


2.Shortest-Job-First (SJF) Scheduling: Shortest Job First (SJF) is a CPU scheduling algorithm that selects the process with the smallest CPU burst time (the shortest job) to execute next. The process that needs the least time to complete runs first.

- This is also known as shortest job first, or SJF
- This is a non-preemptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processor should know in advance how much time process will take.

Given: Table of processes, and their Arrival time, Execution time

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8



Waiting time of each process is as follows –

Process	Waiting Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	14 - 2 = 12
P3	8 - 3 = 5

Average Wait Time: $(0 + 4 + 12 + 5)/4 = 21 / 4 = 5.25$

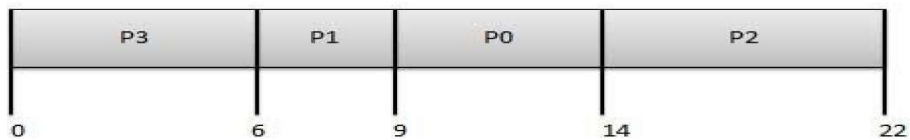
3.Priority Scheduling:- Priority Scheduling is a CPU scheduling algorithm in which each process is assigned a priority value, and the CPU is allocated to the process with the highest priority. If two processes have the same priority, the OS usually uses FCFS (First Come First Serve) to break the tie.

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.

- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Given: Table of processes, and their Arrival time, Execution time, and priority. Here we are considering 1 is the lowest priority.

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



Waiting time of each process is as follows –

Process	Waiting Time
P0	$0 - 0 = 0$
P1	$11 - 1 = 10$
P2	$14 - 2 = 12$
P3	$5 - 3 = 2$

Average Wait Time: $(0 + 10 + 12 + 2)/4 = 24 / 4 = 6$

4. Round Robin (RR) Scheduling: Round Robin Scheduling is a method used by operating systems to manage the execution time of multiple processes that are competing for CPU attention. It is called "round robin" because the system rotates through all the processes, allocating each of them a fixed time slice or "quantum", regardless of their priority.

The primary goal of this scheduling method is to ensure that all processes are given an equal opportunity to execute, promoting fairness among tasks.

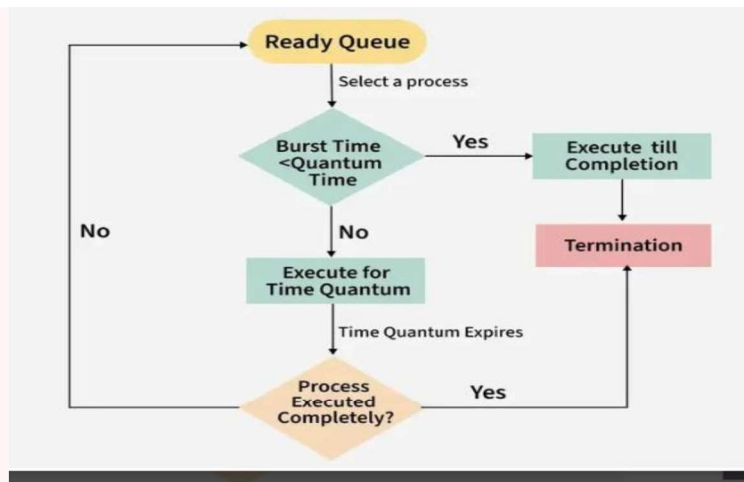
Process Arrival: Processes enter the system and are placed in a queue.

Time Allocation: Each process is given a certain amount of CPU time, called a quantum.

Execution: The process uses the CPU for the allocated time.

Rotation: If the process completes within the time, it leaves the system. If not, it goes back to the end of the queue.

Repeat: The CPU continues to cycle through the queue until all processes are completed.



Consider the set of 5 processes whose arrival time and burst time are given below-

Scenario 1: Processes with Same Arrival Time: Consider the following table of arrival time and burst time for three processes P1, P2 and P3 and given **Time Quantum = 2 ms**

Process	Burst Time	Arrival Time
P1	4 ms	0 ms
P2	5 ms	0 ms
P3	3 ms	0 ms

Step-by-Step Execution:

- Time 0-2 (P1):** P1 runs for 2 ms (total time left: 2 ms).
- Time 2-4 (P2):** P2 runs for 2 ms (total time left: 3 ms).
- Time 4-6 (P3):** P3 runs for 2 ms (total time left: 1 ms).
- Time 6-8 (P1):** P1 finishes its last 2 ms.
- Time 8-10 (P2):** P2 runs for another 2 ms (total time left: 1 ms).
- Time 10-11 (P3):** P3 finishes its last 1 ms.
- Time 11-12 (P2):** P2 finishes its last 1 ms.

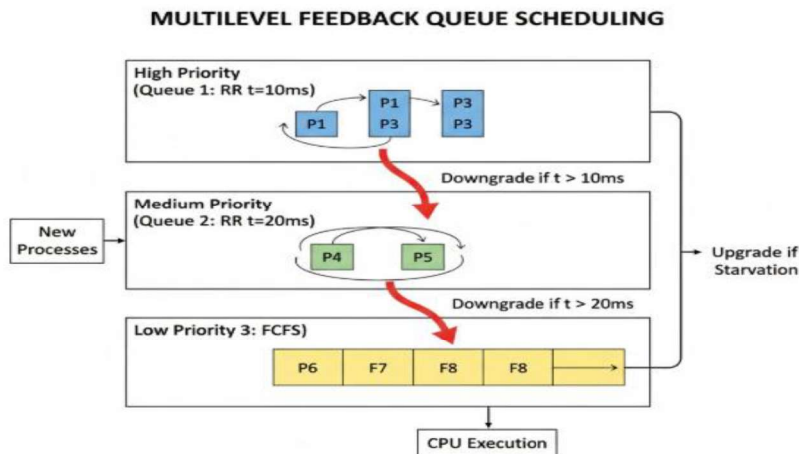
5. Multilevel Queue Scheduling: Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.

- Multiple queues are maintained for processes with common characteristics.
- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue.

For example, CPU-bound jobs can be scheduled in one queue and all I/O-bound jobs in another queue. The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.

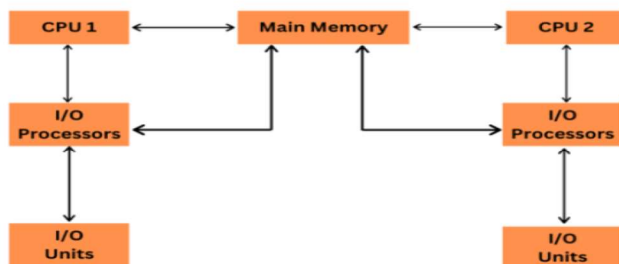
6. Multilevel Feedback Queue Scheduling: It is an advanced CPU scheduling algorithm that uses multiple queues with different priorities — and allows processes to move between these queues based on their behavior and requirements.

- The system has **multiple ready queues**, each with its **own priority level**.
- Each queue can have its **own scheduling algorithm** (e.g., Round Robin in one queue, FCFS in another).
- **High-priority queues** are served **before** lower-priority ones.
- If a process **uses too much CPU time**, it's **moved to a lower-priority queue**.
- If a process **waits too long**, it can be **moved up** (to avoid starvation).



Q) Multiple Processors Scheduling in Operating System:

Multiple processor scheduling involves deciding which processes should be assigned to which processor or core and how long they should be permitted to run. While ensuring that all processes are fairly and appropriately prioritized, the objective is to achieve efficient utilization of the available processors.



- Multiple processor scheduling (multiprocessor scheduling) aims to create a scheduling system that **uses several processors at the same time**.
- In this system, **multiple CPUs share the workload**, allowing many processes to run **concurrently**.
- It is **more complex** than single-processor scheduling because many processors must be managed together.
- Since the processors are **identical**, any process can run on **any processor** at any time.

- All CPUs in the system **communicate with each other** and share **memory, bus, and peripherals**, so the system is **strongly connected**.
- These systems are used when **large amounts of data** need to be processed, such as in **satellites and weather forecasting**.
- In many multiprocessor systems, processors work in a **homogeneous** (identical) way, so any waiting process can run on any free processor.
- Multiprocessor systems can be:
 - **Homogeneous** – all processors are the same
 - **Heterogeneous** – processors are different types
 - In this case, some special rules may apply, such as certain devices connected to only one CPU.
- There is **no perfect or universal scheduling method** for systems with one processor.
- Similarly, **there is no ideal scheduling strategy** for systems with multiple processors.

Approaches to Multiple Processor Scheduling: There are **two** different architectures utilized in multiprocessor systems:

Symmetric Multiprocessing:

- In an **SMP (Symmetric Multiprocessing)** system, all processors are **equal** and have the **same access** to memory and I/O devices.
- There is **no master–slave setup**; all CPUs work at the **same level**.
- Every processor can access **any memory location** and **any I/O device** without limitations.
- The **operating system** handles how tasks are shared among the processors.
- Because all processors are identical, **any processor can perform any task**.

Asymmetric Multiprocessing:

- In an **AMP (Asymmetric Multiprocessing)** system, one processor is the **master processor**.
- The master processor has **full access** to all system resources like memory and I/O devices.
- The master processor is responsible for **assigning tasks** to the other processors, called **slave processors**.
- Each slave processor performs only the tasks given by the master processor.
- The operating system gives tasks to the **master processor**, and the master then passes the work to the slave processors.

Types of Multiprocessor Scheduling Algorithms: Operating systems utilize a range of multiprocessor scheduling algorithms.

1. Round-Robin Scheduling: In round-robin scheduling, each CPU is given a fixed time slice, called a time quantum. Processes are then run one after another in a circular order on each processor. This method makes sure every process receives an equal amount of CPU time. Because of this fairness, round-robin scheduling is especially useful in systems where all processes have the same priority.

2. Priority Scheduling: In this method, each process is given a priority level, and processes with higher priority are run first. This approach is useful in systems where certain tasks, such as real-time jobs, need to be handled more urgently than others.

3. Scheduling with the shortest job first (SJF): This algorithm schedules processes based on how long they are expected to take. The shortest job is run first, followed by the next shortest, and so on. It is useful in systems with many quick tasks because it helps reduce the overall response time.

4. Fair-share scheduling: In this technique, the amount of time given to each process depends on both the number of processors and the priority of the process. It ensures that every process gets a fair share of CPU time, making it useful for systems that have a mix of long and short tasks.

5. Earliest deadline first (EDF) scheduling: In this algorithm, every process has a deadline, and the process with the **earliest deadline runs first**. This method is useful in systems with real-time tasks that must be completed on time.

6. Scheduling using a multilevel feedback queue (MLFQ): In a **multilayer feedback queue (MLFQ)**, processes are assigned different priority levels. A process can move **up or down** the priority levels depending on how it behaves. This method is useful in systems that have a mix of short and long tasks.

Use Cases of Multiple Processors Scheduling in Operating System

- **High-Performance Computing:** Multiple processor scheduling is very important in **high-performance computing (HPC)**, where big scientific simulations, data analysis, or complex calculations are done. By efficiently distributing tasks across multiple processors, processes can run in **parallel**, which makes computations faster and improves overall system performance.
- **Server Virtualization:** In virtualized environments, multiple virtual machines (VMs) run on a single server with many processors. Effective scheduling ensures that each VM gets a fair share of CPU time, uses the processing power efficiently, and maintains good performance for all VMs.
- **Real-Time Systems:** Real-time systems, like those in aerospace, defense, and industrial automation, must meet strict timing requirements. Multiple processor scheduling algorithms, such as Earliest Deadline First (EDF), make sure that important tasks with tight deadlines run on time, ensuring a prompt and reliable response.
- **Multimedia Processing:** Multimedia applications, like video rendering or audio processing, need a lot of computing power. Scheduling tasks across multiple processors lets them run in parallel, which makes processing faster and ensures smooth real-time performance.
- **Distributed Computing:** In distributed computing systems, tasks are shared across multiple processors or nodes to work together. Good scheduling ensures balanced workloads, reliability, and efficient use of resources, which improves the system's overall performance and scalability.
- **Cloud Computing:** Cloud service providers use multiple processors to handle many client requests at the same time. Scheduling algorithms help assign virtual machines and containers efficiently, ensuring fair use of resources, scalability, and smooth performance in the cloud.
- **Big Data Processing:** Big data analytics deals with huge amounts of data. Multiple processor scheduling allows tasks like data collection, transformation, and analysis to run in parallel, which speeds up processing and helps provide real-time or near-real-time insights.
- **Scientific Simulations and Modeling:** Numerical simulations and scientific modeling need a lot of computing power. Multiple processor scheduling lets simulation tasks run **in parallel**, which **speeds up results** and helps researchers study complex phenomena more accurately.

- **Gaming:** Modern gaming systems use multiple processors to manage graphics, physics, and AI tasks. Effective scheduling ensures smooth gameplay, reduces lag, and makes the best use of the processors for an immersive gaming experience.
- **Embedded Systems:** Embedded systems with multiple processors, like those in **cars, IoT devices, or robots**, need efficient scheduling to ensure **timely responses and real-time control**. Scheduling algorithms **prioritize important tasks** and manage resources to meet system needs.

Q) Process synchronization

Process synchronization is the coordination of concurrent processes in an operating system to ensure they access shared resources without causing data inconsistency or conflicts. It is crucial in multiprocess systems to prevent issues like **race conditions** and ensure data integrity and correct program execution. Synchronization is achieved through mechanisms such as **semaphores, monitors, mutex locks, critical sections** and so on.

Why process synchronization is needed:

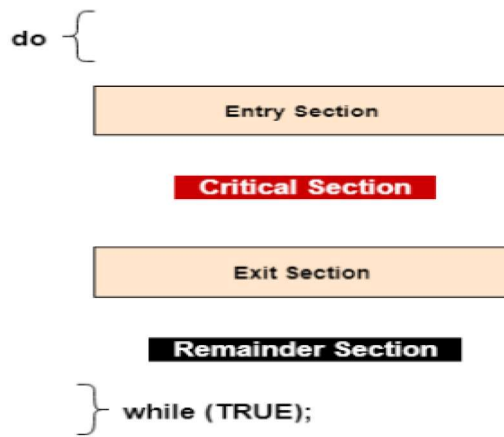
- **To prevent race conditions:** When multiple processes access shared data simultaneously, their operations can be interleaved in an unpredictable order, leading to incorrect results.
- **To maintain data consistency:** Synchronization ensures that shared data remains reliable and consistent, even when multiple processes are modifying it.
- **To avoid data corruption:** Without synchronization, a process's update could be overwritten by another before it's complete, leading to data corruption.
- **To manage shared resources:** It provides a way to control access to shared resources like memory or hardware, ensuring they are used in an orderly manner.

Synchronization mechanisms

1. The critical-section problem
2. Synchronization hardware
3. Semaphores
4. Classic problems of Synchronization
5. Monitors

1. The critical-section problem: The critical section is a code segment where the shared variables can be accessed. An atomic action is required in a critical section i.e. only one process can execute in its critical section at a time. All the other processes have to wait to execute in their critical sections.

A diagram that demonstrates the critical section is as follows –



1. Entry Section:

- The Entry Section is the part of the code where a process tries to gain permission to enter its critical section.
- Since the critical section involves accessing shared resources, it is essential to ensure mutual exclusion—only one process can be inside the critical section at a time.
- In the entry section, the process executes synchronization mechanisms (such as locks, semaphores, or algorithms) to check if it is safe to enter the critical section.
- If another process is already inside the critical section, the current process waits or retries until the critical section becomes free.
- The entry section acts as a gatekeeper to prevent concurrent access and ensure the system's correctness.

2. Critical Section:

- The Critical Section is the segment of the program where the process actually accesses and manipulates shared resources.
- During this time, the process performs operations such as reading or writing shared variables, updating data structures, or interacting with hardware devices that cannot be safely used by multiple processes simultaneously.
- It is crucial that only one process executes in the critical section at any moment to avoid data corruption or inconsistent results.
- Proper synchronization ensures that once a process enters its critical section, it has exclusive access until it finishes its operations and leaves.

3. Exit Section:

- The Exit Section is the code segment executed immediately after the process finishes its critical section.
- The purpose of this section is to release any locks, signals, or flags set during the entry and critical sections.
- By doing so, it informs the system that the process has completed its critical operation and that other waiting processes can now enter their critical sections.

- This section ensures that the mutual exclusion condition is properly relaxed so that the next process in line can proceed safely.

4. Remainder Section:

- The Remainder Section is the portion of the process outside the critical section where it executes code unrelated to accessing shared resources.
- This may include performing local computations, input/output operations that do not require synchronization, or any other independent work.
- Since the remainder section does not involve shared data, multiple processes can execute their remainder sections concurrently without any synchronization constraints.
- Once the process completes its remainder section, it typically tries again to enter the critical section, repeating the cycle.

This loop (`do { ... } while(TRUE);`) represents the continuous execution of these four parts, allowing processes to repeatedly perform critical section operations safely while maintaining proper synchronization.

Example: Multiple employees want to print documents using a single shared printer. The printer is the shared resource.

Problem Without Critical Section:

- If two employees send print jobs at the same time, their pages may **mix together**, causing confusion.

Critical Section Solution:

- Access to the printer is treated as a **critical section**.
- **Only one employee** can print at a time; others **wait in a queue**.

Step-by-Step:

1. Employee A enters the critical section → sends document to printer → printing starts.
2. Employee B waits until Employee A finishes → enters critical section → prints document.
3. Employee C waits their turn, and so on.

Result: Documents are printed **in order**, without mixing, and the printer resource is safely used.

2. Synchronization hardware:

Synchronization hardware uses atomic CPU instructions to manage concurrent processes, preventing race conditions and ensuring data consistency when multiple processes access shared resources. Key hardware algorithms include [Test and Set](#) and [Swap](#), which use instructions to automatically check and modify a lock variable, ensuring only one process can enter a critical section at a time.

Why Hardware Synchronization?

- Software solutions might be inefficient or complex, especially when multiple CPUs or cores are involved.

- Hardware solutions can guarantee **atomicity** and **speed**, preventing race conditions right at the instruction level.

Common Hardware Instructions for Synchronization

1. Test-and-Set Instruction (TAS)

- This is a simple atomic instruction that tests the value of a memory location and sets it to 1 **in one atomic step**.
- It returns the old value before setting.
- If the old value was 0, the process can enter the critical section; otherwise, it must wait.

How it works?

```
bool TestAndSet(bool *lock) {
    bool old_value = *lock;
    *lock = true;
    return old_value;
}
```

- A process uses this to implement a **spinlock**: it repeatedly calls **Test and Set** until it acquires the lock (i.e., the old value was 0).

2. Swap Instruction

- Atomically swaps the values of a register and a memory location.
- This can be used to exchange a lock variable and a local variable, allowing processes to test and acquire locks without interference.

3. Fetch-and-Add

- Atomically increments (or decrements) a variable and returns its old value.
- Useful for implementing counters or ticket locks in synchronization.

Example: Implementing Mutual Exclusion Using Test-and-Set

Suppose multiple processes want to access a shared printer (critical section). We can use a Boolean variable lock initialized to false to control access.

Pseudo code:

```
while (TestAndSet(&lock))
{
    // busy wait (spin) until lock is free (lock == false)
}
// Critical Section - access shared printer
print_document();
// Exit Section
lock = false; // release lock
```

- Here, TestAndSet(&lock) atomically tests if the lock is free and sets it to true if it is.
- If another process holds the lock (lock == true), the process keeps spinning (waiting).
- Once the lock is released, one waiting process acquires it and enters the critical section.

Advantages of Synchronization Hardware

- **Atomicity:** Guarantees that operations like test-and-set are indivisible.
- **Efficiency:** Faster than software-only solutions due to direct CPU support.

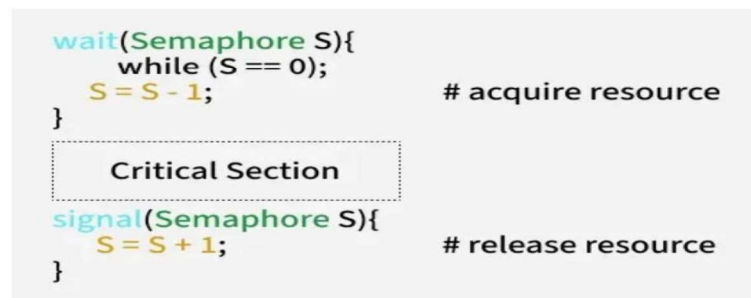
- **Simplicity:** Easier to implement basic synchronization primitives.

Limitations

- **Busy Waiting:** Simple hardware locks like spinlocks cause busy waiting, wasting CPU cycles.
- **Scalability:** Spinlocks are inefficient for high contention or many CPUs.

3.Semaphore:

A **semaphore** is a variable (an integer) that helps control access to shared resources in a multiprogramming system. When many processes want to use the same resource—like files, printers, or memory—the operating system uses semaphores to keep everything organized. A semaphore makes sure that only a certain number of processes are allowed to use a resource at the same time. It works using two basic operations. 1. Wait 2. Signal.



1. wait(Semaphore S) function

```

wait(Semaphore S) {
    while (S == 0);
    S = S - 1;
}

```

- **Purpose:** To **acquire** (lock or take) a resource before entering the critical section (the part of the program that uses the shared resource).

How it works:

- `while (S == 0);` means the process will **wait (busy wait)** as long as S is 0.
- `S == 0` means **no resources are available** right now.
- Once S becomes greater than 0, the process proceeds and **decrements** S by 1: `S = S - 1;`
- This means the process has successfully acquired one unit of the resource.

2. Critical Section

- This is the code block where the shared resource is accessed or modified.
- Only one or limited processes are allowed here at a time (depending on the semaphore count).

3. signal(Semaphore S) function

```

signal(Semaphore S) {
    S = S + 1;
}

```

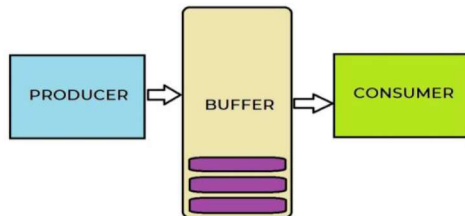
- Purpose: To release (unlock or free) the resource after finishing the critical section.
- **How it works:**
 - It simply increments the semaphore S by 1, meaning one more resource is now available.

Operation	Statement	Meaning
wait()	while (S == 0);	Wait until resource is available
wait()	S = S - 1;	Lock/acquire the resource
critical section		Code requiring mutual exclusion
signal()	S = S + 1;	Release resource and wake next process

4. **Classic problems of Synchronization:** Synchronization problems arise because multiple processes/threads try to access shared resources at the same time. If not controlled, it leads to **race conditions, deadlocks, or inconsistent states**. There are **four** classical problems:

1. Producer–Consumer Problem
2. Readers–Writers Problem
3. Dining Philosophers Problem
4. Sleeping Barber Problem

1. Producer–Consumer Problem



The producer-consumer problem is a situation where a producer creates items and puts them into a limited-size buffer, and a consumer takes items out of it. If the buffer is full, the producer must wait. If the buffer is empty, the consumer must wait. To avoid mistakes or data problems, we use synchronization tools like locks or semaphores so both do not use the buffer at the same time.

How it works with synchronization

Initial state: The buffer is initially empty.

Producer actions:

- Checks if the buffer is full. If it is, it waits.
- If there is space, it produces an item and adds it to the buffer.
- It then signals the consumer that an item is now available.

Consumer actions:

- Checks if the buffer is empty. If it is, it waits.
- If there is an item, it removes the item from the buffer.
- It then signals the producer that a slot is now free.

Synchronization solutions

- **Semaphores:** A common solution involves using three semaphores: one to count empty slots, one for full slots, and one for mutual exclusion to prevent both processes from accessing the buffer at the same time.
- **Locks/Mutexes:** A lock can be used to ensure that only one process can access the shared buffer at any given moment, preventing simultaneous modification.
- **Condition variables:** These are often used with locks to allow processes to wait for a specific condition (e.g., buffer not full or not empty) to become true.

2. Readers writers' problem:

- The reader–writer problem is a classic synchronization issue in operating systems involving shared resources.
- It deals with controlling access for two types of processes:
 - **Readers** → only read the data.
 - **Writers** → read and modify the data.
- Many readers can read the shared resource at the same time without any problem.
- However, **only one writer** can access the resource at a time.
- When a writer is writing, **no reader is allowed** to read the data.
- Synchronization is needed to:
 - Prevent readers from getting incorrect or corrupt data, and
 - Allow multiple readers to access the data when no writer is active.
- The main goal is to maintain data consistency while allowing maximum parallel reading.

Dining Philosophers Problem

- Imagine **five philosophers** sitting around a circular table.
- Each philosopher does **two activities**:
 1. **Thinking**
 2. **Eating**
- There are **five forks** on the table, one between each pair of philosophers.
- To **eat**, a philosopher needs **two forks** (one from the left and one from the right).

The Problem

- If all philosophers pick up their **left fork first** and then wait for the **right fork**, no one can eat.
- This situation leads to a **deadlock**, where all philosophers are waiting indefinitely.

Issues in the Problem

1. **Deadlock** → All philosophers are stuck waiting for a fork.
2. **Starvation** → Some philosophers may never get a chance to eat if others keep taking forks.
3. **Resource allocation conflicts** → Multiple philosophers compete for the same forks simultaneously.

Solutions to Dining Philosophers Problem:

1. Using a Semaphore (Mutual Exclusion)

- Introduce a **mutex** to control access to forks.
- Only allow a philosopher to pick up forks if **both forks are available**.
- This prevents deadlock because no one can hold just one fork and wait indefinitely.

2. Numbering Philosophers and Picking Forks in Order

- Number the philosophers 1 to 5.
- **Odd-numbered philosophers** pick up **left fork first**, then **right fork**.
- **Even-numbered philosophers** pick up **right fork first**, then **left fork**.
- This breaks the circular wait condition and **prevents deadlock**.

4. Sleeping Barber Problem

- The problem shows how to **synchronize processes** in an operating system.
- Involves:
 - **Barber** → gives haircuts and **sleeps if there are no customers**.
 - **Customers** → come for a haircut.
 - **Waiting room** → has a **limited number of chairs**.

The Problem

- If the barber has **no customers**, he should sleep.
- If a customer arrives and a **chair is free**, they wait for their turn.
- If **all chairs are full**, the customer leaves.
- The system should avoid:
 - Deadlock (everyone waiting forever)
 - Busy-waiting (wasting CPU cycles)

Solution Approach

- Use **semaphores** to manage access to:
 - **Barber chair** → only one customer can sit at a time.
 - **Waiting chairs** → track available seats.
- Semaphores ensure:
 - The barber **wakes up** when a customer arrives.
 - Customers wait **only if chairs are available**.
 - No conflicts occur when multiple customers arrive at the same time.

Monitors:

- Monitor is a synchronization construct used to manage access to shared resources.
- It ensures that only one process can execute within a critical section at a time.
- It is a higher-level abstraction than semaphores.
- Monitors provide a structured way to combine data, operations, and synchronization rules into a single unit.
- They prevent race conditions and data corruption in concurrent programming.
- Monitors are often implemented directly within programming languages like Java and C#.
- They protect their internal variables from direct external access.
- Interaction with monitors is only allowed through their defined procedures.

Key features of monitors

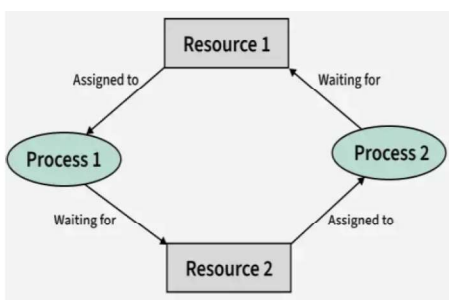
- **Mutual exclusion:** Monitors inherently guarantee that only one process can be active within the monitor's code at any given time, protecting shared data from concurrent access.
- **Encapsulation:** A monitor packages shared data and the procedures that operate on it into a single, unified entity, simplifying the design of concurrent systems.
- **Language-level construct:** Monitors are typically implemented as a feature of a programming language, unlike semaphores which are often provided by the operating system itself.
- **Protected variables:** Variables declared within a monitor cannot be accessed directly by processes outside the monitor; they can only be manipulated through the monitor's procedures.
- **Condition variables:** Monitors use condition variables to allow processes to wait for a specific condition to become true before proceeding, and to signal other processes when that condition has been met

Why monitors are used

- **Preventing race conditions:** By controlling access to shared resources, monitors prevent multiple processes from trying to modify the same data simultaneously, which can lead to unpredictable results.
- **Simplifying synchronization:** They provide a more structured and less error-prone alternative to using low-level primitives like semaphores directly.
- **Ensuring data integrity:** They maintain the consistency and integrity of shared data structures by enforcing a strict access protocol.

Q) Dead Locks?

Deadlock in an operating system is a situation where two or more processes are permanently blocked because each process is waiting for a resource that another process in the group is holding. As a result, none of the processes can continue execution.



Scenario: There are two processes: **Process P1, Process P2**
And two resources: **Resource R1, Resource R2**

Steps:

1. **P1 locks R1**
2. **P2 locks R2**
3. Now **P1** needs **R2** to continue — but **R2** is held by **P2**

4. **P2 needs R1 to continue** — but **R1** is held by **P1**, Result: Both processes are **waiting for each other forever** → **Deadlock**.

Q)Deadlock Characterization:

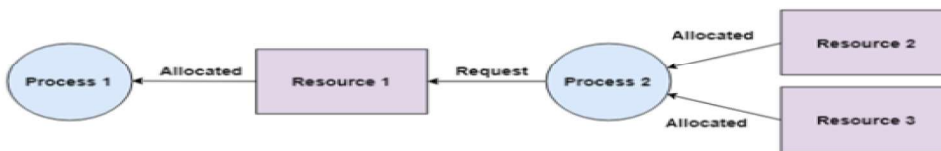
Deadlock Characterization in an Operating System **refers to the** conditions and methods used to describe, identify, and understand how a deadlock can occur.

Four Necessary Conditions for Deadlock:

1. Mutual Exclusion: There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.



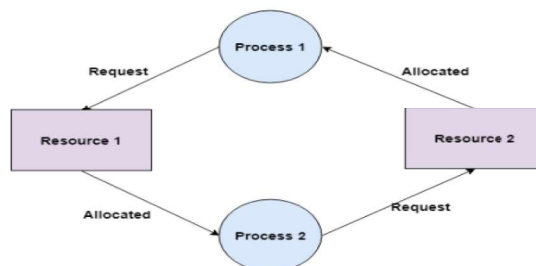
2. Hold and Wait: A can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



3. No Preemption: A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



4. Circular Wait: A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



Q) Deadlock handling: Deadlock handling methods are strategies used in operating systems to ensure processes do not remain permanently blocked, maintaining smooth execution and system reliability.

There are four approaches to dealing with deadlocks.



1. Deadlock Prevention

- This method completely avoids **the possibility of deadlocks** by designing the system in a safe way.
- The OS prevents deadlocks by ensuring that **at least one of the four necessary conditions for deadlock is never allowed to happen**.
- It uses **strict rules and restrictions** on how processes can request and use resources.
- Prevention often requires processes to **request all needed resources upfront**, which can reduce resource utilization.
- It may also require the OS to **preempt a resource** (in some methods) or make processes **release all resources** before requesting new ones.
- This method can sometimes be **conservative** because it may force processes to wait longer than necessary.
- It guarantees **total safety**, but sometimes at the cost of **system efficiency**.

2. Deadlock Avoidance

- The OS does **not block any deadlock condition completely**, but instead **analyzes each request** to ensure it won't lead to a deadlock.
- It keeps track of **how many resources each process may need in the future**.
- Before granting a resource, the OS simulates the outcome and checks whether the system will remain in a **safe state**.
- A **safe state** means there is at least one order in which all processes can finish without getting stuck.
- If a resource request may lead the system into an **unsafe state**, the OS **denies or delays** the request.
- Deadlock avoidance requires **advance knowledge** of how many resources each process may need, which may not always be practical.
- It is more **flexible and efficient** than deadlock prevention because it does not impose strict rules on all resource requests.
- However, it requires **complex calculations** and more **runtime overhead**.
- **Example: Banker's Algorithm**
- Works like a banker who only gives loans if confident that all customers (processes) can eventually repay (finish).
- Ensures the system stays safe by checking available resources, maximum needs, and current allocations.

3. Deadlock Detection & Recovery

- In this method, the OS doesn't stop deadlocks from happening. It allows them to occur and fixes the problem afterward.

- The system uses **deadlock detection algorithms** to regularly check if any processes are stuck in a deadlock cycle.
- Detection is usually done by examining a **resource allocation graph** or using a **detection matrix**.
- Once a deadlock is found, the OS chooses a strategy to **break it**.
- Recovery may involve **terminating one or more processes**, usually the one with the lowest priority or smallest cost.
- Another recovery method is **resource preemption**, where the OS temporarily takes a resource away from a process and restores it later.
- Processes might be **rolled back to a previous safe state** if possible.
- This method is useful in systems where deadlocks are **rare**, because it avoids the cost of constant prevention or avoidance.
- However, recovery can be **expensive**, especially if important or long-running processes are terminated.

4. Deadlock Ignorance

- This is the **simplest and cheapest** method for handling deadlocks.
- The OS **completely ignores** the possibility of deadlocks.
- It assumes that deadlocks are **rare**, so dealing with them is not worth the cost.
- If a deadlock occurs, the system usually **hangs or freezes**, and the user has to manually close or restart the program.
- It reduces **overhead**, as the system does not need to run complex algorithms for prevention or detection.
- This method works well in systems where users can **easily restart** programs without major loss.
- It is practical for **general-purpose operating systems** (e.g., Windows, Linux, macOS) because avoiding deadlocks in every situation would be too expensive.

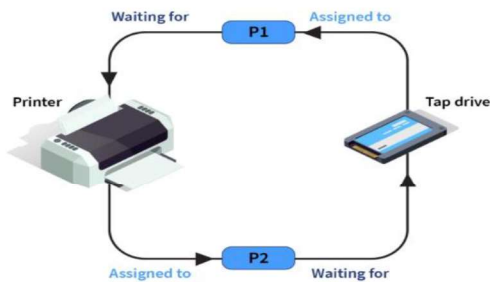
Q) Deadlock Avoidance?

- Deadlock Avoidance is a process used by the Operating System to avoid Deadlock.
- Let's first understand what is Deadlock in an Operating System is.
- Deadlock is a situation that occurs in the Operating System when any Process enters a waiting state because another waiting process is holding the demanded resource.
- Deadlock is a common problem in multi-processing where several processes share a specific type of mutually exclusive resource known as a soft lock or software.

HOW CAN AN OPERATING SYSTEM AVOID DEADLOCK?

- The operating system avoids Deadlock by knowing the maximum resource requirements of the processes initially, and also, the Operating System knows the free resources available at that time.

- The operating system tries to allocate the resources according to the process requirements and checks if the allocation can lead to a safe state or an unsafe state.
- If the resource allocation leads to an unsafe state, then the Operating System does not proceed further with the allocation sequence.



How does Deadlock Avoidance Work?

Let's understand the working of Deadlock Avoidance with the help of an intuitive example.

Process	Maximum Required	current Available	Need
P1	9	5	4
P2	5	2	3
P3	3	1	2

Let's consider three processes P1, P2, P3. Some more information on which the processes tell the Operating System are :

P1 process needs a maximum of 9 resources (Resources can be any software or hardware Resources like tape drive or printer etc..) to complete its execution. P1 is currently allocated with 5 Resources and needs 4 more to complete its execution.

P2 process needs a maximum of 5 resources and is currently allocated with 2 resources. So it needs 3 more resources to complete its execution.

P3 process needs a maximum of 3 resources and is currently allocated with 1 resource. So it needs 2 more resources to complete its execution.

The Operating System knows that only 2 resources out of the total available resources are currently free.

But only 2 resources are free now. Can P1, P2, and P3 satisfy their requirements? Let's try to find out.

As only 2 resources are free for now, only P3 can satisfy its need for 2 resources. If P3 takes 2 resources and completes its execution, then P3 can release its 3 (1+2) resources. Now the three free resources that P3 released can satisfy the need of P2. Now, P2 after taking the three free resources, can complete its execution and then release 5 (2+3) resources. Now five resources are free. P1 can now take 4 out of the 5 free resources

and complete its execution. So, with 2 free resources available initially, all the processes were able to complete their execution leading to a Safe State. The order of execution of the processes was <P3, P2, P1>.

Safe State and Unsafe State

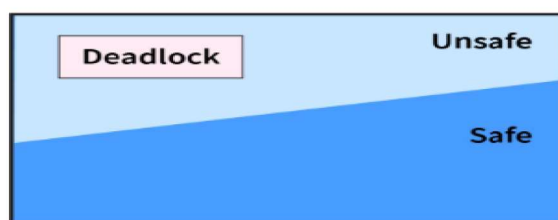
Safe State - In the above example, we saw that the Operating System was able to satisfy the needs of all three processes, P1, P2, and P3, with their resource requirements. So all the processes were able to complete their execution in a certain order like P3->P2->P1.

So, If the Operating System is able to allocate or satisfy the maximum resource requirements of all the processes in any order then the system is said to be in Safe State.

So safe state does not lead to Deadlock.

Unsafe State - If the Operating System is not able to prevent Processes from requesting resources which can also lead to a Deadlock, then the System is said to be in an Unsafe State.

Unsafe State does not necessarily cause deadlock it may or may not cause deadlock.



So, in the above diagram shows the three states of the System. An unsafe state does not always cause a Deadlock. Some unsafe states can lead to a Deadlock, as shown in the diagram.

How Deadlock Avoidance Prevents Deadlock

- By always checking for a **Safe State** before granting a resource request, the OS ensures that the system never enters a state that could definitely cause deadlock.
- Even if a request could potentially cause an unsafe state, the OS **delays allocation**, ensuring resources are allocated safely.
- This proactive approach guarantees that **deadlock cannot occur** as long as the OS strictly follows the safe state principle.

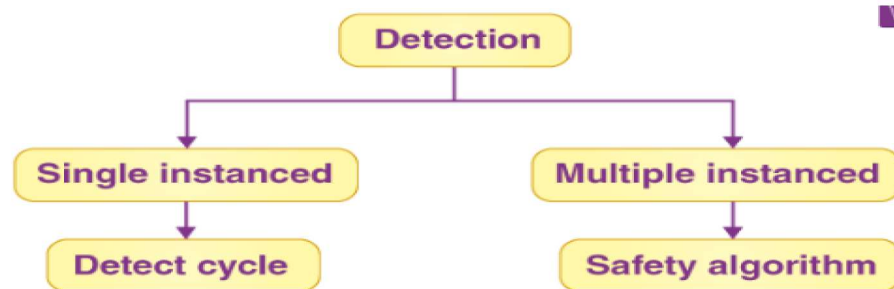
Q) Deadlock detection:

Deadlock detection is the method used by an operating system to **identify when a set of processes is permanently blocked** because each process is waiting for resources held by others, creating a deadlock situation.

How to Detect Deadlock in Operating Systems?

The OS does not use any mechanisms to avoid or prevent deadlocks in this approach. As a result, the system predicts that the deadlock will occur. The OS periodically scans the system for any deadlocks in order to avoid them. If any deadlocks are discovered, the OS will attempt to restore the system using several ways.

The OS's primary responsibility is to detect deadlocks. With the help of the resource allocation graph, the OS can detect deadlocks.



If each resource has only one instance, then **any cycle in the resource graph means a deadlock**. But if resources have **multiple instances**, a cycle does **not always** mean deadlock.

So, we convert the resource graph into an **allocation matrix** and a **request matrix**, and then use the **safety (deadlock-detection) algorithm** to check whether the system is actually in a deadlock.

Deadlock Detection Steps (Detailed)

A. When each resource has only ONE instance

1. The system uses a **Wait-For Graph**.
2. **Processes are nodes**, and an arrow $P1 \rightarrow P2$ means $P1$ is waiting for $P2$.
3. The OS **checks for cycles** in the graph.
4. **If a cycle exists → Deadlock is definitely present.**

B. When resources have MULTIPLE instances

1. The OS uses a detection algorithm similar to the **Banker's Algorithm**.
2. It uses three main tables:
 - **Available[]** → Free resources
 - **Allocation[][]** → Resources each process currently has
 - **Request[][]** → Resources each process still needs
3. Create two helper arrays:
 - **Work[] = Available[]** (temporary copy used for simulation)
 - **Finish[] = false** for all processes
 - If a process has **Allocation = 0**, set **Finish = true**
4. Find a process that:
 - Has **Finish = false**, and
 - **Request ≤ Work** (its needs can be met)
5. Pretend that process finishes:

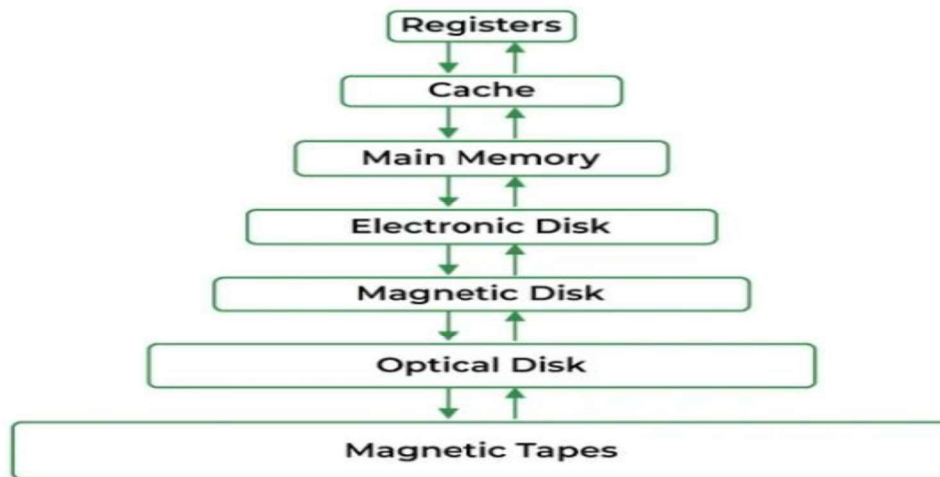
- Add its allocated resources back to **Work**
 - Set **Finish = true**
6. Repeat the search until:
 - **All processes have Finish = true** → **No deadlock**, OR
 - **No more processes can be satisfied** → **Deadlock exists**
 7. Any process with **Finish = false** at the end is **deadlocked**.

UNIT-3

Memory Management:

Memory management is a critical aspect of operating systems that ensures efficient use of the computer's memory resources. It controls how memory is allocated and deallocated to processes, which is key to both performance and stability. Below is a detailed overview of the various components and techniques involved in memory management.

The memory management hierarchy is defined as follows.



1. Registers

- Located inside the CPU.
- Fastest type of memory in a computer system.
- Stores data currently being executed or processed by the CPU.
- Very small in size (measured in bytes/words).
- Provides **highest speed** but **very limited capacity**.
- Access time is in nanoseconds or less.

2. Cache Memory

- High-speed memory located between CPU and main memory.
- Stores frequently accessed instructions and data.
- Reduces average time to access memory.
- Types: **L1, L2, L3 cache**.
- Faster than RAM but slower and larger than registers.
- Managed automatically by hardware.

3. Main Memory (RAM)

- Primary storage used by the CPU for currently running programs.
- Volatile memory (data lost when power is off).
- Larger capacity than cache but slower.
- Holds OS, application programs, and data in use.
- Access time slower than cache but still fast.

4. Electronic Disk (SSD)[no moving]

- Solid State Drive; uses flash memory.
- Faster than magnetic disks (HDDs).
- No mechanical moving parts → more reliable.
- Lower access time and latency.
- Used for secondary storage of data and programs.

5. Magnetic Disk (HDD)[moving]

- Traditional hard drives using rotating magnetic platters.
- Slower than SSD due to mechanical movement.
- High storage capacity at low cost.
- Non-volatile secondary storage.
- Used for long-term data storage.

6. Optical Disk

- Uses laser technology to read/write data.
- Types: CD, DVD, Blu-ray.
- Slower than SSDs(solid State Drive) and HDDs(Hard Disk Drive).
- Mostly used for media distribution and backups.
- Low cost and portable storage.

7. Magnetic Tapes

- Oldest and slowest storage medium.
- Very high capacity at extremely low cost.
- Sequential access (not random access).
- Mostly used for **archival and long-term backups**.
- Best for storing large volumes of rarely accessed data.

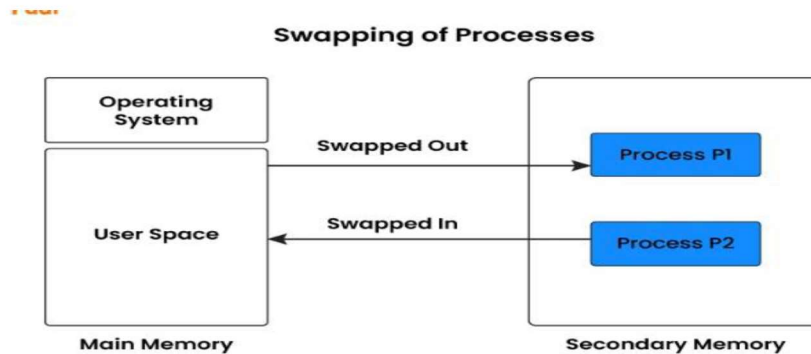
Q) Swapping in Operating System: -

- Swapping is a way to **manage memory efficiently**.
- It happens when **RAM is full** and the system needs more space.
- The OS temporarily moves a program or data that is **not currently being used** to a special area on the hard disk called **swap space**. [finally it moves the data into hard disk]
- The process of moving data **back and forth between RAM and swap space** is called **swapping**.
- Swapping allows the computer to **run multiple programs at the same time**, even if there isn't enough RAM.
- This ensures that **active programs have enough memory** to run properly.
- Proper swap space management helps **reduce memory pressure** and ensures smooth execution of programs.

Example

- **P1** is in RAM, but **P2** needs to run.
- The OS **swaps out P1** to the swap space to free RAM for **P2**.
- **P2** is loaded into RAM and starts execution.
- When **P1** is needed again, the OS **swaps it back** from swap space.

This process ensures that all programs can run efficiently, even if RAM is limited.



Phases of Swapping in Operating System: Swapping in OS occurs in two phases. These phases are responsible for moving data between main memory and the swap space that is located on the hard disk. The two phases of swapping work together so that the system can handle more programs to execute than the amount of space that is available in the RAM.

1. Swap In Phase in Operating System: The Swap In phase is the process of moving a program or data back into main memory from the swap space. This takes place when a program that was moved out is needed again. The operating system checks if there is space available for the program in the RAM. When the available space is found, the data is loaded back into main memory so that it can be used again by the processor. This phase is very important because it allows the user to continue working on a program without noticing any delay.

2. Swap Out Phase in Operating System: The Swap Out phase is the process of moving data or a program from the main memory to swap space on the hard disk. This would usually occur when the system has run out of memory and has to swap out a program to free up memory for a priority program to be executed. The operating system will look for a program that is not being used at this moment and swap the program to the swap area.

Working of Swapping in Operating System

Step 1: Process Selection for Swapping

The operating system monitors all active processes and identifies those that are idle or not frequently used. These selected processes are removed from the main memory to make space for other important or high-priority programs. This helps ensure that memory is allocated efficiently and that essential tasks continue to run without interruption.

Step 2: Data Swapping to Swap Space

Once a process is selected, the operating system transfers its data from RAM to a designated area on the hard disk known as swap space. This process frees up memory so that other programs currently in use can function smoothly, and the efficiency of the swapping affects the overall swap performance. The data is safely stored in the swap space and remains accessible when the process needs to run again.

Step 3: Process Suspension

After the data has been moved to the swap space, the process is marked as suspended and enters a waiting state. While in this state, the process does not receive any CPU time or system resources. The operating system keeps track of the process's location and status so that it can be easily restored when needed.

Step 4: Reloading the Process

When the user or the system requests the suspended process again, the operating system first checks if there is enough free memory available in RAM. If the memory is still full, the system may repeat the swapping process by moving another inactive process to swap space. The suspended process is then reloaded into memory only after the operating system confirms that enough space has been made available.

Step 5: Process Resumption

Once the process has been successfully loaded back into memory, it resumes its execution from the exact point where it was paused. The CPU continues processing the task without any data loss or reset. To the user, the process appears to run continuously, maintaining the smooth and efficient operation of the system.

Advantages of Swapping in OS

1. **Efficient Memory Use:** Swapping ensures that the memory is used efficiently by moving the program that is unused to the hard disk.
2. **Smooth Multitasking:** It allows the user to run many programs at the same time without affecting the performance of the system.
3. **System Stability:** Swapping helps in preventing the system from crashing when the memory is full. It keeps the system stable by moving the unused data out of the RAM during heavy usage.
4. **Support of Large applications:** Programs requiring more memory than is physically available in RAM can still execute effectively because there is additional space that is generated through the swapping process, moving the unused information.
5. **Improved User Experience:** Swapping enhances system responsiveness by minimizing delays during multitasking.

Disadvantages of Swapping in OS

1. **Not Good for Real-Time Systems:** Swapping is not suitable for systems that must respond quickly, as the delays caused due to swapping could affect performance.
2. **Reduced Performance:** Swapping can reduce performance since it takes time to move data from main memory to the hard disk.
3. **Increased Disk Activity:** Frequently swapping increases the read and write operations in the hard disk, which can affect the hard disk over time.

4. Increased CPU Overhead: The processor has to spend additional time managing swap operations, which may decrease efficiency for other tasks.
5. Temporary Freezes: If a system keeps swapping multiple processes in and out, the system temporarily freezes.

Contiguous memory allocation

Introduction:

- Contiguous memory allocation is a memory management technique used by operating systems.
- It allocates memory to processes in a single, continuous block.
- The allocated memory for a process is adjacent and contiguous in physical RAM.
- This technique helps ensure efficient use of memory.
- It minimizes fragmentation and reduces wasted memory space.
- It provides fast access to memory because all data is stored in a continuous region.
- Management of memory is simpler compared to other allocation techniques.
- However, it can suffer from external fragmentation.
- It may require large contiguous blocks, which are not always available.
- It often imposes fixed block-size limitations.
- Despite these drawbacks, it remains an important and widely used memory allocation method in modern operating systems.

Definition:

Continuous memory allocation is a memory management technique in which memory is given to processes in one continuous block. This helps use memory efficiently, with very little fragmentation or waste. It also makes memory management easier for the operating system because it can work with larger blocks, resulting in faster memory access and better system performance. Overall, continuous memory allocation is an important technique that helps operating systems manage memory effectively and ensure efficient use of resources.

Purpose of memory management techniques

- The purpose of contiguous memory allocation is to efficiently manage the memory resources of a computer system.
- It allocates memory to processes in continuous blocks, helping reduce fragmentation and wasted memory.
- Contiguous blocks allow faster access to memory compared to non-contiguous blocks.
- This technique simplifies memory management because the operating system can work with larger memory blocks instead of many small, scattered ones.
- Managing fewer, larger blocks reduces overhead and improves overall system performance.
- The main goal is to ensure memory is used efficiently and managed effectively for better system speed.
- It is especially useful for applications that need a lot of memory, such as multimedia editing and gaming.

- It is also beneficial for systems with a small number of processes that require fast memory access.

Characteristics of Contiguous Memory Allocation

Efficient use of memory: Contiguous memory allocation is efficient in terms of memory utilization, as there is no internal fragmentation within a process's allocated memory block.

Easy to manage: Contiguous memory allocation is easy to manage, as the operating system can quickly allocate and de-allocate memory to processes by assigning contiguous blocks.

External fragmentation: One of the main drawbacks of contiguous memory allocation is external fragmentation, which occurs when small gaps of free memory are scattered throughout the memory space. Over time, these small gaps can accumulate and result in larger portions of memory becoming unusable, even though the total amount of free memory may be sufficient to satisfy a request for memory.

Compaction: To address external fragmentation, operating systems may use techniques such as compaction, where the operating system rearranges memory blocks to eliminate gaps and consolidate free memory into larger contiguous blocks.

Overall, contiguous memory allocation is an efficient and easy-to-manage memory management technique, but it may suffer from external fragmentation over time. Operating systems may use various techniques to reduce external fragmentation and ensure that memory is utilized effectively.

Advantages:

Efficient memory utilization: Contiguous memory allocation is efficient in terms of memory utilization, as there is no internal fragmentation within a process's allocated memory block.

Simple and easy to manage: This technique is simple and easy to manage, as the operating system can quickly allocate and de-allocate memory to processes by assigning contiguous blocks.

Fast access : Since the memory is allocated in contiguous blocks, access to the memory is faster than other memory management techniques.

Disadvantages

External Fragmentation: One of the main disadvantages of contiguous memory allocation is external fragmentation, which occurs when small gaps of free memory are scattered throughout the memory space.

Limited memory capacity: Contiguous memory allocation is limited by the size of the memory blocks available on the system, which may limit the total amount of memory that can be allocated to a process.

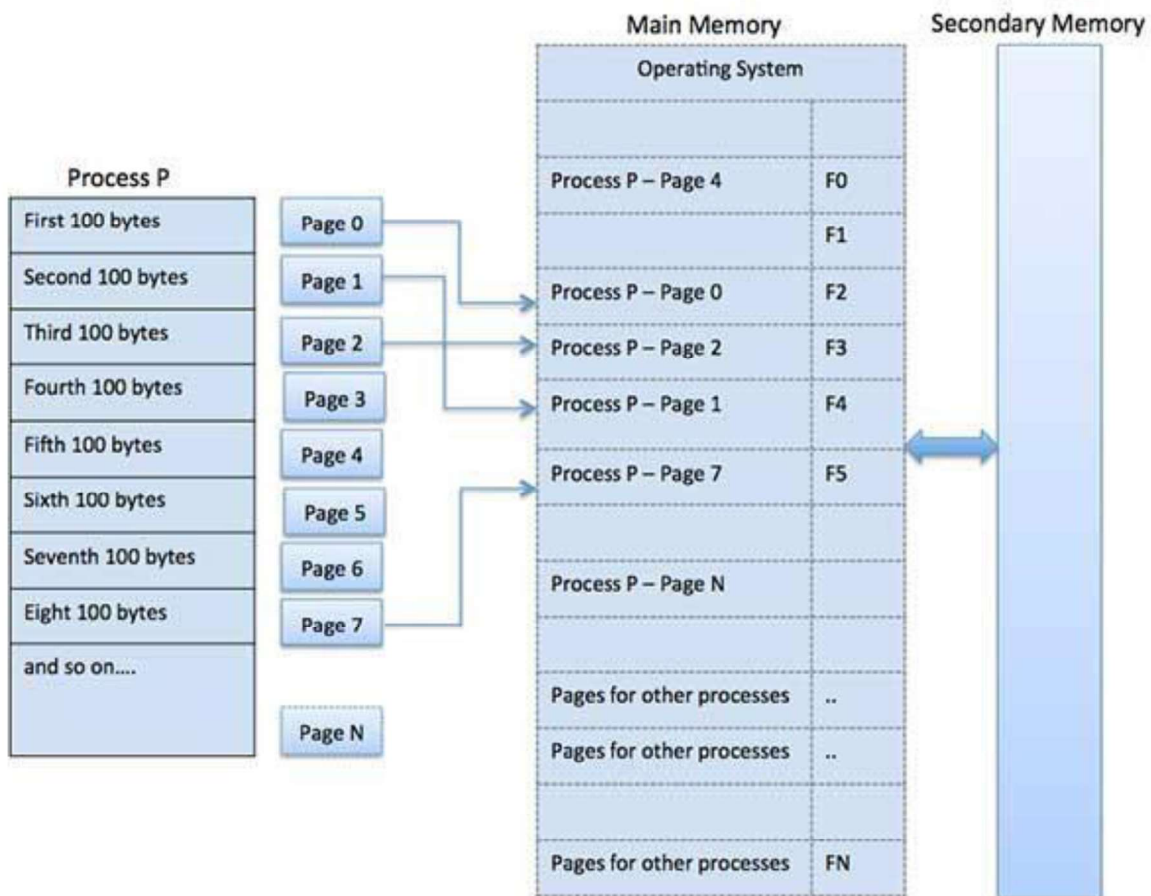
Difficulty in sharing memory: This technique makes it difficult to share memory between multiple processes, as each process is assigned a contiguous block of memory that cannot be shared with other processes.

Lack of flexibility: Contiguous memory allocation lacks flexibility in allocating and de-allocating memory, as the operating system can only allocate memory in contiguous blocks.

Paging:

Paging is a **memory management technique** used by operating systems to eliminate the problem of contiguous memory allocation. In paging, the **physical memory is divided into fixed-size blocks** called **frames**, and the **logical memory (process memory)** is divided into blocks of the **same size** called **pages**. Because pages and frames are of equal size, any page can fit into any frame in the physical memory.

The main purpose of paging is to **remove external fragmentation** and to allow a process's memory to be stored **non-contiguously** in different locations of physical memory. This allows efficient use of memory and enables processes to execute even when sufficient contiguous blocks are not available.



1. Process P (Logical Memory Space): This section on the left side shows how the operating system views the memory of a process.

- A process (Process P) contains instructions and data that must be executed by the CPU.
- Instead of loading the whole process into memory at once, the OS divides the process into **fixed-size units called pages**.
 - For example:
 - The first 100 bytes → **Page 0**
 - The second 100 bytes → **Page 1**
 - The third 100 bytes → **Page 2**, and so on.
 - All pages are **equal in size** (100 bytes in this example), which makes memory management easier for the OS.

Why this is done?

Because it avoids the need for one continuous large block of memory. Dividing into pages means the process can fit into memory more flexibly.

2. Pages (Middle Section: Page List): This section shows the **logical pages** created from Process P.

- After dividing the process into fixed-size blocks, each block is assigned a page number.
- These pages exist conceptually (logical memory), not physically.
- Examples:
 - Page 0 contains the first 100 bytes.
 - Page 1 contains the next 100 bytes.
 - Page 2 contains the next 100 bytes, etc.
- The page numbers are simply labels the OS uses to identify each part of the process.

Purpose: Pages allow the OS to load only those parts of the process that are currently required into physical memory, improving efficiency and reducing memory waste.

3. Main Memory (Frames F0, F1, F2, F3... FN): This section is the **physical memory (RAM)** where actual execution happens. Because

- RAM is divided into **frames**, which are also fixed-size blocks, **exactly the same size as pages**.
- Frames are marked as F0, F1, F2, F3... etc.
- A page can be placed into **any free frame**, not necessarily in order.

For example, the diagram shows:

- Page 4 stored in Frame F0
- Page 0 stored in Frame F2
- Page 2 stored in Frame F3
- Page 1 stored in Frame F4
- Page 7 stored in Frame F5

This means the process is **not stored continuously**, but rather scattered in different parts of main memory.

Purpose:

- It removes the limitation of needing a continuous physical block.
- Even if free memory is scattered, the process can still be loaded.
- It **eliminates external fragmentation**.

The bottom part of main memory shows frames that belong to **other processes**, proving that multiple processes can coexist using paging.

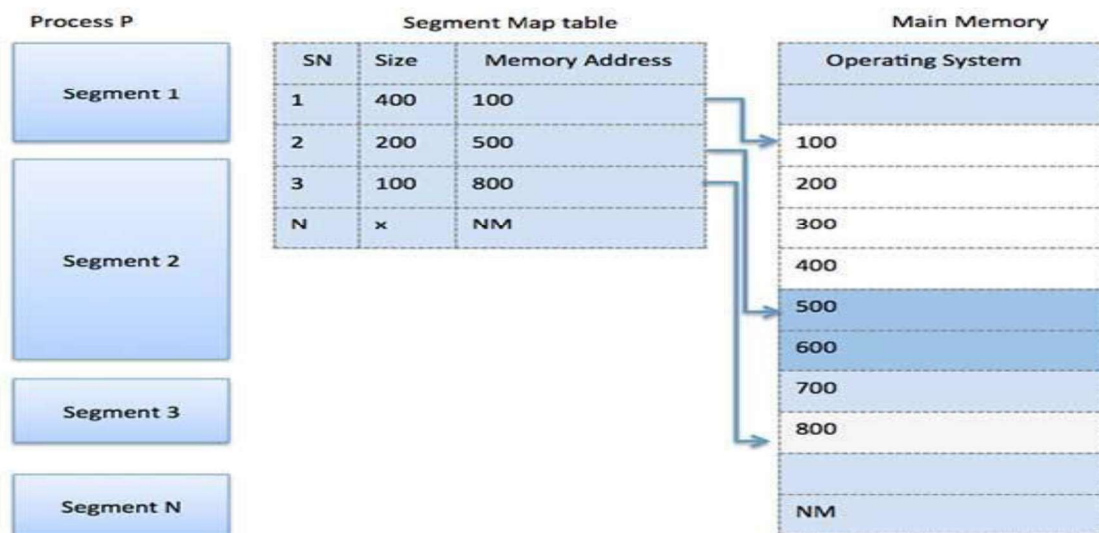
4. Secondary Memory (Backing Store): This is the storage area where unused pages are kept when they are not loaded into RAM. Some major information's are defined as follows.

- Secondary memory (usually a hard disk or SSD) stores all pages of the process.
- Only necessary pages are brought into RAM at any time.
- If a page is needed but currently not in memory, a **page fault** occurs.
- The OS then fetches the required page from secondary memory into a free frame in main memory.

Purpose:

- RAM is limited, so only essential pages are kept in memory.
- Secondary memory ensures the entire process is still available even when RAM space is insufficient.

Segmentation: Segmentation in an operating system is a **memory management technique** in which a program is divided into **logical units called segments**. Each segment represents a meaningful part of the program—such as code, data, stack, functions, or modules—rather than being split into fixed-size blocks.



Why Segmentation is necessary?

1. Segmentation emerged as a response to challenges observed in the paging technique. [To overcome the paging technique].
2. Paging in Operating System involves dividing a function or code into pages, often without consideration for the potential division of related code segments.

Imagine a program where:

- **Function A calls Function B many times. But paging may place:**
 - **Half of Function A in Page 1**
 - **The other half in Page 2**
 - **Function B in Page 3**

Paging does **not** care about functions—it only cares about keeping all pages the same size.

3. Consequently, when a process is in execution, the CPU must load multiple pages into frames to ensure that the relevant code is available for execution.

4. More pages are loaded into the main memory, making paging less efficient.

5. In contrast, Segmentation was introduced as a solution.

6. It involves dividing the code into distinct modules, ensuring that related code segments can be grouped within a single block.

7. This modular approach enhances efficiency by allowing the CPU to load only the necessary elements, reducing the number of pages required to load a process into the main memory.

8. Another significant drawback of various memory management techniques is separating the actual view of physical memory and the user's perception of physical memory.

9. Segmentation addresses this issue by organizing the user's program into segments based on specific requirements, providing a more intuitive and tailored approach to memory management.

What is segment table?

1. Segment Table: A segment table is an important data structure used to manage segments in a program efficiently.

2. Purpose: It helps track information about each segment, such as its base address and limit.

3. Role in OS: The segment table is crucial for the Operating System (OS) because it allows easy access to and management of different segments in the program.

Elements of segment table

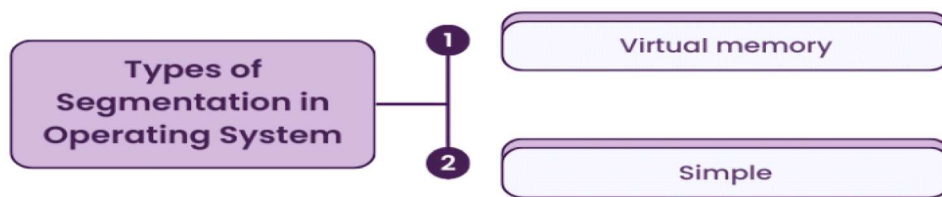
Elements of segment table



1) Segment base: This crucial element denotes the starting address of a particular segment in memory. It is a foundational reference point for efficient memory retrieval during program execution.

2) Segment limit: This element delineates the size or limit of the segment, establishing clear boundaries within which the segment operates. It plays a pivotal role in preventing unintended memory overflows.

Types of Segmentation in OS: The following describes the types of Segmentation in Operating System (OS):



1) Virtual memory: This variant of Segmentation leverages virtual memory techniques to manage segments. It introduces an additional abstraction layer in memory organization, enhancing the system's adaptability to varying demands.

Imagine your laptop has **8 GB RAM**, but you open:

- Chrome (3 GB)
- VS Code (2 GB)
- A game (4 GB)
- Spotify (500 MB)

Total = **9.5 GB**, which is more than your 8 GB RAM.

How virtual memory helps

The OS moves **unused parts** of apps from RAM to the **swap space/page file** on the disk.

This creates the illusion that you have more memory than physically installed.

- ✓ Apps don't crash
- ✓ They think they have plenty of memory
- ✓ System still runs smoothly (maybe slower)

2) Simple: As the name suggests, this represents the fundamental form of Segmentation. This approach divides the program into segments without incorporating intricate memory management techniques, offering a straightforward yet effective memory organization.

A C program automatically divided into three basic segments:

1. **Code Segment** – stores instructions
2. **Data Segment** – stores global & static variables
3. **Stack Segment** – stores function calls, local variables, return addresses

The operating system divides this program into **simple, fixed-purpose segments**:

Code Segment : Contains all executable instructions

- `main()`
- `display()`
- `printf()` calls

Data Segment: Contains global and static variables

- `x = 10`

Stack Segment: Contains dynamic data created at runtime

- Local variables `y` and `z`
- Return addresses for function calls

Benefits of Segmentation in Operating System (OS):The following are the benefits of Segmentation in Operating System (OS):

- 1) Elimination of internal fragmentation:** One notable advantage of Segmentation lies in its ability to eradicate internal fragmentation. By dividing processes into modules of variable sizes, Segmentation ensures that memory is utilized efficiently without wasting space within individual segments.
- 2) The space efficiency of segment table:** Compared to the page table used in paging, the segment table in Segmentation consumes less space. This compact representation contributes to a more efficient utilization of memory management resources, which can be further optimized with effective CPU scheduling in Operating System.
- 3) Enhanced CPU utilization:** Segmentation enhances CPU utilization by loading complete modules into memory all at once. This approach ensures that the related code segments necessary for execution are readily available, optimizing the processing efficiency of the CPU.
- 4) User-friendly memory perception:** Segmentation aligns the user's perception of physical memory closely with the program's organizational structure. Users can conceptually divide their programs into modules through Segmentation, where each module encapsulates the codes of distinct processes.
- 5) User-specified segment size:** In Segmentation, users can specify the size of each segment, providing a customized approach to memory allocation. This stands in contrast to paging, where the hardware dictates the fixed size of pages.
- 6) Data and security segregation:** Segmentation serves as a method to segregate data, contributing to security operations. This capability allows for a structured separation of sensitive data, enhancing the security protocols within the system.
- 7) Flexibility in memory allocation:** Segmentation offers a higher degree of flexibility compared to paging. The variable size of segments and the ability to design processes with multiple components enable a more fine-grained and adaptable approach to memory allocation.
- 8) Sharing capabilities:** Segmentation facilitates the sharing of memory segments between processes. This feature proves valuable for inter-process communication or sharing code libraries, fostering collaboration and resource optimization.
- 9) Protection mechanism:** Segmentation introduces a protective layer between segments, preventing unauthorized access or modification of one process's memory segment by another. This inherent protection enhances the overall security and stability of the system.

Q) Virtual Memory

Virtual memory is a memory management technique used by the operating system (OS) that allows programs to use more memory than physically available in the system (RAM). This is achieved by using disk space (usually a hard drive or SSD) as temporary "extra" memory.

Example: Running Google Chrome with many tabs open on a 4 GB RAM computer

Suppose your computer has **4 GB of physical RAM**, but you open:

- Google Chrome (20 tabs) → needs 3 GB
- Microsoft Word → needs 1 GB
- Spotify → needs 0.5 GB
- Other background apps → need 1 GB

Total required memory = **5.5 GB**, but your RAM = **4 GB** → Not enough!

How Virtual Memory Help: The operating system uses a feature called **virtual memory** to make your system work **even when you don't have enough RAM**.

What the OS does:

- It stores the most active data in **RAM**.
- It moves less-used data (background tabs or idle apps) to **disk memory (swap file or page file)**.
- This gives your computer the *illusion* that it has **5.5 GB of memory**, even though it only has 4 GB of RAM.

Several important **components** or techniques are used to make virtual memory work effectively:

1. **Demand Paging:**
2. **Page Replacement:**
3. **Allocation of Frames:**

What is Demand Paging in Operating Systems?

Demand Paging is a memory management technique used by modern operating systems to implement virtual memory. It allows programs to use more memory than physically available in RAM by only loading the portions of a program (pages) into memory when they are actually needed during execution. This helps optimize the use of limited physical memory and makes the system more efficient.

Simple Example: Opening a PDF File Page by Page

Imagine you open a **100-page PDF** in a PDF viewer (like Adobe Reader). The total PDF size may be **50 MB**.

But when you open it:

✓ **The OS does NOT load all 50 MB into RAM.**

Instead, it loads only:

- The first page
- Viewer interface
- Basic rendering code

Let's say only **5 MB** of pages are loaded.

How Demand Paging Works Here

You scroll to Page 2

- Page 2's data is **not yet in RAM**
- A **page fault** occurs
- OS loads **only Page 2** from disk to RAM

You scroll to Page 10

- Pages 3–9 might not be loaded
- More page faults
- OS loads pages 10's data as needed

You never scroll to Page 80

Those pages are **never loaded into RAM**, saving memory.

Why This Example Is Simple and Effective

- Easy to visualize
- Shows that only *needed* pages are loaded
- Demonstrates page faults
- Avoids loading the entire file

Key Concepts in Demand Paging:

1. Virtual Memory: In a system using demand paging, programs are given the illusion of having access to a large, continuous block of memory, even if the physical RAM is much smaller. Pages and frames are the units of memory management. Pages refer to blocks of memory in the program (typically 4KB in size), and frames refer to corresponding blocks in physical RAM.

2. Lazy Loading: In demand paging, instead of loading the entire program into memory when it starts, the operating system only loads the specific pages the program actually uses at runtime. This is known as lazy loading or paging on demand.

If a page is required and it is not already in memory, the system triggers a page fault and loads the page from secondary storage (like a hard drive or SSD) into RAM.

Example: Loading a Webpage

Imagine you open a **long webpage** with images, text, and videos.

How Lazy Loading Works

1. You open the page

- The browser **does NOT load all images and videos at once**.
- Only the text and the images/videos visible on your screen (above the fold) are loaded.

2. You scroll down

- The browser detects that new content is about to appear
- It **triggers a page fault** for the unloaded images/videos
- Loads only the images/videos you are about to see

3. You never scroll to the bottom

- Images or videos in sections you never reach **are never loaded**
- Saves memory, bandwidth, and improves performance

Why This Example Works Well

- Everyone can relate to it
- Shows **lazy loading clearly** in action
- Demonstrates **loading on demand**, just like in OS memory management

How Demand Paging Works:

a step-by-step breakdown of how demand paging works in an operating system:

1. Program Execution: When a program starts, the OS does not load the entire program into RAM. Instead, the program is divided into pages and only the pages needed for the initial execution are loaded into memory. [It load only required pages]

2. Page Fault: When the program attempts to access a page that is not currently in RAM, a page fault occurs. This is a hardware interrupt triggered by the processor when it cannot find the required page in memory.

The OS must handle the page fault by loading the missing page from disk storage into an available frame in RAM.

Example of a Page Fault

Suppose a student opens a very large PDF file. The operating system does not load the entire PDF into RAM at once; instead, it loads only the first few pages that the student is viewing.

Now the student scrolls to page **50** of the PDF for the first time.

- Page 50 is **not currently in RAM**.
- The CPU tries to access that page.
- The **Memory Management Unit (MMU)** looks in the page table and sees that page 50 is marked as “**not present**”.
- This triggers a **page fault**.
- The OS pauses the program, loads page 50 from the disk (storage) into a free frame in RAM, updates the page table, and then resumes the program.

The student sees page 50 load after a short delay — this delay is caused by the **page fault handling**.

3. Page Swap: If RAM is full, the OS must decide which page in memory to swap out to make room for the new page. The OS selects a page to move from RAM to disk (this process is managed by page replacement algorithms, such as LRU (Least Recently Used) or FIFO (First In, First Out)).

Once space is freed up, the required page is loaded into memory, and the program can continue execution.

4. Address Translation: To map the virtual memory addresses used by a program to physical memory addresses (in RAM), the OS uses a page table. The page table stores the mapping between virtual pages and physical frames.

When a page fault occurs, the OS looks up the page table to find the location of the requested page on disk and loads it into the corresponding frame in physical memory.

Benefits of Demand Paging:

1. Efficient Memory Usage: Demand paging allows the OS to load only the necessary parts of a program into memory, saving space and ensuring that physical memory is used efficiently.

Programs don't need to occupy the full memory space at once, which is especially useful for large programs or systems with limited RAM.

2. Increased Program Size: Since only parts of a program are loaded at a time, the OS can support programs that are much larger than the available physical memory. This is essential in modern systems where applications can be very large (e.g., games, video editing software, etc.).

3. Faster Startup: Programs can start running more quickly because the OS only needs to load the initial pages, rather than the entire program, into memory. This reduces startup time.

Drawbacks and Challenges of Demand Paging:

1. Page Fault Overhead: When a page fault occurs, the OS has to stop the program and load the required page from disk into memory. Disk access is much slower than memory access, which can cause a performance penalty, especially if page faults occur frequently (this is called thrashing).

2. Complexity: Managing page faults and swapping pages in and out of memory introduces additional complexity in both the operating system's memory management and hardware.

3. Latency (delay): If a program frequently accesses pages that are not in memory, the time spent handling page faults can slow down overall performance. This issue is often mitigated by using smart page replacement algorithms to keep commonly used pages in memory.

2. What is Page Replacement:

In an operating system that uses paging, a page replacement algorithm is needed when a page fault occurs and no free page frame is available. In this case, one of the existing pages in memory must be replaced with the new page.

The virtual memory manager performs this by:

1. Selecting a victim page using a page replacement algorithm.
2. Marking its page table entry as "not present."
3. If the page was modified (dirty), writing it back to disk before replacement.

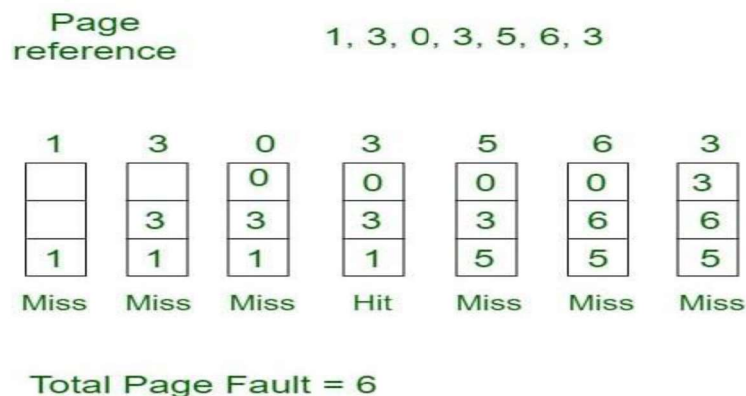
The efficiency of a page replacement algorithm directly affects the page fault rate, which in turn impacts system performance.

Common Page Replacement Techniques

1. First In First Out (FIFO)
2. Optimal Page replacement
3. Least Recently Used (LRU)
4. Most Recently Used (MRU)

1. First in First Out (FIFO): This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Example 1: Consider page reference string 1, 3, 0, 3, 5, 6, 3 with 3-page frames. Find the number of page faults using FIFO Page Replacement Algorithm.



- Initially, all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots ---> 3 Page Faults.
- When 3 comes, it is already in memory so ---> 0 Page Faults.
- Then 5 comes, it is not available in memory, so it replaces the oldest page slot i.e 1. ---> 1 Page Fault.
- 6 comes, it is also not available in memory, so it replaces the oldest page slot i.e 3 ---> 1 Page Fault.
- Finally, when 3 come it is not available, so it replaces 0, 1-page fault.

Step	Page	Action	Frame Status	Page Fault
1	1	Empty → Load	1	1
2	3	Empty → Load	1 3	1
3	0	Empty → Load	1 3 0	1
4	3	Already exists	1 3 0	0
5	5	Replace 1	5 3 0	1
6	6	Replace 3	5 6 0	1
7	3	Replace 0	5 6 3	1

2. Optimal Page Replacement: In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Example: Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4-page frame. Find number of page fault using Optimal Page Replacement Algorithm.

Page reference	7,0,1,2,0,3,0,4,2,3,0,3,2,3														No. of Page frame - 4
7	0	1	2	0	3	0	4	2	3	0	3	2	3		
			2	2	2	2	2	2	2	2	2	2	2		
		1	1	1	1	1	4	4	4	4	4	4	4		
	0	0	0	0	0	0	0	0	0	0	0	0	0		
7	7	7	7	7	3	3	3	3	3	3	3	3	3		
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit	Hit	Hit	Hit	Hit	Hit		
Total Page Fault = 6															

- Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots ---> 4 Page faults
- 0 is already there so ---> 0 Page fault. when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future---> 1 Page fault.
- 0 is already there so ---> 0 Page fault. 4 will takes place of 1 ---> 1 Page Fault.

Now for the further page reference string ---> 0 Page fault because they are already available in the memory. Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

3. Least Recently Used: In this algorithm, page will be replaced which is least recently used.
Example Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4-page frames. Find number of page faults using LRU Page Replacement Algorithm.

Page reference	7,0,1,2,0,3,0,4,2,3,0,3,2,3													No. of Page frame - 4	
7	0	1	2	0	3	0	4	2	3	0	3	2	3		
			2	2	2	2	2	2	2	2	2	2	2		
		1	1	1	1	1	4	4	4	4	4	4	4		
	0	0	0	0	0	0	0	0	0	0	0	0	0		
7	7	7	7	7	3	3	3	3	3	3	3	3	3		
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit	Hit	Hit	Hit	Hit	Hit		
Total Page Fault = 6															

Here LRU has same number of page fault as optimal but it may differ according to question.

- Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots ---> 4 Page faults.
- 0 is already there so ---> 0 Page fault. when 3 came it will take the place of 7 because it is least recently used ---> 1 Page fault.
- 0 is already in memory so ---> 0 Page fault.
- 4 will takes place of 1 ---> 1 Page Fault.
- Now for the further page reference string ---> 0 Page fault because they are already available in the memory.

4. Most Recently Used (MRU): In this algorithm, page will be replaced which has been used recently. Belady's anomaly can occur in this algorithm.

Example 4: Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4-page frames. Find number of page faults using MRU Page Replacement Algorithm.

Page reference	7,0,1,2,0,3,0,4,2,3,0,3,2,3													No. of Page frame - 4	
7	0	1	2	0	3	0	4	2	3	0	3	2	3		
			2	2	2	2	2	2	3	0	3	2	3		
		1	1	1	1	1	1	1	1	1	1	1	1		
	0	0	0	0	3	0	4	4	4	4	4	4	4		
7	7	7	7	7	7	7	7	7	7	7	7	7	7		
Miss	Miss	Miss	Miss	Hit	Miss	Miss	Miss	Hit	Miss	Miss	Miss	Miss	Miss		
Total Page Fault = 12															

- Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots ---> 4 Page faults
- 0 is already there so---> 0 page fault
- when 3 comes it will take place of 0 because it is most recently used ---> 1 Page fault
- when 0 comes it will take place of 3 ---> 1 Page fault
- when 4 comes it will take place of 0 ---> 1 Page fault
- 2 is already in memory so ---> 0 Page fault

- when 3 comes it will take place of 2 ---> 1 Page fault
- when 0 comes it will take place of 3 ---> 1 Page fault
- when 3 comes it will take place of 0 ---> 1 Page fault
- when 2 comes it will take place of 3 ---> 1 Page fault
- when 3 comes it will take place of 2 ---> 1 Page fault

The conclusion of the MRU Page Replacement Algorithm for the given reference string is that MRU performs poorly compared to other algorithms like LRU and Optimal. Since MRU always replaces the **most recently used page**, it often removes pages that will be needed again soon, resulting in a higher number of page faults. In this example, the MRU algorithm produces **12 total page faults**, which clearly shows its inefficiency. Additionally, MRU can exhibit **Belady's anomaly**, meaning page faults may increase even when the number of frames is increased, making it an unreliable page replacement strategy in many cases.

4. Allocation of Frames:

Allocation of frames is the method the OS uses to give each process a certain number of frames (pieces of physical memory) so that it can run efficiently without too many page faults.

Example of Allocation of Frames: Suppose a computer has **100 frames** of physical memory (RAM), and **4 processes** are running:

Process	Size (in pages)
P1	10
P2	20
P3	30
P4	40

1. Equal Allocation

- Each process gets **an equal share** of frames.
- Total frames = 100, total processes = 4 → **each process gets 25 frames**.

Process	Frames Allocated
P1	25
P2	25
P3	25
P4	25

Outcome: Simple and fair in terms of equal distribution, but P4 may need more frames than P1, and P1 may not need all 25 frames → some inefficiency.

2. Proportional Allocation

- Frames are allocated **based on the size of each process**.
- Total pages = $10 + 20 + 30 + 40 = 100$ pages
- Frames = 100 → Allocate proportional to process size

Process	Size	Frames Allocated
P1	10	10
P2	20	20
P3	30	30
P4	40	40

Outcome: Each process gets just enough frames according to its size → **reduces page faults and improves efficiency**.

Finally, OS decides **how many frames each process gets** so that all processes can run efficiently without overloading memory or causing too many page faults.

The main memory of the system is divided into frames. The OS has to allocate a sufficient number of frames for each process and to do so, the OS uses various algorithms.

The five major ways to allocate frames are as follows:

1. Proportional frame allocation:

The proportional frame allocation algorithm allocates frames based on the size that is necessary for the execution and the number of total frames the memory has.

The only disadvantage of this algorithm is it does not allocate frames based on priority. This situation is solved by **Priority frame allocation**.

Suppose a computer has **100 frames of RAM** and **3 processes**:

Process	Size (pages)
P1	10
P2	20
P3	30

Step 1 — Calculate total pages:

$$10 + 20 + 30 = 60 \text{ pages}$$

Step 2 — Allocate frames proportionally:

- P1: $(10 \div 60) \times 100 = \sim 17$ frames
- P2: $(20 \div 60) \times 100 = \sim 33$ frames
- P3: $(30 \div 60) \times 100 = \sim 50$ frames

Outcome:

- Bigger processes get more frames, smaller processes get fewer frames.
- Each process gets memory according to its size.

Limitation: If P1 was a high-priority process, it still only gets 17 frames. Proportional allocation **doesn't consider priority**. That's where **Priority Frame Allocation** would be used.

2. Priority frame allocation

Priority frame allocation allocates frames based on the priority of the processes and the number of frame allocations. If a process is of high priority and needs more frames then the process will be allocated that many frames. The allocation of lower priority processes occurs after it.

Suppose a computer has **100 frames of RAM** and **3 processes** with different priorities:

Process	Priority	Pages Needed
P1	High	30
P2	Medium	40
P3	Low	50

Step 1 — Allocate frames based on priority

- P1 is high priority, so it is allocated frames first: 30 frames.
- P2 is medium priority, allocated next: 40 frames.
- P3 is low priority, allocated last: remaining 30 frames (not full 50 needed).

Outcome:

- **High-priority processes get memory first.**
- **Lower-priority processes get whatever is left.**

In simple words: Priority frame allocation = “**Important processes get the memory they need first, and less important ones get what remains.**”

3. Global replacement allocation

When there is a page fault in the operating system, then the global replacement allocation takes care of it. The process with lower priority can give frames to the process with higher priority to avoid page faults.

4. Local replacement allocation: In local replacement allocation, the frames of pages can be stored on the same page. It doesn't influence the behaviour of the process as it did in **global replacement allocation**.

5. Equal frame allocation

In equal frame allocation, the processes are allocated equally among the processes in the operating system.

The only disadvantage in equal frame allocation is that a process requires more frames for allocation for execution and there are only a set number of frames.

Q) File System Interface & Implementation

The file system is the most prominent aspect of any operating system for most users. This provides users with methods for storage and access to data and programs of the operating system that all computer system users can use.

The file system consists of two distinct parts:

- A collection of files that store related data.
- A directory structure that organizes and provides information about all the files in the system.

File Attribute:

Files are named to make it easy for users to refer to them. A file name is typically a string of characters, such as "filename.cpp", which includes an extension that identifies the file format. Some systems, like Linux, are case-sensitive regarding file names, while others are not.

Once a file is named, it becomes independent of the user, process, or system that created it. For example, one user might create a file called "filename.cpp", while another might edit it using its name. The file's owner could also write it to a CD, send it via email, or copy it across a network, and it would still be known as "filename.cpp" on the destination system.

Fundamental components of a file:

A file's attributes vary from one operating system to another but typically consist of these:

- **Name:** Name is the symbolic file name and is the only information kept in human-readable form.
- **Identifier:** This unique tag is a number that identifies the file within the file system; it is in the non-human-readable form of the file.
- **Type:** This information is needed for systems that support different types of files or their formats.
- **Location:** This information is a pointer to a device pointing to the file's location on the device where it is stored.
- **Size:** The current file size (which is in bytes, words, etc.), possibly the maximum allowed size, gets included in this attribute.
- **Protection:** Access-control information establishes who can do the reading, writing, executing, etc.
- **Date, Time, and user identification:** This information might be kept for creating the file, its last modification, and its previous use. These data might be helpful in the field of protection, security, and monitoring its usage.

File Concepts in Operating Systems

A **file** in an operating system is a named collection of related information that is recorded on secondary storage such as a hard disk, SSD, CD, etc. Files represent the main method of storing data persistently.

OS file concepts can be categorized into several main topics:

1. File Attributes: Each file has metadata stored in the file system. Common file attributes include:

1. Name: The name of a file is the human-readable label used to identify it in a directory. It helps users recognize and select the right file easily. A file name usually has two parts: the main name and an extension that shows the file type, such as notes.txt or photo.jpg. Different operating systems have different rules for naming files—Windows restricts certain symbols, while Linux is more flexible. Even though users see and use file names, the operating system internally uses a separate unique number to track each file. File names must be unique inside a single folder, but the same name can be used in different folders without any problem.

2. Identifier: The identifier of a file is a unique number or code given by the operating system to every file. Unlike the file name, which users can change anytime, the identifier stays the same for the entire life of the file. The OS uses this identifier internally to quickly find and manage the file, even if other files have the same name in different folders. For example, Linux uses an **inode number(unique)**, and Windows uses a **file ID** as the identifier. This unique tag helps the system keep track of the file's location, size, and permissions.

3. Type: The *type* of a file indicates what kind of data it contains and how the operating system or applications should handle it. File types are usually identified by their extensions, such as .txt for text files, .jpg for images, .mp3 for audio files, or .exe for executable programs. Knowing the file type helps the OS choose the correct program to open or process the file. For example, a .pdf file opens in a PDF reader, while a .docx file opens in a word processor. Some operating systems rely heavily on file extensions to identify file types, while others, like Linux, also use internal file metadata to determine the type.

Eg: Text, Binary, Executable,, Image, Audio,, Video

4. Location: The *location* of a file refers to the physical address or position where the file is stored on the disk. It tells the operating system exactly **where to find the file's data blocks** so it can read or write information. The location is managed by the file system, which keeps track of all free and occupied blocks on the storage device. Even if a file is moved or renamed, the OS updates its location internally so that the file can still be accessed correctly. For example, in Linux, the inode contains pointers to the disk blocks where the file's data is stored, which represents the file's location. This attribute is crucial for efficient file retrieval and storage management.

5. Size: File size, measured in bytes, represents the actual amount of data contained in a file. However, the storage a file occupies on disk can differ from its logical size due to the way operating systems allocate space. Most file systems store data in fixed-size blocks or clusters, meaning even a small file may take up an entire block. For example, a 1,500-byte file on a file system with 4,096-byte blocks will consume 4,096 bytes of disk space.

6. Protection: File protection refers to the mechanisms that control who can access or modify a file. It ensures that sensitive data is only available to authorized users and prevents accidental or malicious changes. Protection information is stored by the operating system and is a key part of file metadata.

7. Time, Date, User Information: Files store metadata that tracks important timestamps and ownership information. This includes the time of creation, the last modification time, and sometimes the last access time. Alongside these, files also record user and group ownership, which helps the operating system enforce security and track accountability.

File Operations

1. Create(): The Create() system call is used to generate a new, empty file within the file system. When a file is created, the operating system allocates space for its metadata, including attributes such as size, timestamps, protection, and ownership. Some file systems also allow creating files with initial content or specific permissions set at creation time.

2. Open(): The Open() operation loads a file's metadata into memory and prepares it for access. This includes establishing a file descriptor or handle that the OS uses to track the file. Opening a file can specify modes such as read, write, or both, and can also include flags like "append" or "exclusive access." Proper error handling ensures that the OS prevents conflicts if multiple processes try to open the same file in incompatible modes.

3. Close(): Close() releases a file from active use by the program. This ensures that any data buffered in memory is written to disk and that system resources, such as file descriptors, are freed. Closing a file is important to prevent memory leaks, data corruption, or locking issues, especially in multi-user or multitasking environments.

4. Read(): The Read() operation transfers data from the file to a program's memory. It uses the file pointer to determine the position in the file from where to start reading. Many operating systems support buffered reads for efficiency, and reading beyond the end of the file typically returns an end-of-file indicator. Reading can be sequential or random, depending on whether the file pointer is moved manually.

5. Write(): Write() transfers data from memory to the file. If the file already contains data at the write location, it may be overwritten, or additional space may be allocated. Most operating systems use caching or buffering to optimize write operations, writing in

blocks rather than byte-by-byte. File permissions and protection attributes determine whether a process is allowed to write to a file.

6.Delete():The Delete() operation removes a file from the file system. In most systems, this involves de-allocating the file's metadata and freeing its disk space. Some operating systems move files to a temporary "recycle bin" or "trash" for recovery, while others permanently remove the file immediately. Deletion is subject to access permissions, ensuring only authorized users can remove files.

7.Seek():Seek() moves the file pointer to a specific location within a file. This allows programs to read from or write to arbitrary positions, rather than sequentially. Seek is essential for random access files, databases, or large multimedia files, where direct access to specific data blocks improves performance. The operation usually supports moving relative to the start, current position, or end of the file.

8. Append(): Append() adds new data to the end of an existing file without altering its current contents. Many logging or data collection programs use append mode to continuously add information. The OS ensures that the file pointer is automatically positioned at the end of the file during append operations, preventing accidental overwrites.

9. Rename(): The Rename() operation changes the name of an existing file without modifying its content. Some file systems also allow moving the file to a different directory as part of the rename. Proper permissions are required, and the operation typically updates metadata such as timestamps to reflect the change. Renaming is an efficient way to reorganize files without rewriting them.

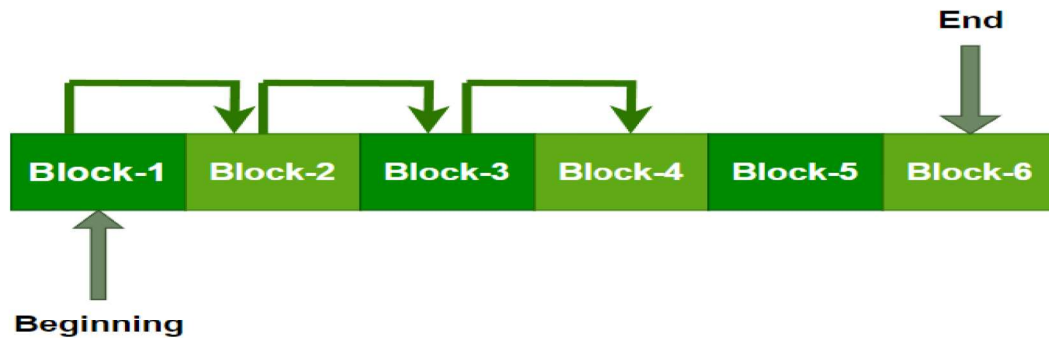
Access methods:

File access methods are techniques used by an OS to read and write data in files. They define how information is organized, retrieved, and modified. Choosing the right method is important for performance and data management.

There are three ways to access a file in a computer system:

- Sequential-Access
 - Direct Access
 - Index sequential Method
- **Sequential Access:** A file access method where data is read or written **in order, one record after another**, starting from the beginning. The file pointer moves forward automatically after each operation.

Sequential Access Method



Key Points related to Sequential Access

- Data is accessed from one record right after another record in an order.
- When we use the read command, it moves ahead pointer by one.
- When we use the write command, it will allocate memory and move the pointer to the end of the file.
- Such a method is reasonable for tape.

Advantages of Sequential Access Method

- It is simple to implement this file access mechanism.
- It uses lexicographic order to quickly access the next entry.
- It is less prone to data corruption as the data is written sequentially and not randomly.

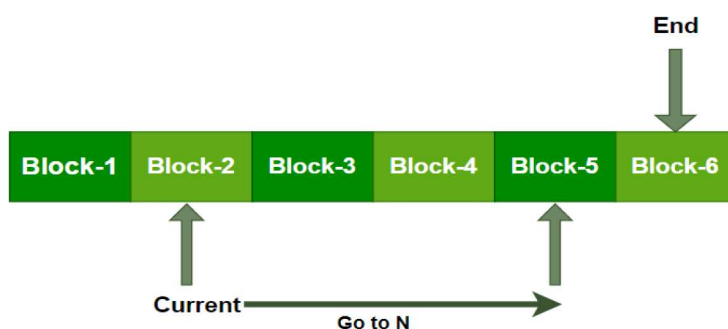
Disadvantages of Sequential Access Method

- Slow for searching specific records.
- Inserting/updating in the middle is difficult.
- Can waste storage if records have varying lengths.

➤ Direct Access Method

A file access method that allows data to be read or written directly at any block or record, using its address (block number). It supports random access without scanning previous records.\

Direct Access Method



Advantages of Direct Access Method

- The files can be immediately accessed decreasing the average access time.
- In the direct access method, in order to access a block, there is no need of traversing all the blocks present before it.

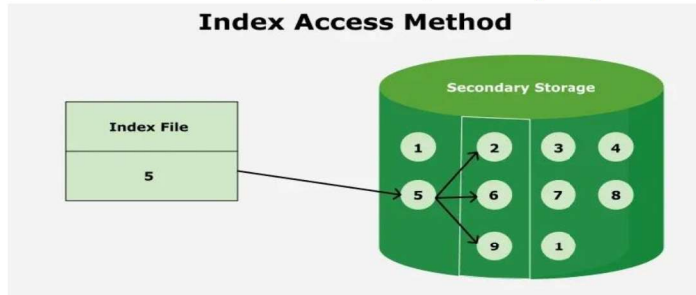
Disadvantages of Direct Access Method

- **Complex Implementation** : Implementing direct access can be complex, requiring sophisticated algorithms and data structures to manage and locate records efficiently.

- **Higher Storage Overhead** : Direct access methods often require additional storage for maintaining data location information (such as pointers or address tables), which can increase the overall storage requirements.

➤ **Index Sequential method**

It is the other method of accessing a file that is built on the top of the sequential access method. These methods construct an index for the file. The index, like an index in the back of a book, contains the pointer to the various blocks. To find a record in the file, we first search the index, and then by the help of pointer we access the file directly.



Key Points Related to Index Sequential Method

- It is built on top of Sequential access.
- It control the pointer by using index.

Advantages of Index Sequential Method

- **Fast Searching**: Index enables quick lookups.
- **Flexible**: Supports both sequential and random access.
- **Reduced Access Time**: Quickly locates data in large files.

Disadvantages of Index Sequential Method

Complexity: Harder to implement and maintain than sequential access.

Extra Storage: Requires additional space for indexes.

Slower Updates: Both data and index must be updated during insertions, deletions, or modifications.

Maintenance Overhead: Indexes need frequent updates in dynamic environments.

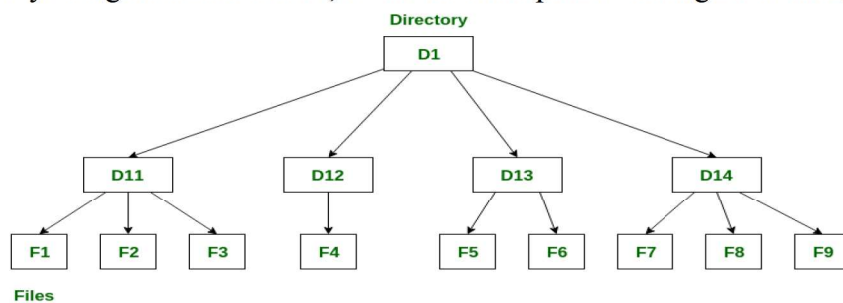
Structures of Directory in Operating System:

The operating system uses directories to track where files are stored, just like using folders to organize papers.

Different directory structures can be used to suit various organizational needs.

Understanding directory structures helps in organizing and accessing files more easily.

By using these structures, it becomes simpler to manage and navigate files on your computer.

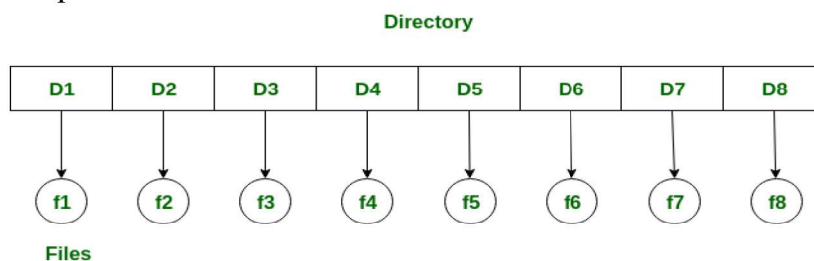


Different Types of Directories in OS: In an operating system, there are different types of directory structures that help organise and manage files efficiently. Each type of directory has its own way of arranging files and directories, offering unique benefits and features. These are

1. Single-Level Directory
2. Two-Level Directory
3. Tree Structure/ Hierarchical Structure
4. Acyclic Graph Structure
5. General-Graph Directory Structure

1) Single-Level Directory: The single-level directory is the simplest directory structure. In it, all files are contained in the same directory which makes it easy to support and understand.

A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user. Since all the files are in the same directory, they must have a unique name. If two users call their dataset test, then the unique name rule is violated.



Advantages

Logical Organization: Arrange files hierarchically for easy navigation.

Efficiency: Faster file searching and access.

Security: Restrict access at directory level to protect data.

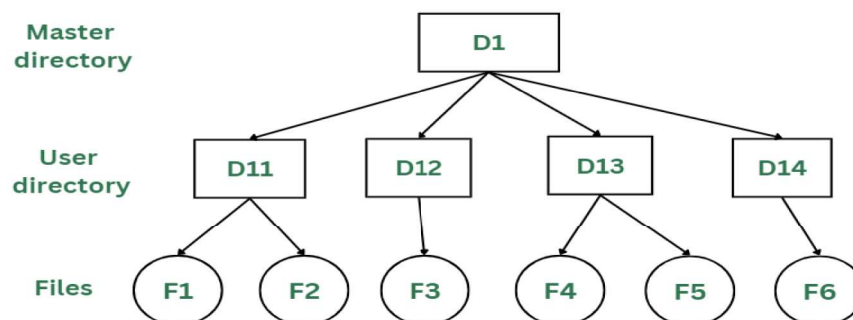
Backup & Recovery: Simplifies locating and restoring files.

Scalability: Easily supports growth with new files and directories

Disadvantages

- There may be chance of name collision because two files can have the same name.
- Searching will become time taking if the directory is large.
- This can not group the same type of files together.

2) Two-Level Directory: In a two-level directory structure, each user has a separate User File Directory (UFD) containing only their files. A Master File Directory (MFD) stores entries for all users and points to their respective UFDs, preventing filename conflicts between users.



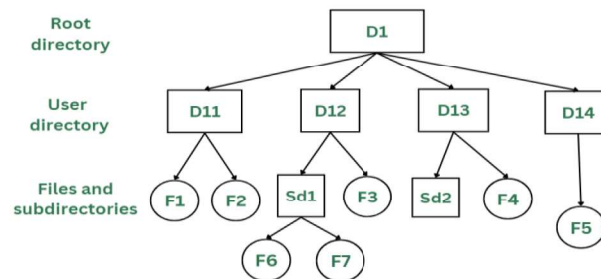
Advantages

- The main advantage is there can be more than two files with same name, and would be very helpful if there are multiple users.
- A security would be there which would prevent user to access other user's files.
- Searching of the files becomes very easy in this directory structure.

Disadvantages

- As there is advantage of security, there is also disadvantage that the user cannot share the file with the other users.
- Unlike the advantage users can create their own files, users don't have the ability to create subdirectories.
- Scalability is not possible because one user can't group the same types of files together.

3) Tree Structure/ Hierarchical Structure: The tree directory structure is the most common in personal computers. It resembles an upside-down tree, with the root directory at the top containing all user directories. Each user can create files and subdirectories within their own directory but cannot access or modify the root or other users' directories.



Advantages

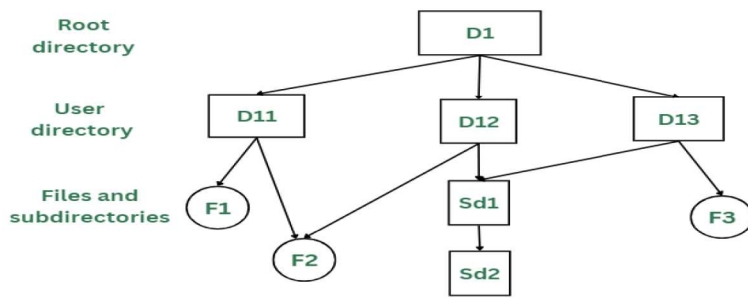
- This directory structure allows subdirectories inside a directory.
- The searching is easier.
- File sorting of important and unimportant becomes easier.
- This directory is more scalable than the other two directory structures explained.

Disadvantages

- As the user isn't allowed to access other user's directory, this prevents the file sharing among users.
- As the user has the capability to make subdirectories, if the number of subdirectories increase the searching may become complicated.
- Users cannot modify the root directory data.
- If files do not fit in one, they might have to be fit into other directories.

4) Acyclic Graph Structure: In the earlier directory structures, a file could only be accessed from the directory it was stored in. The acyclic graph directory structure solves this by allowing a file or subdirectory to be shared across multiple directories using links. Changes made by one user are visible to all users sharing that file.

In the below figure, this explanation can be nicely observed, where a file is shared between multiple users. If any user makes a change, it would be reflected to both the users.



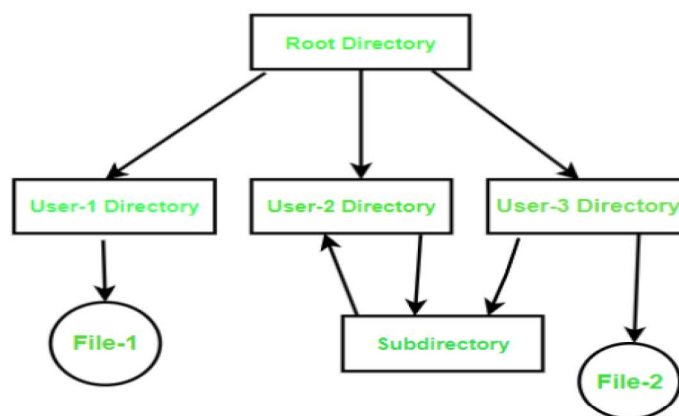
Advantages

- Sharing of files and directories is allowed between multiple users.
- Searching becomes too easy.
- Flexibility is increased as file sharing and editing access is there for multiple users.

Disadvantages

- Because of the complex structure it has, it is difficult to implement this directory structure.
- The user must be very cautious to edit or even deletion of file as the file is accessed by multiple users.
- If we need to delete the file, then we need to delete all the references of the file in order to delete it permanently.

5) General-Graph Directory Structure: Unlike the acyclic-graph directory, which avoids loops, the general-graph directory can have cycles, meaning a directory can contain paths that loop back to the starting point. This can make navigating and managing files more complex.



In the above image, you can see that a cycle is formed in the User 2 directory. While this structure offers more flexibility, it is also more complicated to implement.

Advantages of General-Graph Directory

- More flexible than other directory structures.
- Allows cycles, meaning directories can loop back to each other.

Disadvantages of General-Graph Directory

- More expensive to implement compared to other solutions.
- Requires garbage collection to manage and clean up unused files and directories.

File Sharing:

File Sharing in an Operating System(OS) denotes how information and files are shared between different users, computers, or devices on a network; and files are units of data that are stored in a computer in the form of documents/images/videos or any others types of information needed.

For Example: Suppose letting your computer talk to another computer and exchange pictures, documents, or any useful data. This is generally useful when one wants to work on a project with others, send files to friends, or simply shift stuff to another device. Our OS provides ways to do this like email attachments, cloud services, etc. to make the sharing process easier and more secure.

Now, file sharing is nothing like a magical bridge between Computer A to Computer B allowing them to swap some files with each other.

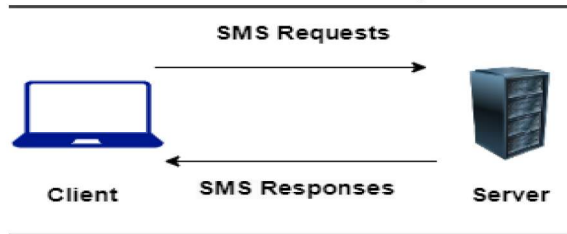
Primary Terminology Related to File Sharing: some important points are defined as follows.

- **Folder/Directory:** It is basically like a container for all of our files on a computer. The folder can contain files and even other folders maintaining like hierarchical structure for organizing data.
- **Networking:** It is involved in connecting computers or devices where we need to share the resources. Networks can be local (LAN) or global (Internet).
- **IP Address:** It is numerical data given to every connected device on the network
- **Protocol:** It is given as the set of rules which drives the communication between devices on a network. In the context of file sharing, protocols define how files are transferred between computers.
- **File Transfer Protocol (FTP):** FTP is a standard network protocol used to transfer files between a client and a server on a computer network.

Various Ways to Achieve File Sharing: various ways through which we can achieve file sharing in an OS.

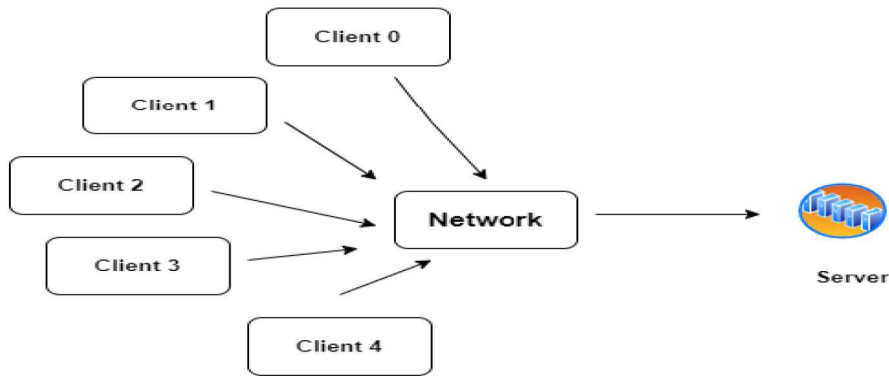
1. Server Message Block (SMB): SMB is like a network based file sharing protocol mainly used in windows operating systems. It allows our computer to share files/printer on a network. SMB is now the standard way for seamless file transfer method and printer sharing.

Example: Imagine in a company where the employees have to share the files on a particular project. Here SMB is employed to share files among all the windows based operating system. Orate on projects. SMB/CIFS is employed to share files between Windows-based computers. Users can access shared folders on a server, create, modify, and delete files.



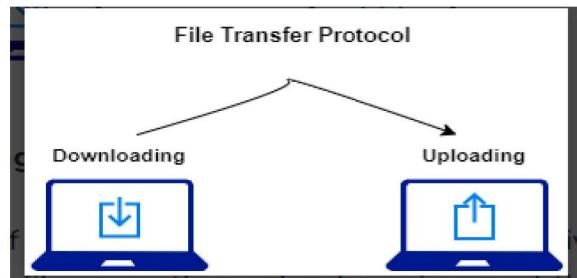
2. Network File System (NFS): NFS is a distributed based file sharing protocol mainly used in Linux/Unix based operating System. It allows a computer to share files over a network as if they were based on local. It provides a efficient way of transfer of files between servers and clients.

Example: Many Programmer/Universities/Research Institution uses Unix/Linux based Operating System. The Institutes puts up a global server datasets using NFS. The Researchers and students can access these shared directories and everyone can collaborate on it.



3. File Transfer Protocol (FTP): It is the most common standard protocol for transferring of the files between a client and a server on a computer network. FTPs supports both uploading and downloading of the files, here we can download, upload and transfer of files from Computer A to Computer B over the internet or between computer systems.

Example: Suppose the developer makes changes on the server. Using the FTP protocol, the developer connects to the server they can update the server with new website content and updates the existing file over there.



4. Cloud-Based File Sharing: It involves the famous ways of using online services like Google Drive, Drop Box, One Drive ,etc. Any user can store files over these cloud services and they can share that with others, and providing access from many users. It includes collaboration in real-time file sharing and version control access.

Ex: Several students working on a project and they can use Google Drive to store and share for that purpose. They can access the files from any computer or mobile devices and they can make changes in real-time and track the changes over there.



These all file sharing methods serves different purpose and needs according to the requirements and flexibility of the users based on the operating system.

Q)System Protection in Operating System

Introduction: System protection in an operating system refers to the mechanisms implemented by the operating system to ensure the security and integrity of the system. System protection involves various techniques to prevent unauthorized access, misuse, or modification of the operating system and its resources.

There are several ways in which an operating system can provide system protection:

User authentication: The operating system requires users to authenticate themselves before accessing the system. Usernames and passwords are commonly used for this purpose.

Access control: The operating system uses access control lists (ACLs) to determine which users or processes have permission to access specific resources or perform specific actions.

Encryption: The operating system can use encryption to protect sensitive data and prevent unauthorized access.

Firewall: A firewall is a software program that monitors and controls incoming and outgoing network traffic based on predefined security rules.

Antivirus software: Antivirus software is used to protect the system from viruses, malware, and other malicious software.

System updates and patches: The operating system must be kept up-to-date with the latest security patches and updates to prevent known vulnerabilities from being exploited.

By implementing these protection mechanisms, the operating system can prevent unauthorized access to the system, protect sensitive data, and ensure the overall security and integrity of the system.

What is Protection?

Protection refers to a mechanism which controls the access of programs, processes, or users to the resources defined by a computer system. We can take protection as a helper to multi programming operating system, so that many users might safely share a common logical name space such as directory or files.

Need for Protection:

- To prevent the access of unauthorized users
- To ensure that each active programs or processes in the system uses resources only as the stated policy
- To improve reliability by detecting latent errors

Role of Protection: The role of protection is to provide a mechanism that implement policies which defines the uses of resources in the computer system. Some policies are defined at the time of design of the system, some are designed by management of the system and some are defined by the users of the system to protect their own files and programs. Every application has different policies for use of the resources and they may change over time so protection of the system is not only concern of the designer of the operating system. Application programmer should also design the protection mechanism to protect their system against misuse. Policy is different from mechanism. Mechanisms determine how something will be done and policies determine what will be done. Policies are changed over time and place to place. Separation of mechanism and policy is important for the flexibility of the system.

Advantages of system protection in an operating system:

- Ensures the security and integrity of the system
- Prevents unauthorized access, misuse, or modification of the operating system and its resources
- Protects sensitive data
- Provides a secure environment for users and applications
- Prevents malware and other security threats from infecting the system
- Allows for safe sharing of resources and data among users and applications
- Helps maintain compliance with security regulations and standards

Disadvantages of system protection in an operating system:

- Can be complex and difficult to implement and manage
- May slow down system performance due to increased security measures
- Can cause compatibility issues with some applications or hardware
- Can create a false sense of security if users are not properly educated on safe computing practices
- Can create additional costs for implementing and maintaining security measures.

File System Implementation in Operating System

A file is a collection of related information. The file system resides on secondary storage and provides efficient and convenient access to the disk by allowing data to be stored, located, and retrieved. File system implementation in an operating system refers to how the file system manages the storage and retrieval of data on a physical storage device such as a hard drive, solid-state drive, or flash drive.

File system implementation is a critical aspect of an operating system as it directly impacts the performance, reliability, and security of the system. Different operating systems use different file system implementations based on the specific needs of the system and the intended use cases. Some common file systems used in operating systems include NTFS and FAT in Windows, and ext4 and XFS in Linux.

Components of File System Implementation: The file system implementation includes several components, including:

- **File System Structure:** The file system structure refers to how the files and directories are organized and stored on the physical storage device. This includes the layout of file systems data structures such as the directory structure, file allocation table, and inodes.
- **File Allocation:** The file allocation mechanism determines how files are allocated on the storage device. This can include allocation techniques such as contiguous allocation, linked allocation, indexed allocation, or a combination of these techniques.
- **Data Retrieval:** The file system implementation determines how the data is read from and written to the physical storage device. This includes strategies such as buffering and caching to optimize file I/O performance.
- **Security and Permissions:** The file system implementation includes features for managing file security and permissions. This includes access control lists (ACLs), file permissions, and ownership management.
- **Recovery and Fault Tolerance:** The file system implementation includes features for recovering from system failures and maintaining data integrity. This includes techniques such as journaling and file system snapshots.

Different Types of File Systems: There are several types of file systems, each designed for specific purposes and compatible with different operating systems. Some common file system types include:

- **FAT32 (File Allocation Table 32):** Commonly used in older versions of Windows and compatible with various operating systems.
- **NTFS (New Technology File System):** Used in modern Windows operating systems, offering improved performance, reliability, and security features.
- **ext4 (Fourth Extended File System):** Used in Linux distributions, providing features such as journaling, large file support, and extended file attributes.
- **HFS+ (Hierarchical File System Plus):** Used in macOS systems prior to macOS High Sierra, offering support for journaling and case-insensitive file names.
- **APFS (Apple File System):** Introduced in macOS High Sierra and the default file system for macOS and iOS devices, featuring enhanced performance, security, and snapshot capabilities.
- **ZFS (Zettabyte File System):** A high-performance file system known for its advanced features, including data integrity, volume management, and efficient snapshots.

Layers in File System: A file system in an operating system is organized into multiple layers, each responsible for different aspects of file management and storage. Here are the key layers in a typical file system:



- **Application Programs:** This is the topmost layer where users interact with files through applications. It provides the user interface for file operations like creating, deleting, reading, writing, and modifying files. Examples include text editors, file browsers, and command-line interfaces.
- **Logical File system** - It manages metadata information about a file i.e includes all details about a file except the actual contents of the file. It also maintains via file control blocks. File control block (FCB) has information about a file - owner, size, permissions, and location of file contents.
- **File Organization Module** - It has information about files, the location of files and their logical and physical blocks. Physical blocks do not match with logical numbers of logical blocks numbered from 0 to N. It also has a free space that tracks unallocated blocks.
- **Basic File system** - It Issues general commands to the device driver to read and write physical blocks on disk. It manages the memory buffers and caches. A block in the buffer can hold the contents of the disk block and the cache stores frequently used file system metadata.
- **I/O Control level** - Device drivers act as an interface between devices and OS, they help to transfer data between disk and main memory. It takes block number as input and as output, it gives low-level hardware-specific instruction.
- **Devices Layer:** The bottommost layer, consisting of the actual hardware devices. It performs the actual reading and writing of data to the physical storage medium. This includes hard drives, SSDs, optical disks, and other storage devices.

Implementation Issues:

- **Management of Disc space:** To prevent space wastage and to guarantee that files can always be stored in contiguous blocks, file systems must manage disc space effectively. Free space management, fragmentation prevention, and garbage collection are methods for managing disc space.
- **Checking for Consistency and Repairing Errors:** The consistency and error-free operation of files and directories must be guaranteed by file systems. Journaling, check summing, and redundancy are methods for consistency checking and error recovery. File systems may need to perform recovery operations if errors happen in order to restore lost or damaged data.
- **Locking Files and Managing Concurrency:** To prevent conflicts and guarantee data integrity, file systems must control how many processes or users can access a file at once. File locking, semaphore, and other concurrency-controlling methods are available.

- **Performance Optimization:** File systems need to optimize performance by reducing file access times, increasing throughput, and minimizing system overhead. Caching, buffering, prefetching, and parallel processing are methods for improving performance.

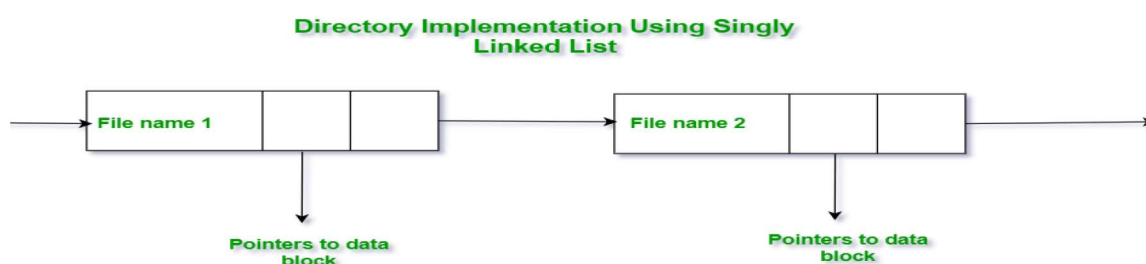
Directory Implementation in Operating System

Directory implementation in the operating system can be done using Singly Linked List and Hash table. The efficiency, reliability, and performance of a file system are greatly affected by the selection of directory-allocation and directory-management algorithms. There are numerous ways in which the directories can be implemented. But we need to choose an appropriate directory implementation algorithm that enhances the performance of the system.

The below are two ways of implementing the directory in the operating system :

- Directory Implementation using Singly Linked List
- Directory Implementation using Hash Table

Directory Implementation using Singly Linked List: The implementation of directories using a singly linked list is easy to program but is time-consuming to execute. Here we implement a directory by using a linear list of filenames with pointers to the data blocks.



Steps to Implement the Directory Using Singly Linked List: The steps are given below for the implementation of the directory:

- To create a new file the entire list has to be checked such that the new directory does not exist previously.
- The new directory then can be added to the end of the list or at the beginning of the list.
- In order to delete a file, we first search the directory with the name of the file to be deleted. After searching we can delete that file by releasing the space allocated to it.
- To reuse the directory entry we can mark that entry as unused or we can append it to the list of free directories.
- To delete a file linked list is the best choice as it takes less time.

Advantages

- **Simple Implementation:** Easy to implement with low memory overhead.
- **Dynamic Structure:** Grows or shrinks as needed without fixed size constraints.
- **Efficient for Small Directories:** Works well when the number of files is small and manageable.
- **Low Complexity:** No need for collision handling or resizing, making it simpler.

Disadvantages

- **Lookup Time** : File lookup requires a linear search, which can be time-consuming.
- **Impact of Frequent Access** : Directory information is accessed frequently, leading to slow access times with larger directories.
- **Solution- Caching** : Operating systems maintain a cache of recently accessed entries to enable quicker access without full traversal.

File Allocation Methods:

The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

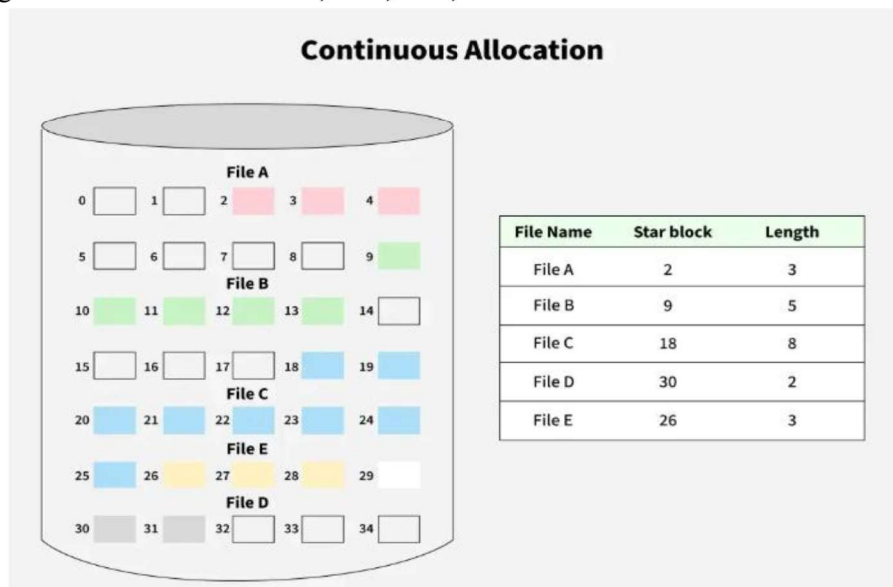
1. Contiguous Allocation
2. Linked Allocation
3. Indexed Allocation

The main idea behind these methods is to provide Efficient disk space utilization & Fast access to the file blocks.

1. **Contiguous Allocation:** In this scheme, each file occupies a contiguous set of blocks on the disk. This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file. The directory entry for a file with contiguous allocation contains:

- Address of starting block
- Length of the allocated portion.

Example: If a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: $b, b+1, b+2, \dots, b+n-1$.



Advantages

- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the k th block of the file which starts at block b can easily be obtained as $(b+k)$
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

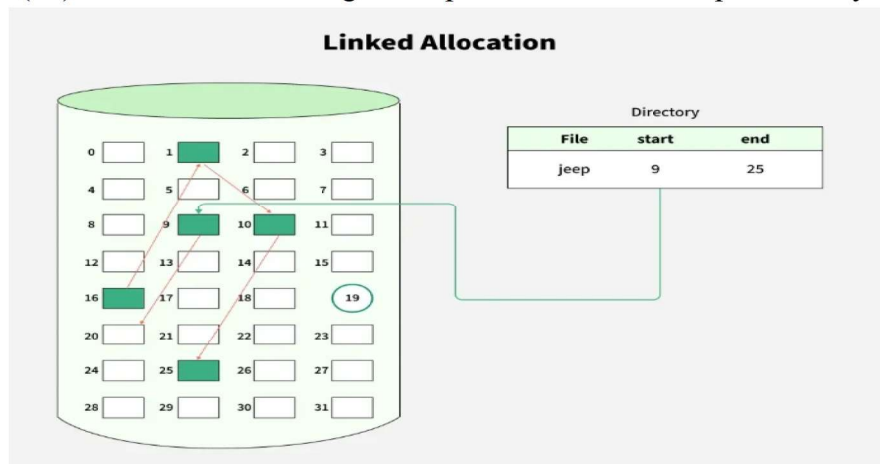
Disadvantages

- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

Linked Allocation: In this scheme, each file is a linked list of disk blocks which need not be contiguous. Here,

- The disk blocks can be scattered anywhere on the disk. The directory entry contains a pointer to the starting and the ending file block.
- Each block contains a pointer to the next block occupied by the file.

Example: The file 'jeep' in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.



Advantages

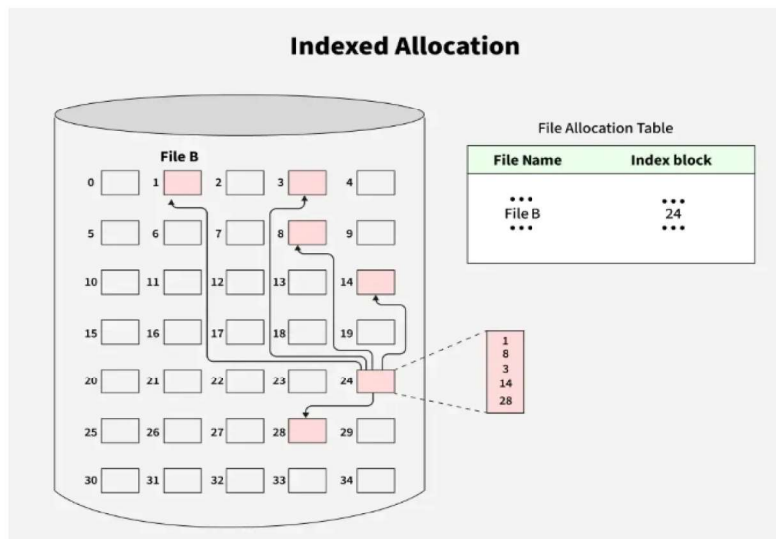
- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

Disadvantages

- Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
- It does not support random or direct access. We can not directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead.

Indexed Allocation: In this scheme, a special block known as the Index block contains the pointers to all the blocks occupied by a file. Here,

- Each file has its own index block.
- The i^{th} entry in the index block contains the disk address of the i^{th} file block.
- The directory entry contains the address of the index block as shown in the image.



Advantages

- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

Disadvantages

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

Indexing Schemes for Large Files: When a file is very large, a single index block may not have enough space to hold all the pointers to the data blocks. To manage this, the following indexing mechanisms are used:

1. Linked Scheme: In this approach, multiple index blocks are linked together.

Each index block contains some pointers to disk blocks and a pointer to the next index block.

The chain continues until all the pointers are covered.

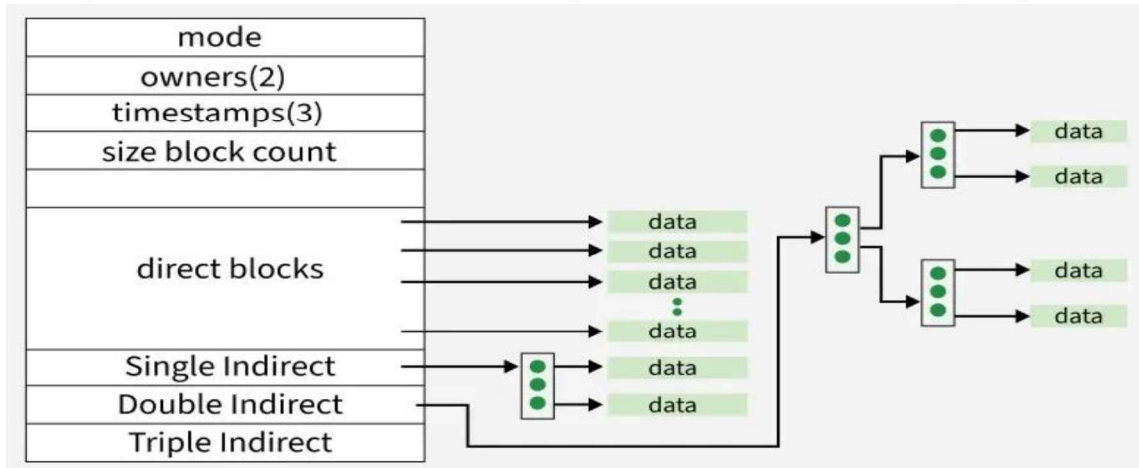
2. Multilevel Index: Instead of a single large index block, we use a hierarchy of index blocks.

The first-level index block points to second-level index blocks and those second-level blocks contain pointers to the actual data blocks. Can be extended to three or more levels as required by file size.

3. Combined Scheme (Inode Structure): A special block called the Inode (Information Node) stores File metadata (name, size, permissions, timestamps) & Block addresses (pointers).

The Inode typically contains:

1. **Direct Pointers:** Point directly to data blocks (e.g., first 10 pointers).
2. **Single Indirect Pointer:** Points to a block that holds further block addresses.
3. **Double Indirect Pointer:** Points to a block that contains pointers to other indirect blocks, which in turn point to data blocks.
4. **Triple Indirect Pointer:** Adds another layer of indirection for extremely large files.



UNIT-4

Secondary Storage Structure: Secondary storage refers to non-volatile memory that permanently stores data, programs, and files. Unlike primary memory (RAM), secondary storage retains data even when the power is off. The operating system (OS) manages this storage through various structures and techniques to ensure efficiency, reliability, and performance.

Basically, we want the programs and data to reside in main memory permanently. This arrangement is usually not possible for the following two reasons:

Main memory is usually too small to store all needed programs and data permanently.

Main memory is a volatile storage device that loses its contents when power is turned off or otherwise lost.

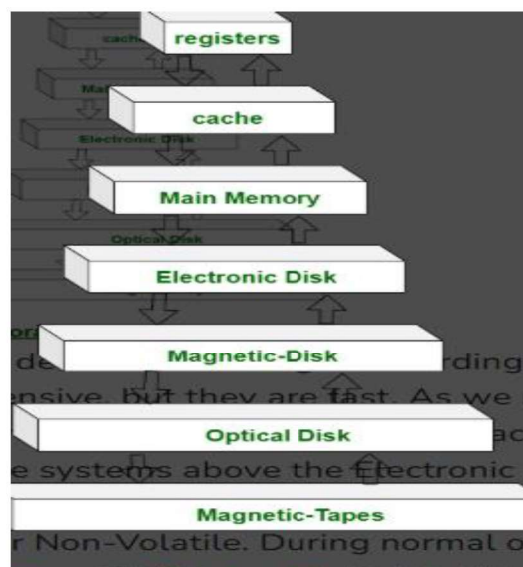
There are two types of storage devices

Volatile Storage Device - It loses its contents when the power of the device is removed.

Non-Volatile Storage device – It does not lose its contents when the power is removed. It holds all the data when the power is removed.

Secondary Storage is used as an extension of main memory. Secondary storage devices can hold the data permanently. Storage devices consist of Registers, Cache, Main-Memory, Electronic-Disk, Magnetic-Disk, Optical-Disk, Magnetic-Tapes. Each storage system provides the basic system of storing a datum and of holding the datum until it is retrieved at a later time. All the storage devices differ in speed, cost, size and volatility. The most common Secondary-storage device is a Magnetic-disk, which provides storage for both programs and data.

In this fig Hierarchy of storage is shown –



In this hierarchy all the storage devices are arranged according to speed and cost. The higher levels are expensive, but they are fast. As we move down the hierarchy, the cost per bit generally decreases, whereas the access time generally increases. The storage systems above the Electronic disk are Volatile, whereas those below are Non-Volatile. An Electronic disk can be

either designed to be either Volatile or Non-Volatile. During normal operation, the electronic disk stores data in a large DRAM array, which is Volatile. But many electronic disk devices contain a hidden magnetic hard disk and a battery for backup power. If external power is interrupted, the electronic disk controller copies the data from RAM to the magnetic disk. When external power is restored, the controller copies the data back into the RAM. The design of a complete memory system must balance all the factors. It must use only as much expensive memory as necessary while providing as much inexpensive, Non-Volatile memory as possible. Caches can be installed to improve performance where a large access-time or transfer-rate disparity exists between two components.

Disk Scheduling Algorithms:

Disk scheduling algorithms manage how data is read from and written to a computer's hard disk. These algorithms help determine the order in which disk read and write requests are processed.

- Disk scheduling is also known as I/O Scheduling.
- The main goals of disk scheduling are to optimize the performance of disk operations, reduce the time it takes to access data and improve overall system efficiency.
- Common disk scheduling methods include First-Come, First-Served (FCFS), Shortest Seek Time First (SSTF), SCAN, C-SCAN, LOOK, and C-LOOK.

Importance of Disk Scheduling in Operating System

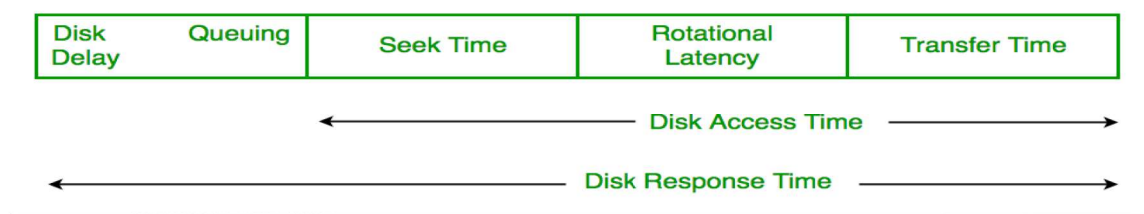
- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
- Two or more requests may be far from each other so this can result in greater disk arm movement.
- Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

Key Terms Associated with Disk Scheduling

- **Seek Time:** Time taken to move the disk arm to the track where data is located.
- **Rotational Latency:** Time taken for the desired sector to rotate under the read/write head.
- **Transfer Time:** Time taken to actually read or write the data, depending on disk speed and data size.

$\text{Disk Access Time} = \text{Seek Time} + \text{Rotational Latency} + \text{Transfer Time}$

$\text{Total Seek Time} = \text{Total head Movement} * \text{Seek Time}$



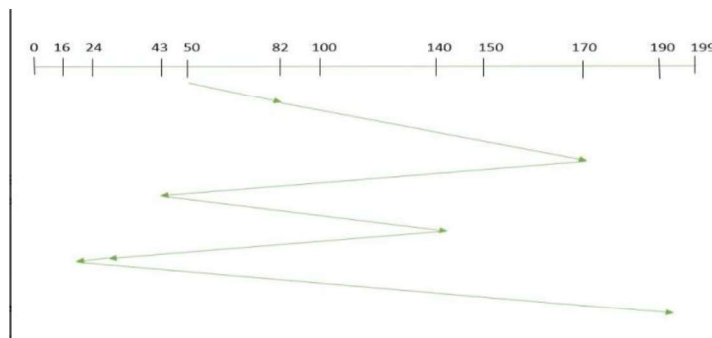
Goals of Disk Scheduling Algorithms

- Minimize Seek Time
- Maximize Throughput
- Minimize Latency
- Ensuring Fairness
- Efficiency in Resource Utilization

Disk Scheduling Algorithms: There are several Disk Scheduling Algorithms. We will discuss in detail each one of them.

1. FCFS (First Come First Serve)
2. SSTF (Shortest Seek Time First)
3. SCAN
4. C-SCAN
5. LOOK
6. C-LOOK
7. RSS (Random Scheduling)
8. LIFO (Last-In First-Out)
9. N-STEP SCAN
10. F-SCAN

1. FCFS (First Come First Serve): FCFS is the simplest of all Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue. Let us understand this with the help of an example.



Suppose the order of request is- (82,170,43,140,24,16,190) and current position of Read/Write head is: 50

So, total overhead movement (total distance covered by the disk arm) =
 $(82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16) = 642$

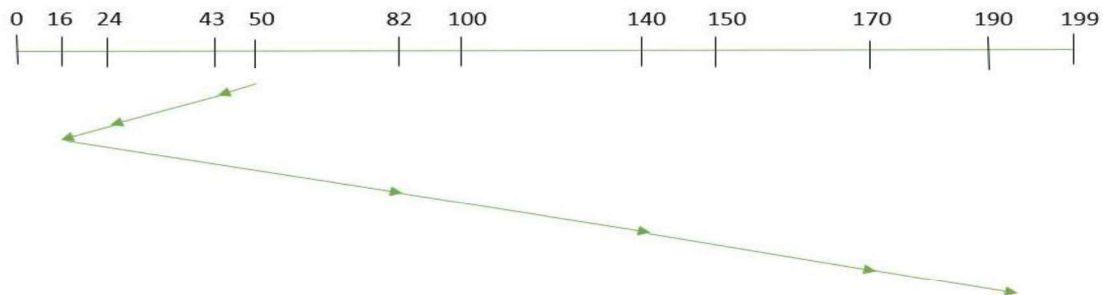
Advantages of FCFS

- Every request gets a fair chance
- No indefinite postponement

Disadvantages of FCFS

- Does not try to optimize seek time
- May not provide the best possible service

2. SSTF (Shortest Seek Time First): In SSTF (Shortest Seek Time First), requests having the shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of the system. Let us understand this with the help of an example.



Suppose the order of request is- (82,170,43,140,24,16,190) and current position of Read/Write head is: 50

total overhead movement (total distance covered by the disk arm) =
 $(50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-140)+(190-170) = 208$

Advantages of Shortest Seek Time First

Here are some of the advantages of Shortest Seek Time First.

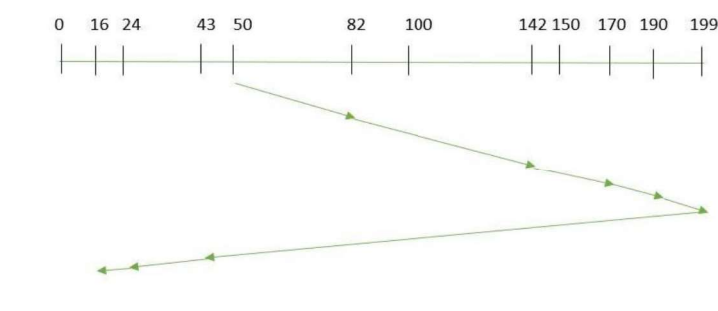
- The average Response Time decreases
- Throughput increases

Disadvantages of Shortest Seek Time First

Here are some of the disadvantages of Shortest Seek Time First.

- Overhead to calculate seek time in advance
- Can cause Starvation for a request if it has a higher seek time as compared to incoming requests
- The high variance of response time as SSTF favors only some requests

3. SCAN: In the SCAN algorithm the disk arm moves in a particular direction and services the requests coming in its path and after reaching the end of the disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and is hence also known as an elevator algorithm. As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.



Suppose the requests to be addressed are-82,170,43,140,24,16,190 and the Read/Write arm is at 50, and it is also given that the disk arm should move "towards the larger value".

Therefore, the total overhead movement (total distance covered by the disk arm) is calculated as $= (199-50) + (199-16) = 332$.

Advantages of SCAN Algorithm

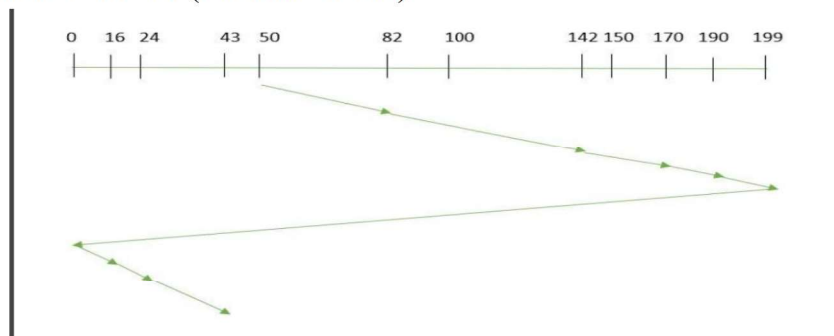
Here are some of the advantages of the SCAN Algorithm.

- High throughput
- Low variance of response time
- Average response time

Disadvantages of SCAN Algorithm: Long waiting time for requests for locations just visited by disk arm.

4. C-SCAN: In the SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.

These situations are avoided in the CSCAN algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to the SCAN algorithm hence it is known as C-SCAN (Circular SCAN).



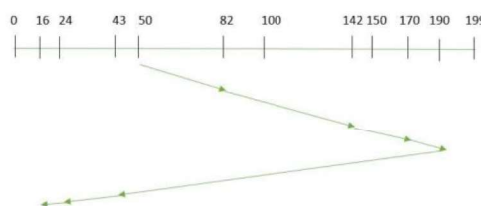
Suppose the requests to be addressed are- 82,170,43,140,24,16,190 and the Read/Write arm is at 50, and it is also given that the disk arm should move "towards the larger value".

So, the total overhead movement (total distance covered by the disk arm) is calculated as:

$$=(199-50) + (199-0) + (43-0) = 391.$$

Advantages of C-SCAN Algorithm: Provides more uniform wait time compared to SCAN.

5. LOOK: LOOK Algorithm is similar to the SCAN disk scheduling algorithm except for the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only. Thus, it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.



Suppose the requests to be addressed are- 82,170,43,140,24,16,190 and the Read/Write arm is at 50, and it is also given that the disk arm should move "towards the larger value".

So, the total overhead movement (total distance covered by the disk arm) is calculated as:

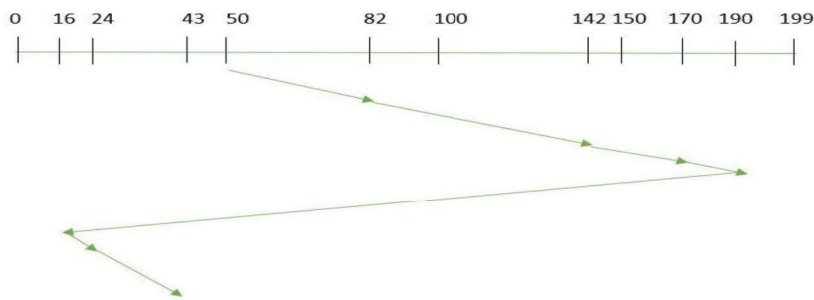
$$= (190-50) + (190-16) = 314.$$

Advantages

- **Reduced Unnecessary Movement:** The disk arm only goes as far as the last request in each direction, avoiding travel to the disk's physical end (unlike SCAN).
- **Faster Response:** Less head movement leads to quicker service for requests.

6. C-LOOK: As LOOK is similar to the SCAN algorithm, in a similar way, C-LOOK is similar to the CSCAN disk scheduling algorithm. In CLOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Example: Suppose the requests to be addressed are-82,170,43,140,24,16,190 and the Read/Write arm is at 50, and it is also given that the disk arm should move "towards the larger value".



So, the total overhead movement (total distance covered by the disk arm) is calculated as

$$= (190-50) + (190-16) + (43-16) = 341.$$

Advantages:

- **Uniform Wait Time:** Requests are serviced in a circular manner, so waiting times are more predictable and fair.
- **Reduced Head Movement:** The arm only goes as far as the last request in one direction, then jumps back, saving time compared to C-SCAN.

7. RSS (Random Scheduling): It stands for Random Scheduling and just like its name it is natural. It is used in situations where scheduling involves random attributes such as random processing time, random due dates, random weights, and stochastic machine breakdowns this algorithm fits perfectly. Which is why it is usually used for analysis and simulation.

8. LIFO (Last-In First-Out): In LIFO (Last In, First Out) algorithm, the newest jobs are serviced before the existing ones i.e. in order of requests that get serviced the job that is newest or last entered is serviced first, and then the rest in the same order.

Advantages of LIFO

- Maximizes locality and resource utilization

- Can seem a little unfair to other requests and if new requests keep coming in, it cause starvation to the old and existing ones.

9. N-STEP SCAN: It is also known as the N-STEP LOOK algorithm. In this, a buffer is created for N requests. All requests belonging to a buffer will be serviced in one go. Also once the buffer is full no new requests are kept in this buffer and are sent to another one. Now, when these N requests are serviced, the time comes for another top N request and this way all get requests to get a guaranteed service

Advantages of N-STEP SCAN: It eliminates the starvation of requests completely.

10. F-SCAN: This algorithm uses two sub-queues. During the scan, all requests in the first queue are serviced and the new incoming requests are added to the second queue. All new requests are kept on halt until the existing requests in the first queue are serviced.

Advantages of F-SCAN

- Prevents Arm Stickiness: The head doesn't get stuck near one area because requests are split into two queues.
- Fairness: All requests in the first queue are guaranteed service before moving to the second, avoiding indefinite delays.

Disk management:

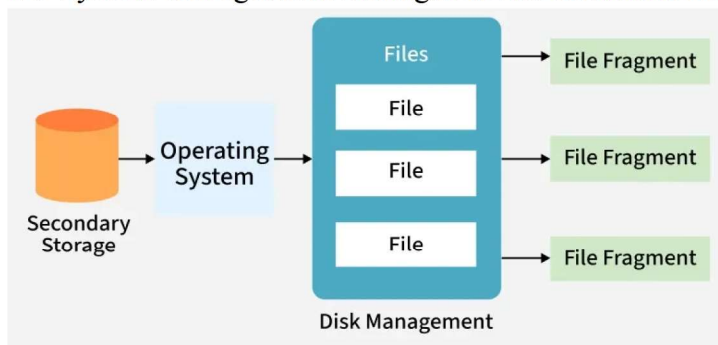
Disk management is a critical function of the operating system (OS) that deals with organizing, optimizing, and securing data on secondary storage devices such as hard disk drives (HDDs) and solid-state drives (SSDs).

It ensures:

- Efficient use of storage space
- Safe and reliable read/write operations
- Faster access times

OS Management Functions: Modern operating systems implement four major management functions, and disk management is one of them:

- Process Management: handles process execution.
- Memory Management: manages main memory allocation.
- File & Disk Management: organizes data on secondary storage.
- I/O System Management: manages device communication.



Why Disk Management is Important: Most computer systems use secondary storage devices (like hard disks, SSDs, tapes, optical media, and flash drives) to store programs and data at low cost and with non-volatile storage. Data is stored in the form of files.

The operating system (OS) manages file storage by allocating disk space as needed. Files are not always stored in one continuous block, large files may be fragmented into parts stored in different disk locations, especially when space is limited.

The OS keeps track of where each file (and its fragments) is located, often handling thousands of such entries. It ensures:

- Files can be quickly located for read/write operations
- Safe and reliable access to stored data
- Efficient management of access times

Key Operations in Disk Management

1. Disk Formatting

- **Low-level (physical) formatting:** Divides the disk into sectors with headers, data, and error correction codes (ECC).
- **Logical formatting:** Creates a file system, defining free space and allocated space.
- Blocks are grouped into clusters for efficient I/O.
- Some systems allow raw I/O (direct access to disk blocks without a file system).

2. Booting from Disk

- The bootstrap program loads the OS kernel into memory when the computer is powered on.
- A small bootstrap loader resides in ROM.
- The full bootstrap code is stored in the boot block of the disk.
- A disk with a boot partition is called a boot disk (system disk).

3. Bad Block Management: Disks often have bad sectors due to manufacturing defects or usage. Handled using:

- Sector sparing (replacement): faulty sectors are replaced with spare ones.
- Error recovery: for soft errors.
- Manual intervention: required for hard errors.
- Severe disk failures may require replacing the disk and restoring from backup.

Some common disk management techniques used in operating systems include:

- **Partitioning:** Divides a physical disk into multiple logical partitions, each acting as a separate storage device for better organization.
- **Formatting:** Prepares a disk by creating a file system; erases all existing data.
- **File System Management:** Manages file systems (e.g., FAT, NTFS, ext4) to store and access data efficiently.
- **Disk Space Allocation:** Allocates space for files using methods like contiguous, linked, or indexed allocation.
- **Disk Defragmentation:** Rearranges scattered data blocks to improve performance.

Swap-space Management: Swapping is a memory management technique used in multi-programming to increase the number of processes sharing the CPU. It is a technique of removing a process from the main memory and storing it into secondary memory, and then bringing it back into the main memory for continued execution. This action of moving a process out from main memory to secondary memory is called Swap Out and the action of moving a process out from secondary memory to main memory is called Swap In.

Swap-space management is a technique used by operating systems to optimize memory usage and improve system performance. Here are some advantages and disadvantages of swap-space management:

Advantages:

- **Increased memory capacity:** Swap-space management allows the operating system to use hard disk space as virtual memory, effectively increasing the available memory capacity.
- **Improved system performance:** By using virtual memory, the operating system can swap out less frequently used data from physical memory to disk, freeing up space for more frequently used data and improving system performance.
- **Flexibility:** Swap-space management allows the operating system to dynamically allocate and deallocate memory as needed, depending on the demands of running applications.

Swap-Space: The area on the disk where the swapped-out processes are stored is called swap space.

Swap-Space Management :

Swap-Space management is another low-level task of the operating system. Disk space is used as an extension of main memory by the virtual memory. As we know the fact that disk access is much slower than memory access, In the swap-space management we are using disk space, so it will significantly decreases system performance. Basically, in all our systems we require the best throughput, so the goal of this swap-space implementation is to provide the virtual memory the best throughput. In these article, we are going to discuss how swap space is used, where swap space is located on disk, and how swap space is managed.

Swap-Space Use :

Swap-space is used by the different operating-system in various ways. The systems which are implementing swapping may use swap space to hold the entire process which may include image, code and data segments. Paging systems may simply store pages that have been pushed out of the main memory. The need of swap space on a system can vary from a megabytes to gigabytes but it also depends on the amount of physical memory, the virtual memory it is backing and the way in which it is using the virtual memory.

It is safer to overestimate than to underestimate the amount of swap space required, because if a system runs out of swap space it may be forced to abort the processes or may crash entirely. Overestimation wastes disk space that could otherwise be used for files, but it does not harm other.

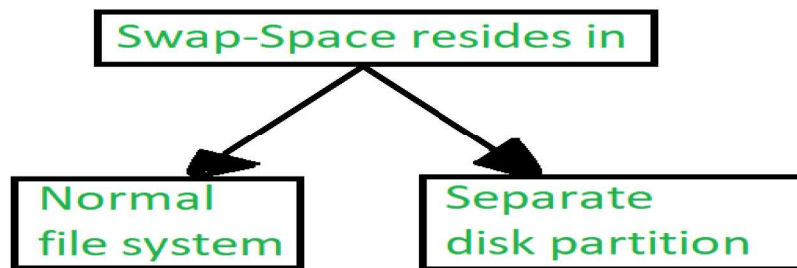
Following table shows different system using amount of swap space:

System	Swap-Space
1. Solaris	Equal amount of physical memory
2. Linux	Double the amount of physical memory

Explanation of above table : Solaris, setting swap space equal to the amount by which virtual memory exceeds page-able physical memory. In the past Linux has suggested setting swap space to double the amount of physical memory. Today, this limitation is gone, and most Linux systems use considerably less swap space.

Including Linux, some operating systems; allow the use of multiple swap spaces, including both files and dedicated swap partitions. The swap spaces are placed on the disk so the load which is on the I/O by the paging and swapping will spread over the system's bandwidth.

Swap-Space Location :



A swap space can reside in one of the two places -

1. Normal file system
2. Separate disk partition

Let, if the swap-space is simply a large file within the file system. To create it, name it and allocate its space normal file-system routines can be used. This approach, through easy to implement, is inefficient. Navigating the directory structures and the disk-allocation data structures takes time and extra disk access. During reading or writing of a process image, external fragmentation can greatly increase swapping times by forcing multiple seeks.

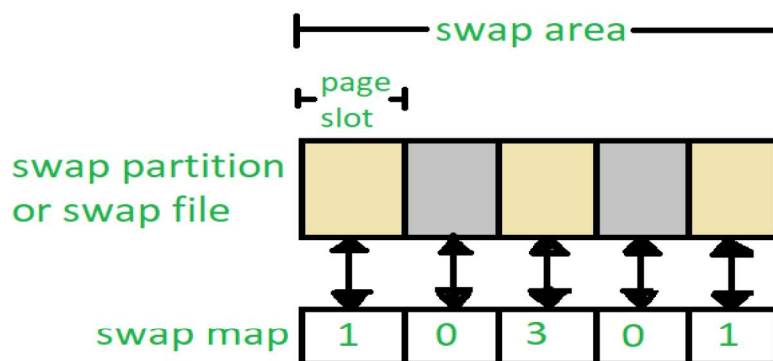
There is also an alternate to create the swap space which is in a separate raw partition. There is no presence of any file system in this place. Rather, a swap space storage manager is used to allocate and de-allocate the blocks. from the raw partition. It uses the algorithms for speed rather than storage efficiency, because we know the access time of swap space is shorter than the file system. By this Internal fragmentation increases, but it is acceptable, because the life span of the swap space is shorter than the files in the file system. Raw partition approach creates fixed amount of swap space in case of the disk partitioning.

Some operating systems are flexible and can swap both in raw partitions and in the file system space, example: Linux.

Swap-Space Management: An Example -

The traditional UNIX kernel started with an implementation of swapping that copied entire process between contiguous disk regions and memory. UNIX later evolved to a combination of swapping and paging as paging hardware became available. In Solaris, the designers changed standard UNIX methods to improve efficiency. More changes were made in later versions of Solaris, to improve the efficiency.

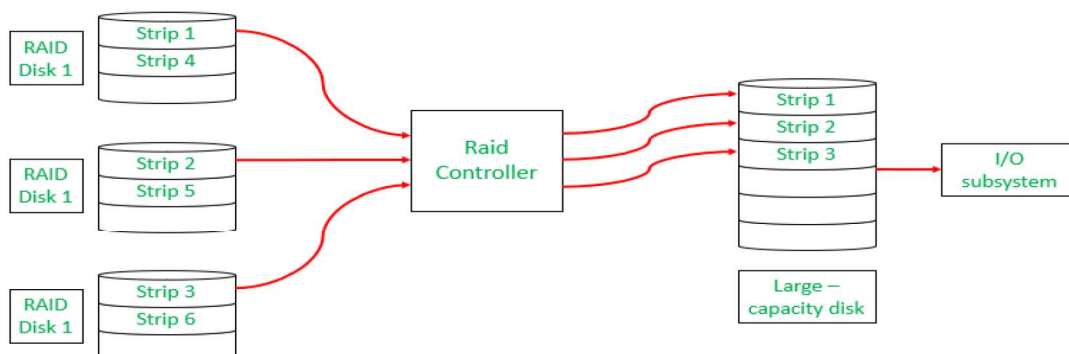
Linux is almost similar to Solaris system. In both the systems the swap space is used only for anonymous memory, it is that kind of memory which is not backed by any file. In the Linux system, one or more swap areas are allowed to be established. A swap area may be in either in a swap file on a regular file system or a dedicated file partition.



Each swap area consists of 4-KB page slots, which are used to hold the swapped pages. Associated with each swap area is a swap-map- an array of integers counters, each corresponding to a page slot in the swap area. If the value of the counter is 0 it means page slot is occupied by swapped page. The value of counter indicates the number of mappings to the swapped page. For example, a value 3 indicates that the swapped page is mapped to the 3 different processes.

RAID Structure.

RAID (Redundant Array of Independent Disks) structures are storage technologies that combine multiple disks into a single logical unit to improve performance, fault tolerance, or both. To the operating system, the array appears as one disk, but it is managed by a RAID controller that uses techniques like striping (data division) or mirroring (data duplication) to enhance data storage. Common RAID levels include RAID 0 (no redundancy), RAID 1 (mirroring), and RAID 5 (striping with distributed parity).



Core concepts

- **Striping (RAID 0):** Data is split into blocks and distributed across multiple drives. This improves performance because multiple drives can be accessed simultaneously, but offers no redundancy—if one drive fails, all data is lost.
- **Mirroring (RAID 1):** Data is written identically to two or more drives, creating a mirror set. This provides excellent fault tolerance; if one drive fails, the data is still available on the other(s). Read performance is improved, but write performance is similar to a single disk, and usable storage is halved.
- **Parity (RAID 5, 6, etc.):** Stores redundant data in the form of parity information across the drives, which can be used to reconstruct data if a drive fails. RAID 5 uses distributed parity, while RAID 6 uses double distributed parity to tolerate two drive failures.
- **RAID Controller:** A hardware or software component that manages the disks in the array, presenting them to the OS as a single logical volume. This abstraction layer handles the striping, mirroring, or parity calculations.

Common RAID levels**RAID 0 (Striping):**

- How it works: Splits data across multiple disks for maximum speed.
- Pros: High read/write performance.
- Cons: No fault tolerance; a single drive failure results in total data loss.

RAID 1 (Mirroring):

- How it works: Duplicates data onto two or more disks.
- Pros: Excellent data redundancy; can survive a single drive failure.
- Cons: 50% storage efficiency; can be expensive due to the need for double the drive capacity.

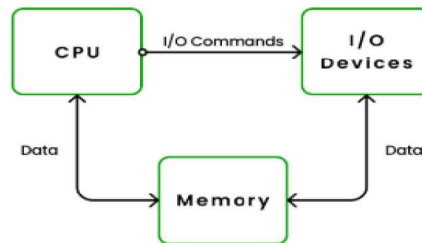
RAID 5 (Striping with distributed parity):

- How it works: Stripes data across drives and distributes parity information across the array.
- Pros: Good balance of performance and fault tolerance, with 100% capacity utilization from the user's perspective.
- Cons: Write performance is slower due to parity calculations; can only tolerate a single drive failure.

RAID 6 (Striping with double distributed parity):

- How it works: Similar to RAID 5, but with two independent parity blocks for increased redundancy.
- Pros: Can tolerate the failure of two drives simultaneously.
- Cons: Higher write penalty than RAID 5; requires more drives than RAID 5.

Introduction about I/O Systems: An operating system's I/O system manages communication between the computer and external devices like keyboards, hard drives, and printers. It acts as a mediator, translating high-level software requests into low-level hardware commands and handling tasks like buffering, caching, and scheduling to ensure efficient and reliable data transfer. This abstraction hides the complexity of hardware from applications, making it easier for them to interact with various devices.



I/O Hardware: I/O Hardware is a set of specialized hardware devices that help the operating system access disk drives, printers, and other peripherals. These devices are located inside the motherboard and connected to the processor using a bus. They often have specialized controllers that allow them to quickly respond to requests from software running on top of them or even respond directly to commands from an application program. This post will discuss in detail I/O Hardware basics such as daisy chain expansion bus controller memory-mapped I/O Direct Memory Access (DMA)

The Daisy chain, expansion bus, controller, and host adapter are used to access the I/O hardware.

The daisy chain is a method of connecting multiple I/O devices with each other through a single connection point (pin). Each device can be accessed by plugging into any of the pins on this connection point. The expansion bus connects devices together in parallel with each other so that they can be accessed simultaneously by using only one cable instead of several cables (one per device). This design allows you to connect more than one peripheral device while maintaining compatibility with older systems that may not support additional peripherals or features such as memory-mapped I/O (MMIO).

A controller manages all incoming data from its associated port and sends outgoing commands from its associated port; it's like an interface between an application software program and hardware components such as disk drives or network adapters

A host adapter is a bridge between the system bus and the expansion bus. It allows you to connect multiple expansion cards at the same time and provides additional interfaces for those cards such as DMA or MMIO. A controller manages all incoming data from its associated port and sends outgoing commands from its associated port; it's like an interface between an application software program and hardware components such as disk drives or network adapters.

1. Polling

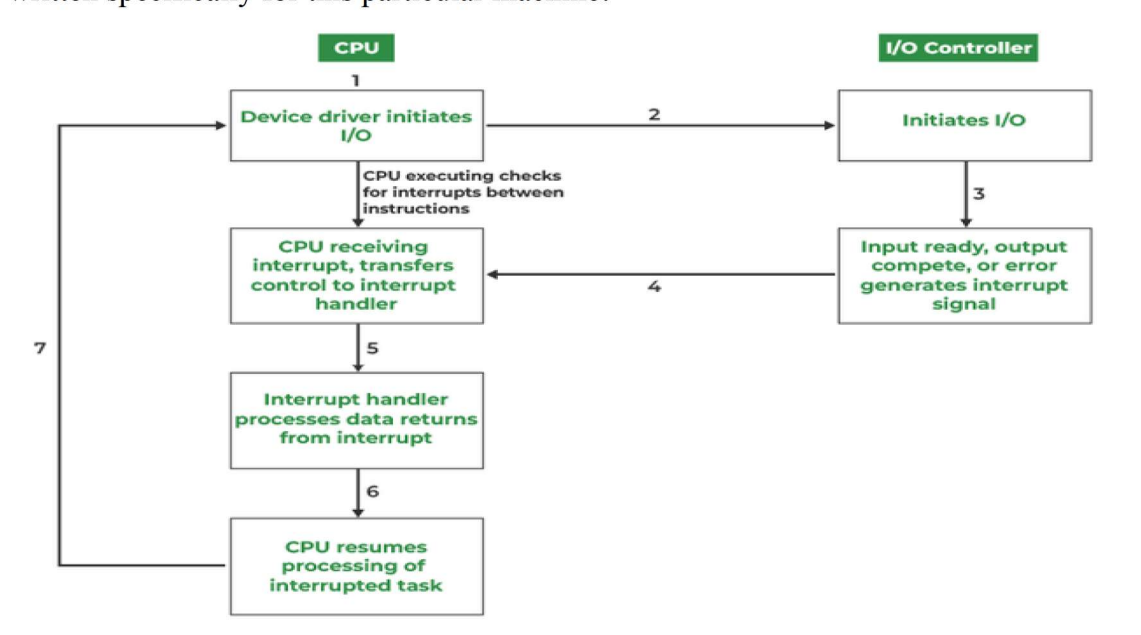
Polling is a software technique that uses a program to check the status of devices. The device can be a disk drive or any other peripheral device in the computer. The program polls the device for information, such as if it has data available or not. Polling is a slow way to get data from a device because it has to wait until another function occurs before being able to get information about its state.

In some cases polling may be desirable; for example, when there are several items being polled simultaneously and only one item updates its state at any time — the rest continue waiting until they receive an acknowledgment from one item that it's done updating their states (which could take seconds).

Polling can also be used to check if a device is online or not. If the device is offline, then this information can be used to take some appropriate action such as pausing or suspending tasks that depend on the device (e.g., stopping a backup in progress).

2. Interrupts

The CPU is interrupted by several different devices, such as I/O hardware and peripheral devices. These interrupts are used by the I/O device to notify the CPU when it needs attention. The CPU can be interrupted by several different devices at any given time, but only one interrupt will be delivered to it at any given time. This can happen because of a hardware or software error occurring on an I/O bus—for example, if a disk drive has failed then all other data transfers will pause until it's repaired; or perhaps another device wants access to memory (or vice versa). In either case, there won't be any way for your application program not to be written specifically for this particular machine!



The operating system can also be interrupted by other processes that need CPU time. For example, if a user application program writes to a file and then exits, the operating system will

write that file on disk before returning control back to the user. This is called multitasking and allows many different programs to run at once.

The operating system is responsible for scheduling the CPU's time among multiple programs. This is done by using a program called an interrupt handler or interrupt service routine (ISR). An ISR is simply a piece of code that runs when an interrupt occurs and typically ends with returning control back to the original program.

3. Direct Memory Access:

DMA is a way to move data between the CPU and I/O devices. When a user wants to access memory from an I/O device (such as a disk drive), the user must first inform the operating system that they are going to perform the operation. This is done by attaching a special command called an interrupt request (IRQ) handler routine for each device that needs access. The interrupt handler routine then tells the user program when it should be ready for another task—usually using interrupts generated by hardware components like joysticks or network cards—and may even tell other processes running on other CPUs not only about what's happening at that moment but also when something has happened before so they can take appropriate action as well!

The first step in using interrupts is to set up the interrupt controller. This is done by calling the BIOS INT 10h service with several parameters, including the number of IRQ lines available (16 is a common value), the maximum number of devices that can be connected to each line, and so on.

The next step is to assign an INT 10h handler for each device that needs access. This is done by calling the BIOS INT 15h service with several parameters, including the number of IRQ lines available (16 is a common value), the maximum number of devices that can be connected to each line, and so on.

I/O Hardware Summary: I/O hardware is a collection of devices that are used to make it easier for the CPU and other processors on your computer to communicate with the outside world. These devices may include:

Peripheral controller cards (also called I/O adapters) connect to the expansion bus (sometimes called PCI or ISA), and control how data is transferred between different parts of a computer system.

Interrupt request lines (IRQs) and serial ports, which allow multiple peripherals such as keyboards, mice, and printers to communicate with one another without having their own dedicated channel in memory.

A device driver software program that tells Windows what kind of hardware the user has installed so it can work properly; this may include information about how much RAM or hard drive space each device has available at any given moment in time because those things change over time depending on how much activity occurs within them throughout their life cycle.

Applications of Input/Output Interface:

I/O Interface:

There is need of interface whenever any CPU wants to communicate with I/O devices. The interface is used to interpret address which is generated by CPU. Thus, surface is used to communicate to I/O devices i.e. to share information between CPU and I/O devices interface is used which is called as I/O Interface.

Various applications of I/O Interface:

Application of I/O is that we can say interface have access to open any file without any kind of information about file i.e., even basic information of file is unknown. It also has feature that it can be used to also add new devices to computer system even it does not cause any kind of interrupt to operating system. It can also used to abstract differences in I/O devices by identifying general kinds. The access to each of general kind is through standardized set of function which is called as interface.

Each type of operating system has its own category for interface of device-drivers. The device which is given may ship with multiple device-drivers-for instance, drivers for Windows, Linux, AIX and Mac OS, devices may is varied by dimensions which is as illustrated in the following table :

S.No.	Basis	Alteration	Example
1.	Mode of Data-transfer	character or block	terminal disk
2.	Method of Accessing data	sequential or random	modem, CD-ROM
3.	Transfer schedule	synchronous or asynchronous	tape, keyboard
4.	Sharing methods	dedicated or sharable	tape, keyboard
5.	Speed of device	latency, seek time, transfer rate, delay between operations	
6.	I/O Interface	read only, write only, read-write	CD-ROM graphics controller disk

1. Character-stream or Block:

A character stream or block both transfers data in form of bytes. The difference between both of them is that character-stream transfers bytes in linear way i.e., one after another whereas block transfers whole byte in single unit.

2.Sequential or Random Access:

To transfer data in fixed order determined by device, we use sequential device whereas user to instruct device to seek to any of data storage locations, random-access device is used.

3.Synchronous or Asynchronous:

Data transfers with predictable response times is performed by synchronous device, in coordination with others aspects of system. An irregular or unpredictable response times not coordinated with other computer events is exhibits by an asynchronous device.

4.Sharable or Dedicated:

Several processes or threads can be used concurrently by sharable device; whereas dedicated device cannot.

5.Speed of Operation:

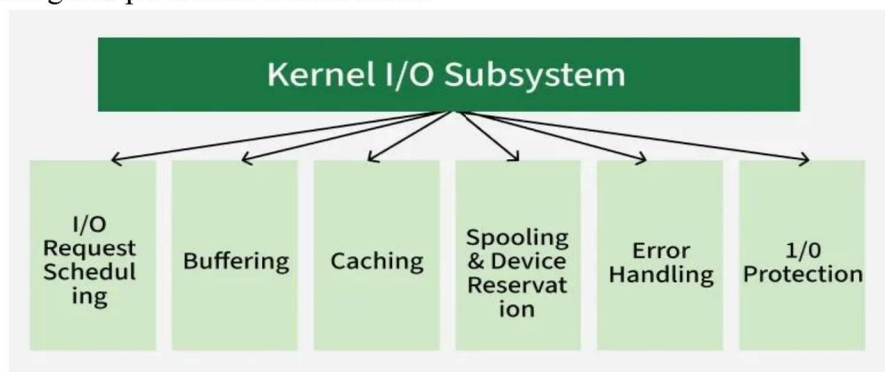
The speed of device has range set which is of few bytes per second to few giga-bytes per second.

6.Read-write, read only, write-only:

Different devices perform different operations, some supports both input and output, but others supports only one data transfer direction either input or output.

Kernel I/O subsystem:

Kernel I/O Subsystem is a key component of the operating system kernel responsible for managing communication between the CPU and I/O devices (e.g., printers, disks, keyboards, network interfaces). It translates high-level I/O requests from applications into low-level hardware commands, handling concurrency, synchronization, error handling, buffering, caching, spooling and protection mechanisms.



Services Provided by the Kernel I/O Subsystem

1. I/O Scheduling

- Determines the optimal order of executing I/O requests.
- Improves system performance by Reducing average waiting time and turnaround time & Sharing device access fairly across multiple processes.
- Implemented via wait queues for each device.
- Example: When multiple processes issue blocking I/O system calls, their requests are queued and reordered by the I/O scheduler for efficiency.

2. Buffering

A memory area temporarily stores data during transfers between devices or between a device and an application.

Main reasons for buffering:

- To manage speed mismatch between producer and consumer.
- To adapt different data transfer sizes.
- To support copy semantics (e.g., copying application data into a buffer before writing to disk).

3. Caching

- A cache stores frequently accessed data in fast memory.
- Example: Frequently used instructions are stored in CPU cache and main memory to avoid slow disk access.
- Difference from buffering: Buffer holds data temporarily during transfer & Cache holds a copy of data to speed up future access.

4. Spooling and Device Reservation

- Spooling stores output (e.g., print jobs) in a separate disk file to prevent interleaved outputs.
- Jobs are queued (FIFO) and executed one by one.
- Prevents output mixing and allows concurrent job submission.

5. Error Handling: Provides robust strategies for handling transient or permanent I/O errors:

- Error Detection Mechanisms: Detect I/O errors.
- Error Reporting: Notify OS and applications of detected errors.
- Error Recovery Mechanisms: Attempt to recover without system failure.
- User Notification: Inform users or administrators via alerts or automated messages.

6. I/O Protection:

- Prevents unauthorized or illegal I/O instructions.
- All I/O instructions are privileged.
- Only the Kernel can issue I/O instructions, protecting the system from malicious or erroneous user processes.

Functions and Services offered by the Kernel: The kernel provides many services related to I/O, some of these are:

1. Process management: Save context of the interrupted program, dispatch a process, manipulate scheduling lists.
2. Process communication: Send and receive inter-process messages.
3. Memory management: Set memory protection information, swap-in/ swap-out, handle page fault.
4. I/O management: Initiate I/O ,process I/O completion interrupt, recover from I/O errors.
5. File management: Open a file, read/ write data.
6. Security and protection: Add authentication information for a new user, maintain information for file protection.
7. Network management: Send/ receive data through a message.

Importance of Kernel I/O Subsystem

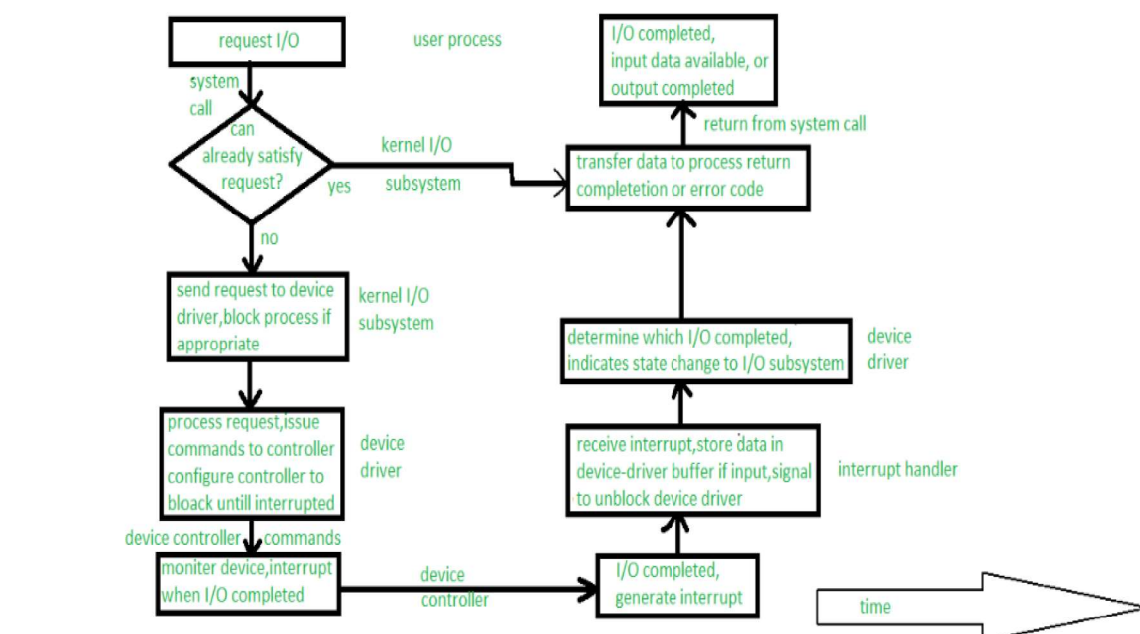
- Provides a unified and consistent interface for applications to access devices.
- Abstracts hardware complexity, enabling device independence.
- Manages concurrency and synchronization when multiple processes access the same device.
- Ensures data integrity and protects the system against unauthorized I/O access.

Transforming I/O to Hardware operations.

When an application requests an I/O operation such as reading a file, it gives the operating system a **file name**, not hardware details. The OS then uses its **file system structures** (like directories, FAT entries, or inodes) to convert the file name into information about where the file is stored physically on the disk. This step tells the OS which blocks or sectors need to be read.

Next, the OS determines **which hardware device** contains those blocks. In systems like UNIX/Linux, this is done using a **mount table**, which connects path prefixes (such as /home) to actual devices (like /dev/sda1). Because of this mapping, paths do not need to show device names directly. The OS then creates a generic I/O request describing which device and which blocks must be accessed.

Finally, this request is sent to the appropriate **device driver**, which knows how to communicate with the hardware. The driver translates the OS request into hardware-specific commands and sends them to the disk controller. The controller performs the actual read operation and returns the data through the driver back to the OS, which then delivers it to the application. This layered design allows new devices and drivers to be added without changing the core OS.



1. System call - Whenever, any I/O request comes, process issues blocking read() system call to previously opened file descriptor of file. Basically, role of system-call code is to check parameters for correctness in kernel. If data we put in form of input is already available in buffer cache, data is going to returned to process, and in that case I/O request is completed.

2.Alternative approach if input is not available - If the data is not available in buffer cache then physical I/O must be performed. The process is removes from run queue and is placed on wait queue for device, and I/O request is scheduled. After scheduling, I/O subsystem sends request to device driver via subroutine call or in-kernel message but it depends upon operating system by which mode request will send.

3.Role of Device driver - After receiving the request, device driver have to receive data and it will receive data by allocating kernel buffer space and after receiving data it will schedules I/O. After all this, command will be given to device controller by writing into device-control registers.

4.Role of Device Controller - Now, device controller operates device hardware. Actually, data transfer is done by device hardware.

5.Role of DMA controller - After data transfer, driver may poll for status and data, or it may have set up DMA transfer into kernel memory. The transfer is managed by DMA controller. At last when transfers complete, it will generates interrupt.

6.Role of interrupt handler - The interrupt is send to correct interrupt handler through interrupt-vector table. It store any necessary data, signals device driver, and returns from interrupt.

7.Completion of I/O request - When, device driver receives signal. This signal determines that I/O request has completed and also determines request's status, signals kernel I/O subsystem that request has been completed. After transferring data or return codes to address space kernel moves process from wait queue back to ready queue.

8.Completion of System call - When process moves to ready queue it means process is unblocked. When the process is assigned to CPU, it means process resumes execution at completion of system call.

UNIT-5

System Security:

The security of a computer system is a crucial task. It is a process of ensuring the confidentiality and integrity of the OS. Security is one of most important as well as the major task in order to keep all the threats or other malicious tasks or attacks or program away from the computer's software system.

A system is said to be secure if its resources are used and accessed as intended under all the circumstances, but no system can guarantee absolute security from several of various malicious threats and unauthorized access.

The security of a system can be threatened via two violations:

Threat: A program that has the potential to cause serious damage to the system.

Attack: An attempt to break security and make unauthorized use of an asset.

Security violations affecting the system can be categorized as malicious and accidental threats. Malicious threats, as the name suggests are a kind of harmful computer code or web script designed to create system vulnerabilities leading to back doors and security breaches. Accidental Threats, on the other hand, are comparatively easier to be protected against. Example: Denial of Service DDoS attack.

Security can be compromised via any of the breaches mentioned:

- Breach of confidentiality: This type of violation involves the unauthorized reading of data.
- Breach of integrity: This violation involves unauthorized modification of data.
- Breach of availability: It involves unauthorized destruction of data.
- Theft of service: It involves the unauthorized use of resources.
- Denial of service: It involves preventing legitimate use of the system. As mentioned before, such attacks can be accidental in nature.

Security System Goal: Henceforth, based on the above breaches, the following security goals are aimed:

1.Integrity: The objects in the system mustn't be accessed by any unauthorized user & any user not having sufficient rights should not be allowed to modify the important system files and resources.

2.Secretcy: The objects of the system must be accessible only to a limited number of authorized users. Not everyone should be able to view the system files.

3.Availability: All the resources of the system must be accessible to all the authorized users i.e. only one user/process should not have the right to hog all the system resources. If such kind of situation occurs, denial of service could happen. In this kind of situation, malware might hog the resources for itself & thus preventing the legitimate processes from accessing the system resources.

Threats can be classified into the following two categories:

1.Program Threats: A program was written by a cracker to hijack the security or to change the behavior of a normal process. In other words, if a user program is altered and further made to perform some malicious unwanted tasks, then it is known as Program Threats.

2.System Threats: These threats involve the abuse of system services. They strive to create a situation in which operating-system resources and user files are misused. They are also used as a medium to launch program threats.

Types of Program Threats:

1.Virus: An infamous threat, known most widely. It is a self-replicating and malicious thread that attaches itself to a system file and then rapidly replicates itself, modifying and destroying essential files leading to a system breakdown.

Further, Types of computer viruses can be described briefly as follows:

- file/parasitic - appends itself to a file
- boot/memory - infects the boot sector
- macro - written in a high-level language like VB and affects MS Office files
- source code - searches and modifies source codes
- polymorphic - changes in copying each time
- encrypted - encrypted virus + decrypting code
- stealth - avoids detection by modifying parts of the system that can be used to detect it, like the read system call
- tunneling - installs itself in the interrupt service routines and device drivers
- multipartite - infects multiple parts of the system

2.Trojan Horse: A code segment that misuses its environment is called a Trojan Horse. They seem to be attractive and harmless cover programs but are really harmful hidden programs that can be used as the virus carrier. In one of the versions of Trojan, the User is fooled to enter confidential login details on an application. Those details are stolen by a login emulator and can be further used as a way of information breaches. One of the major as well as a serious threat or consequences of the Trojan horse is that it will actually perform proper damage once installed or run on the computer's system but at first, a glance will appear to be useful software and later turns out to be maliciously unwanted one.

Another variance is Spyware, Spyware accompanies a program that the user has chosen to install and download ads to display on the user's system, thereby creating pop-up browser windows and when certain sites are visited by the user, it captures essential information and sends it over to the remote server. Such attacks are also known as Convert Channels.

3.Trap Door: The designer of a program or system might leave a hole in the software that only he is capable of using, the Trap Door works on similar principles. Trap Doors are quite difficult to detect as to analyze them, one needs to go through the source code of all the components of the system. In other words, if we may have to define a trap door then it would be like, a trap door is actually a kind of a secret entry point into a running or static program that actually

allows anyone to gain access to any system without going through the usual security access procedures.

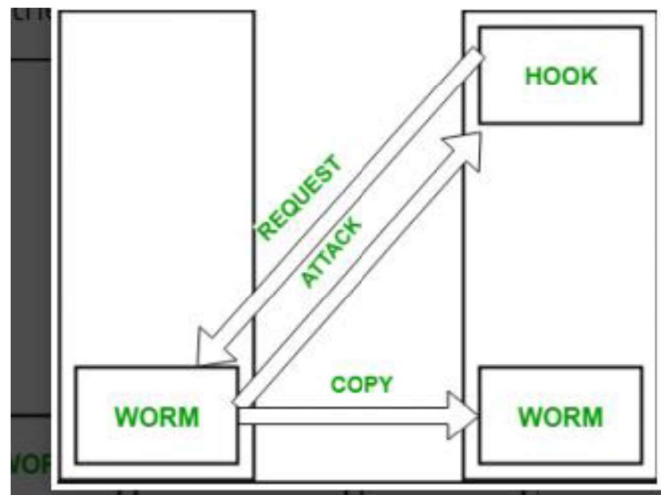
4.Logic Bomb: A program that initiates a security attack only under a specific situation. To be very precise, a logic bomb is actually the most malicious program which is inserted intentionally into the computer system and that is triggered or functions when specific conditions have been met for it to work.

5.Worm: A computer worm is a type of malware that replicates itself and infects other computers while remaining active on affected systems. A computer worm replicates itself in order to infect machines that aren't already infested. It frequently accomplishes this by taking advantage of components of an operating system that are automatic and unnoticed by the user. Worms are frequently overlooked until their uncontrolled replication depletes system resources, slowing or stopping other activities.

Types of System Threats - Aside from the program threats, various system threats are also endangering the security of our system:

1. Worm: An infection program that spreads through networks. Unlike a virus, they target mainly LANs. A computer affected by a worm attacks the target system and writes a small program "hook" on it. This hook is further used to copy the worm to the target computer. This process repeats recursively, and soon enough all the systems of the LAN are affected. It uses the spawn mechanism to duplicate itself. The worm spawns copies of itself, using up a majority of system resources and also locking out all other processes.

The basic functionality of the worm can be represented as:



2. Port Scanning: It is a means by which the cracker identifies the vulnerabilities of the system to attack. It is an automated process that involves creating a TCP/IP connection to a specific port. To protect the identity of the attacker, port scanning attacks are launched from Zombie Systems, that is systems that were previously independent systems that are also serving their owners while being used for such notorious purposes.

3. Denial of Service: Such attacks aren't aimed for the purpose of collecting information or destroying system files. Rather, they are used for disrupting the legitimate use of a system or facility.

These attacks are generally network-based. They fall into two categories:

- Attacks in this first category use so many system resources that no useful work can be performed.
For example, downloading a file from a website that proceeds to use all available CPU time.
- Attacks in the second category involve disrupting the network of the facility. These attacks are a result of the abuse of some fundamental TCP/IP principles. the fundamental functionality of TCP/IP.

Security Measures Taken - To protect the system, Security measures can be taken at the following levels:

Physical:

The sites containing computer systems must be physically secured against armed and malicious intruders. The workstations must be carefully protected.

Human:

Only appropriate users must have the authorization to access the system. Phishing(collecting confidential information) and Dumpster Diving(collecting basic information so as to gain unauthorized access) must be avoided.

Operating system:

The system must protect itself from accidental or purposeful security breaches.

Networking System:

Almost all of the information is shared between different systems via a network. Intercepting these data could be just as harmful as breaking into a computer. Henceforth, Network should be properly secured against such attacks.

Usually, Anti Malware programs are used to periodically detect and remove such viruses and threats. Additionally, to protect the system from Network Threats, a Firewall is also be used.

Program Threats:

Program threats are malicious programs that compromise a system's security or alter its normal behavior, including viruses, trojans, logic bombs, and worms. These threats can be initiated through various methods, such as exploiting software vulnerabilities, using social engineering, or attaching to legitimate programs to spread malicious code.

Types of program threats

Viruses: Malicious code that attaches itself to legitimate programs and replicates when the infected file is executed, often corrupting data or causing system crashes.

Trojans (Trojan Horses): Malware that disguises itself as a legitimate program to trick users into installing it. Once installed, it can perform malicious activities like stealing information or creating a backdoor.

Worms: Self-replicating threats that spread across networks to infect other computers without user interaction.

Logic Bombs: Malicious code that lies dormant until a specific trigger event occurs, such as a particular date or time, at which point it activates its harmful payload.

Rootkits: Programs that gain privileged access to a system and hide their presence, making them very difficult to detect and remove.

Ransomware: Malware that encrypts a user's files and demands payment for the decryption key.

Trap Doors: Security holes deliberately left in a program's code, allowing unauthorized access later without going through normal security procedures.

Buffer Overflow Exploits: Occur when a program attempts to write data beyond the allocated memory buffer, which can be exploited to execute arbitrary code or crash the system.

How they work

Exploiting vulnerabilities: Program threats often exploit weaknesses in software, such as buffer overflow vulnerabilities, to execute their code.

Social engineering: They can manipulate users into taking actions that compromise security, such as clicking a malicious link or downloading an infected file.

Preying on user trust: Trojans and other forms of malware use deception, appearing as legitimate software to gain access to a system.

How they differ from system threats

- **Program threats** involve a malicious program that alters the behavior of normal processes or compromises security directly.
- **System threats** involve the misuse of existing system services, resources, and user files, often initiated through network communication.

System And Network Threats:

System and network threats in an OS exploit vulnerabilities in operating systems and network connections to cause harm, such as stealing data, disrupting services, or gaining unauthorized access. Common threats include malware like viruses and worms, denial-of-service (DoS) attacks, phishing, and exploiting software bugs like buffer overflows and SQL injection.

Common system and network threats

- **Malware:** A broad category of malicious software including viruses, worms, trojans, spyware, and ransomware that can damage systems, steal data, or provide backdoor access.

- **Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS):** Attacks that overwhelm a system or network with traffic, preventing legitimate users from accessing it.
- **Phishing:** A social engineering tactic where attackers deceive users into revealing sensitive information, such as passwords or credit card numbers, through fraudulent emails or websites.
- **Port Scanning:** An attacker method to detect vulnerabilities on a system by sending out packets to its ports.
- **SQL Injection:** Injecting malicious SQL code into vulnerable web applications to manipulate databases, steal data, or gain unauthorized access.
- **Buffer Overflow:** An attack that exploits software bugs by overflowing a buffer, allowing attackers to overwrite memory and execute malicious code.
- **Insider Threats:** Threats originating from inside an organization, such as a malicious employee or a compromised account, who has authorized access.
- **Unauthorized Access:** Gaining access to a system or network without permission through various means, including weak passwords or misconfigurations.
- **Man-in-the-Middle (MitM) Attacks:** An attacker secretly intercepts and relays communication between two parties who believe they are communicating directly with each other.
- **Rootkits:** A type of malware that allows attackers to hide their activities and maintain persistent, privileged access to a system by modifying the operating system kernel.

Q) User Authentication:

Today when more and more companies and organizations are going digital, the security of data and authorization to important systems and services is crucial. User authentication is central to this security paradigm as it refers to the mechanism by which the identity of a user is first confirmed before being granted access to a resource.

What is User Authentication?

User authentication is the process of establishing the identity of an individual who wants to have access to a particular system or service. It involves a process of ensuring that the user claiming to be a specific personality is substantial through proving credentials like passwords, biometric data, security tokens, or other authenticity factors. This verification step is essential in safeguarding systems and systems' components from access and use by unauthorized individuals and entities and in preventing misuse of information that is considered confidential or sensitive.

How User Authentication Works

- **Presentation of Credentials:** It may be in the form of a user ID and password, a fingerprint, or other such scans, a token that the user presents, or any other form of authentication factor that the user presents.

- **Transmission to the Authenticating System:** It forwards the materials to the authenticating system for verification and authorization based on the included information in the document.
- **Verification:** It checks that the provided information is sufficient for the type of access by verifying it with the stored entries.
- **Granting Access:** If the identification details provided are tied up with that in the records and the system has succeeded in authenticating the user, access can be granted to the system or the service.
- **Access Denied:** If many of the details of the credentials given do not match the details of the database provided, or there is evidence of phishing or hijacking, access can be denied, and further security issues could arise, such as alerting the user or system administrator or even authenticating the credentials further.

User Authentication Types:

- **Password-based Authentication:** User authentication types include password-based authentication, whereby people use a unique code to be allowed entry and use a given system or service. It is an easy-to-implement and commonly used technique, though it may be exposed to password theft or guessing attacks. These practices would include proper password management, such as adopting and implementing strict password management policies, besides practicing techniques such as salting and hashing of passwords.
- **Biometric Authentication:** The technique used for validation is a print of a finger, face recognition, or an iris scan, where it is the biometric structure of a user that is identified. Biometric data cannot be easily duplicated, and for that reason, its existence guarantees high security. But the flipside is privacy issues, the other instances include the false negatives and false positives being a bit questionable.
- **Two Factor Authentication (2FA):** This relies on something in the user's possession, such as the password, and also something the user has at a particular period, such as a smartphone or a token. Generally, the security level is too high because if an intruder has access to one of the used factors, the data won't be available. After all, it is impossible to access something you don't have.
- **Multi-factor Authentication (MFA):** Unlike MFA, there is followed by '2FA' and is presupposed from the need to input two or more independent values that can be something they know, possess, or are, namely biometric data. It takes this idea further to enhance security even further, but there can be downsides both in terms of the technical complexity and convenience for an end user.
- **Single Sign-On (SSO):** SSO is needed when a user needs to get into many related or integrated systems or services but using the same username and password. Before, the user had to remember a handful of passwords to log in to his accounts on websites, now, this is only reduced to memorizing just a single password. The convenience can only be equaled by a security flaw of centralizing control, this may quickly become a headache if the credentials were to fall into the wrong hands. Now, all other systems that use similar credentials stand compromised.

- **Token-based Authentication:** A unique token is created for the users. To continue using the system, the users must hold on to this token. There are two types of tokens: physical tokens, such as the USB token, and logical tokens, which are entirely software-based. This is a more secure form than the old-style one, although, of course, links are activated with a password rather than through a token, which is really hard to break or easily duplicated. This, though, might pose an operational issue concerning the process of control and distribution of tokens.

Why is User Authentication Important?

- **Security:** It guarantees that only authorized personnel can access information and resources, thus preventing outsiders or even insiders from accessing resources they are not supposed to access or illicitly obtain data on the resources.
- **Data Protection:** This employs formal protection mechanisms to limit access to the data, and to prevent unauthorized modification or destruction of the information.
- **Compliance:** Most regulatory requirements and industry standards include UX as one of the most essential elements to observe data protection and privacy requirements.
- **User Accountability:** While authentication provides for the identification of users, thereby enabling the rooting of interactions with systems to particular individuals, it also enables accountability for occurred actions.
- **Trust and Confidence:** In this sense, strong authentication methods raise confidence levels and security among users, clients, and investors, as well as the reliability of the systems and services requested.
- **Prevention of Fraud:** There are several benefits of authentication mechanisms including, Safeguard against identity theft, fraud, and other malicious activities whereby the identity of a person is confirmed before he or she is allowed to perform any activity or transaction on SNS.

How to Improve User Authentication?

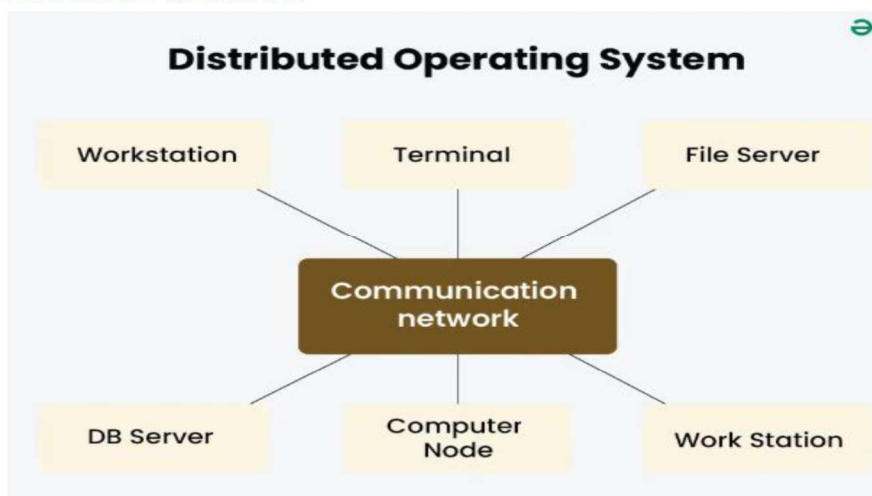
- **Implement Strong Password Policies:** Make your users adhere to a few good password habits by enforcing best practices that disallow weak and guessable passwords. Make Password Patterns and Change them at Regular intervals. Teaching users to change default passwords and to use unique passwords for every account will help reduce the damage from lost credentials.
- **Enable Multi-Factor Authentication (MFA):** MFA adds another layer of identity verification to password entry, such as fingerprint scans, fob tokens, and SMS codes. Regular 2 factors and Multi Factor Authentication reduces the risk of breach significantly compared to just passwords as it combines having something the user in question knows with something they have or are, giving a much stronger security stance.
- **Use Biometric Authentication:** Implement biometric authentication including fingerprint as well as facial recognition which are quite convenient for them. Unlike passwords, which can be stolen and forged, biometric data are unique to each individual, they are an extremely reliable way of ensuring the identity of a user.

- **Frequently Update and Patch Authentication Systems:** Stay one step ahead of things by implementing frequent updates and patches to the authentication system. Updating the application helps to fix vulnerabilities and secure Authentication methods against newly identified threats.
- **Teach Your Users About Security:** Let your users know about risks like account spoofing and other types of scams. Urgent account credential protection Continuous training and awareness campaigns make the users more informed decision-makers and keep them on alert against social engineering techniques.
- **Roll out adaptive authentication:** You can implement context-aware security by enforcing authentication measures based on factors such as user behavior, device type used, or regularity of access. Adaptive method to authenticate user risk level in the security-based decisions, providing a flexible framework for factors and measures of effective access.
- **Monitor and Analyze Authentication Events:** Install logging and monitoring solutions to monitor authentication events in real time, indicating anomalous or suspicious activities. Detection: The ability to analyze authentication logs can help detect a threat before the covered event and thwart it on time for proactive prevention.

About Distributed Operating System:

A Distributed Operating System refers to a model in which applications run on multiple interconnected computers, offering enhanced communication and integration capabilities compared to a network operating system. Here, multiple CPUs are utilized, but for end-users, it appears as a typical centralized operating system.

Structure of Distributed OS: A Distributed Operating System (DOS) manages a collection of independent computers and makes them appear to users as a single unified system. The key structural components connected through a Communication Network are as follows:



1. Workstation

- A general-purpose computer used by users to perform tasks.
- Provides user interface and local computing resources.
- Accesses shared resources on the network.

2. Terminal

- A simple input/output device.
- Relies on the processing power of remote computers.
- Acts as an interface to communicate with the distributed system.

3. File Server

- Responsible for centralized storage of files.
- Manages file access, storage and retrieval for all nodes.
- Provides shared file system access to workstations and terminals.

4. Database (DB) Server

- Handles centralized database services.
- Manages data consistency, transactions and concurrency.
- Allows multiple nodes to query and manipulate data efficiently.

5. Computer Node

- A general-purpose compute node in the distributed system.
- Executes tasks, runs processes and performs calculations.
- Collaborates with other nodes to execute distributed applications.

6. Communication Network

- The backbone that interconnects all components.
- Ensures reliable data transmission between nodes, servers, terminals and workstations.
- Can be implemented using LAN, WAN or specialized distributed network technologies.

Types of Distributed OS: There are many types of Distributed Operating System, some of them are as follows:

1. Client-Server Systems: In a client-server system within a distributed operating system, clients request services or resources from servers over a network.

- Clients initiate communication, send requests and handle user interfaces, while servers listen for requests, perform tasks and manage resources.
- This model allows for scalable resource utilization, efficient sharing, modular development, centralized control and fault tolerance.
- It facilitates collaboration between distributed entities, promoting the development of reliable, scalable and interoperable distributed systems.

2. Peer-to-Peer(P2P) Systems: In peer-to-peer (P2P) systems, interconnected nodes directly communicate and collaborate without centralized control. Each node can act as both a client and a server, sharing resources and services with other nodes.

- P2P systems enable decentralized resource sharing, self-organization and fault tolerance.
- They support efficient collaboration, scalability and resilience to failures without relying on central servers.
- This model facilitates distributed data sharing, content distribution and computing tasks, making it suitable for applications like file sharing, content delivery and blockchain networks.

3. Middleware: Middleware acts as a bridge between different software applications or components, enabling communication and interaction across distributed systems.

- It abstracts complexities of network communication, providing services like message passing, remote procedure calls (RPC) and object management.
- Middleware facilitates interoperability, scalability and fault tolerance by decoupling application logic from underlying infrastructure.
- It supports diverse communication protocols and data formats, enabling seamless integration between heterogeneous systems.
- Middleware simplifies distributed system development, promotes modularity and enhances system flexibility, enabling efficient resource utilization and improved system reliability.

4. Three-Tier: In a distributed operating system, the three-tier architecture divides tasks into presentation, logic and data layers. The presentation tier, comprising client machines or devices, handles user interaction.

- The logic tier, distributed across multiple nodes or servers, executes processing logic and coordinates system functions.
- The data tier manages storage and retrieval operations, often employing distributed databases or file systems across multiple nodes.
- This modular approach enables scalability, fault tolerance and efficient resource utilization, making it ideal for distributed computing environments.

5. N-Tier: In an N-tier architecture, applications are structured into multiple tiers or layers beyond the traditional three-tier model. Each tier performs specific functions, such as presentation, logic, data processing and storage, with the flexibility to add more tiers as needed.

- This architecture enables complex applications to be divided into modular components distributed across multiple nodes or servers.
- Each tier can scale independently, promoting efficient resource utilization, fault tolerance and maintainability.
- N-tier architectures facilitate distributed computing by allowing components to run on separate nodes or servers, improving performance and scalability.
- This approach is commonly used in large-scale enterprise systems, web applications and distributed systems requiring high availability and scalability.

Q)Motivation:

Motivation in operating (or operations management) refers to the set of internal and external factors that stimulate employees to take action, work efficiently, and perform their tasks effectively within an organization's operational environment.

In simple words: **Motivation is what drives employees to do their work well, consistently, and willingly.**

1. Internal (Intrinsic) Motivation: This comes from **within the employee**. It is driven by personal satisfaction, enjoyment of the task, or a sense of accomplishment.

Examples of intrinsic motivators:

- Enjoying the work itself
- Wanting to improve skills
- Feeling proud of completing a task
- Personal goals and passion

2. External (Extrinsic) Motivation: This comes from **outside the employee**. It includes rewards, recognition, or avoiding negative consequences.

Examples of extrinsic motivators:

- Salary, bonuses
- Promotions
- Employee-of-the-month awards
- Avoiding punishment or warnings

Why Motivation is Important in Operations: Motivation directly affects:

- **Productivity** (more motivated workers = higher output)
- **Quality** (employees pay more attention and care)
- **Efficiency** (less waste, fewer mistakes)
- **Employee retention** (motivated employees stay longer)

Example: A Manufacturing Plant

Situation: A car manufacturing plant is facing delays in production because assembly line workers are not putting in full effort.

Management Action (Motivation Strategy):

1. **Extrinsic Motivation:**
 - Introduces a bonus system: workers get a bonus if weekly targets are met.
 - Gives recognition certificates for "Zero-Defect Workers."
2. **Intrinsic Motivation:**

- Offers skill-development training that helps workers grow.
- Gives employees autonomy to propose improvements to the assembly process.

Result:

- Workers feel rewarded for good performance (extrinsic).
- Workers also feel proud and more skilled (intrinsic).
- Production increases, defects decrease, and employee satisfaction improves.

Types of networks- based Operating systems:(Network Structure):

A **network-based operating system (NOS)** is an operating system that is specially designed to manage, operate, and support networks—allowing multiple computers to communicate, share files, printers, applications, and resources.

There are **two main types of network-based operating systems**, and **sometimes** a third category is added for modern systems:

1. **Peer-to-Peer Network Operating System (P2P NOS)**
2. **Client–Server Network Operating System**
3. **Hybrid Network Operating System** (*optional/modern category*)

1. **Peer-to-Peer Network Operating System (P2P NOS):** In a peer-to-peer network OS, **all computers are equal**. There is **no central server**. Each computer (peer) can act as **both a client and a server**, meaning any computer can share resources such as files or printers with others directly.

Key Features

- No dedicated server
- Low cost, easy to set up
- Suitable for small networks (typically 2–10 computers)
- Simple user management
- Limited security and scalability

Examples

- Windows 10/11 in peer-to-peer mode
- macOS file sharing
- Linux peer-to-peer networks

Example Scenario: In a small office with 5 computers:

- Computer A shares a printer.
- Computer B shares a folder.
- All computers access each other's resources directly.

There is **no central control**—each computer manages its own sharing settings.

2. Client–Server Network Operating System: In a client-server network OS, there is a **dedicated server** that manages and controls the entire network. Clients (users' computers) request services, and the **server provides** them.

Key Features

- Centralized management
- High security and access control
- Scalable—works well for large organizations
- Centralized storage, backup, and resource sharing
- Requires a dedicated, powerful server machine

Examples

- Windows Server 2019/2022
- Linux Server OS (Ubuntu Server, Red Hat Enterprise Linux)
- Novell NetWare
- UNIX Servers

Example Scenario: In a school computer lab:

- The server stores all student login accounts.
- Students log in from any computer (client).
- The server controls internet access, printing, software installation, etc.

Here, the server is the central point of management.

3. Hybrid Network Operating System (Modern): A hybrid network OS combines features of both **peer-to-peer** and **client-server** systems. Some services are managed centrally, while others are shared peer-to-peer.

Key Features

- Flexible management
- Supports both small and large networks
- Some resources controlled by servers, others shared locally
- Balanced performance and cost

Examples

- Windows Server with Windows 10/11 clients (mixed environment)
- Linux server managing a network of peer-to-peer Linux desktops
- Cloud-integrated systems like Microsoft Azure AD + Windows PCs

Example Scenario: In a modern office:

- A central server handles user logins and security.

- Employees can still share files directly between their laptops using peer-to-peer sharing.

Comparison Table

Feature	Peer-to-Peer NOS	Client-Server NOS	Hybrid NOS
Server Required	No	Yes	Optional
Security Level	Low	High	Moderate-High
Cost	Low	High	Medium
Scalability	Small networks	Medium-large networks	All sizes
Management	Decentralized	Centralized	Mixed
Performance	Lower	Higher	Balanced

Functions Supported by Network OS Structure

1. Remote Login (Telnet, SSH)
2. Remote File Access (NFS, SMB)
3. File & Printer Sharing
4. User Account Management
5. Network Security & Authentication
6. Communication Services (email, chat)
7. Network Resource Management

Network Structure in Operating System means the OS is designed to allow multiple computers to communicate and share resources over a network. It includes peer-to-peer, client-server, and distributed structures and supports services like file sharing, authentication, remote login, and network communication.

Network Topology

A network topology is the arrangement of devices (nodes) and connections (links) in a computer network. It shows how computers, servers, and other devices are connected and how data flows between them. There are two main types of topology:

Physical Topology: The actual physical layout of cables and devices.

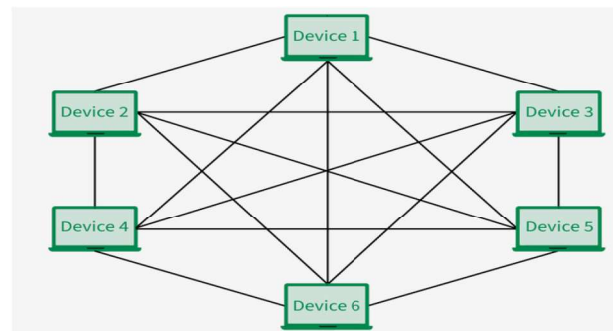
Logical Topology: How data moves across the network, regardless of physical layout.

P Topology: Point-to-point topology is a type of topology that works on the functionality of the sender and receiver. It is the simplest communication between two nodes, in which one is the sender and the other one is the receiver. Point-to-Point provides high bandwidth.

1. Point to Point Topology

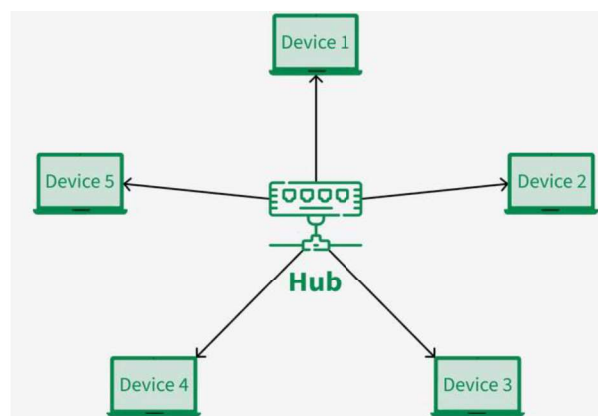


Mesh Topology: In a mesh topology, every device is connected to another device via a particular channel. Every device is connected to another via dedicated channels. These channels are known as links. In Mesh Topology, the protocols used are AHCP (Ad Hoc Configuration Protocols), DHCP (Dynamic Host Configuration Protocol), etc.



- Suppose, the N number of devices are connected with each other in a mesh topology, the total number of ports that are required by each device is $N-1$. In Figure , there are 6 devices connected to each other, hence the total number of ports required by each device is 5. The total number of ports required = $N * (N-1) = 6 * 5 = 30$.
- Suppose, N number of devices are connected with each other in a mesh topology, then the total number of dedicated links required to connect them is $\frac{N(N-1)}{2}$ i.e. $\frac{6 * 5}{2} = 15$.

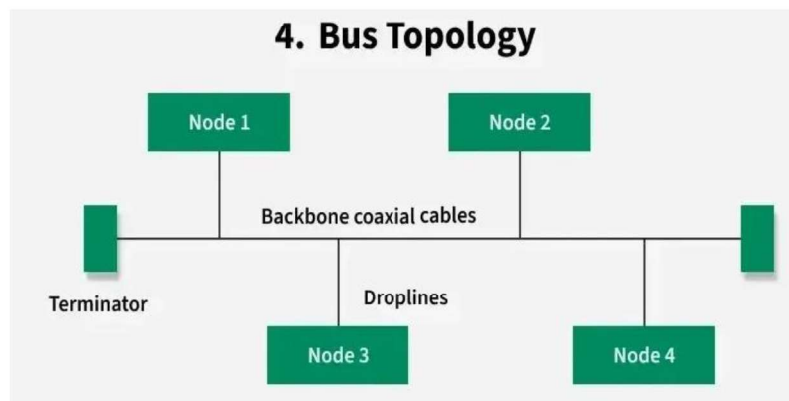
Star Topology: In Star Topology, all the devices are connected to a single hub through a cable. This hub is the central node and all other nodes are connected to the central node. The hub can be passive in nature i.e., not an intelligent hub such as broadcasting devices, at the same time the hub can be intelligent known as an active hub. Active hubs have repeaters in them.



Advantages of Star Topology

- If N devices are connected to each other in a star topology, then the number of cables required to connect them is N. So, it is easy to set up.
- Each device requires only 1 port i.e. to connect to the hub, therefore the total number of ports required is N.
- It is Robust. If one link fails only that link will affect and not other than that.
- Easy to fault identification and fault isolation.
- Star topology is cost-effective as it uses inexpensive coaxial cable.

Bus Topology: Bus Topology is a network type in which every computer and network device is connected to a single cable. It is bi-directional. It is a multi-point connection and a non-robust topology because if the backbone fails the topology crashes. In Bus Topology, various MAC (Media Access Control) protocols are followed by LAN ethernet connections like TDMA, Pure Aloha, CDMA, Slotted Aloha, etc.

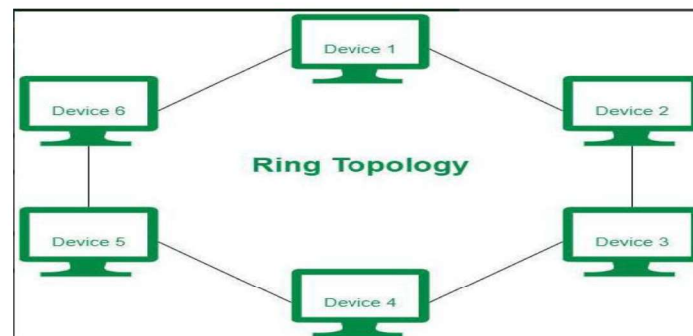


Advantages of Bus Topology

- If N devices are connected to each other in a bus topology, then the number of cables required to connect them is 1, known as backbone cable, and N drop lines are required.
- Coaxial or twisted pair cables are mainly used in bus-based networks that support up to 10 Mbps.
- The cost of the cable is less compared to other topologies, but it is used to build small networks.
- Bus topology is familiar technology as installation and troubleshooting techniques are well known.
- CSMA is the most common method for this type of topology.

Ring Topology: In a Ring Topology, it forms a ring connecting devices with exactly two neighboring devices. A number of repeaters are used for Ring topology with a large number of nodes, because if someone wants to send some data to the last node in the ring topology with 100 nodes, then the data will have to pass through 99 nodes to reach the 100th node. Hence to prevent data loss repeaters are used in the network.

In-Ring Topology, the Token Ring Passing protocol is used by the workstations to transmit the data where, Token passing is a network access method in which a token is passed from one node to another node & Token is a frame that circulates around the network.



Operations of Ring Topology

- One station is known as a monitor station which takes all the responsibility for performing the operations.
- To transmit the data, the station has to hold the token. After the transmission is done, the token is to be released for other stations to use.
- When no station is transmitting the data, then the token will circulate in the ring.

Advantages of Ring Topology

- The data transmission is high-speed.
- The possibility of collision is minimum in this type of topology.
- Cheap to install and expand.

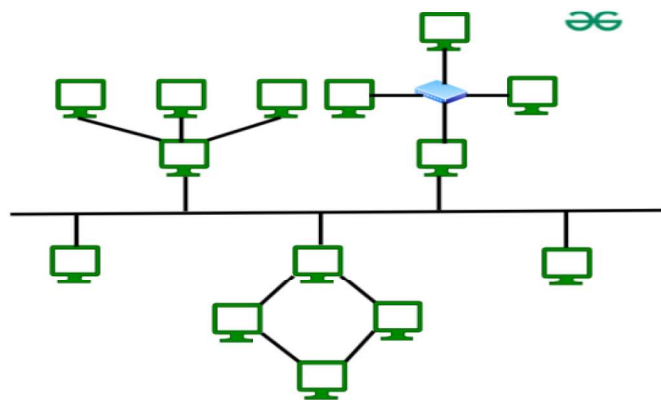
Tree Topology: Tree topology is the variation of the Star topology. This topology has a hierarchical flow of data. In Tree Topology, protocols like DHCP and SAC (Standard Automatic Configuration) are used.

- Here, various secondary hubs are connected to the central hub which contains the repeater.
- This data flow from top to bottom i.e. from the central hub to the secondary and then to the devices or from bottom to top i.e. devices to the secondary hub and then to the central hub.
- It is a multi-point connection and a non-robust topology because if the backbone fails the topology crashes.

Advantages of Tree Topology

- It allows more devices to be attached to a single central hub thus it decreases the distance that is traveled by the signal to come to the devices.
- It allows the network to get isolated and also prioritize from different computers.
- We can add new devices to the existing network.

Hybrid Topology: Hybrid Topology is the combination of all the various types of topologies we have studied above. Hybrid Topology is used when the nodes are free to take any form. It means these can be individuals such as Ring or Star topology or can be a combination of various types of topologies seen above. Each individual topology uses the protocol that has been discussed earlier.



Advantages of Hybrid Topology

- This topology is very flexible.
- The size of the network can be easily expanded by adding new devices.

Communication Structure:

Communication Structure **in an operating system refers to the** methods, mechanisms, and architecture that allow processes, programs, or systems to exchange data and messages.

This communication may happen:

- **Within the same computer** (Process–Process Communication)
- **Between different computers** (Network Communication)

The OS manages communication to ensure data is:

- ✓Delivered correctly
- ✓Delivered to the right process

✓Synchronized

✓Secure

This structure is mainly implemented using **IPC (Inter-Process Communication)** mechanisms and **network communication protocols**.

Two Main Types of Communication Structures

1. **Inter-Process Communication (IPC)**

- Communication between processes on the **same machine**

2. **Network Communication**

- Communication between processes on **different machines** using a network

1. Inter-Process Communication (IPC) Structure: IPC allows processes to share data and coordinate their actions. The OS provides different IPC mechanisms.

IPC Methods in the Communication Structure

(a) Pipes

- One-way communication channel.
- Used between related processes.

Example: In Linux, `ls | grep "txt"`
Output of `ls` goes to `grep` through a pipe.

(b) Message Queues

- Messages stored in a queue.
- Processes send and receive messages asynchronously.

Example: A banking app:

- **Deposit process** sends message → “Add ₹1000”
- **Account process** reads message from queue and updates balance

(c) Shared Memory

- A region of memory shared by multiple processes.
- Fastest communication method.
- Requires synchronization.

Example: Two processes (Sensor app & Monitoring app) share the same memory area to store temperature readings.

(d) Semaphores

- Used to synchronize multiple processes.
- Prevents race conditions.

Example: Two processes trying to access the same printer use a semaphore to avoid conflict.

(e) Sockets

- Allow communication between processes on the same or different computers.
- Used in network applications.

Example:

A web browser (client) communicates with a web server using TCP sockets.

(f) Signals

- Small messages to notify processes of events (interrupts, errors, termination).

Example: Pressing **Ctrl + C** sends `SIGINT` to terminate a program.

2. Network Communication Structure: Used when processes run on **different computers**.

OS provides:

- Protocols (TCP/IP, UDP)
- Remote Procedure Calls (RPC)
- Remote Method Invocation (RMI)
- Distributed file systems (NFS, SMB)
- Network sockets

Example: You send a message on WhatsApp:

- Your device's OS uses **sockets + TCP/IP** to send data to WhatsApp servers.
- The server forwards it to the receiver. This entire communication is managed by the OS communication structure.

Why Communication Structure is Important in OS

- Allows multitasking systems to run efficiently
- Supports client-server applications
- Enables distributed computing
- Handles synchronization
- Ensures data integrity and security
- Allows modular application design

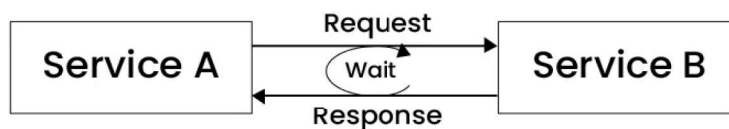
Q) Communication Protocols:

Modern distributed systems rely heavily on communication protocols for both design and operation.

- Communication protocols facilitate smooth coordination and communication in distributed systems by defining the norms and guidelines for message exchange between various components.
- By choosing the right communication protocol, system architects can design systems that are scalable, reliable, and efficient.

Synchronous Communication: The pattern of communication known as "synchronous communication" occurs when services exchange requests and answers, typically waiting for a response before proceeding. This means that when a micro service sends a request to another micro service, it will stop and wait for a response before continuing with its function.

Synchronous Communication



Example of Synchronous Communication:

- HTTP Request-Response: One micro service can submit an HTTP request to another micro service and wait for a response by using HTTP protocols such as REST or SOAP.
- RPC (Remote Procedure Call): Services can use RPC frameworks like gRPC to make remote procedure calls and wait for the response before continuing.
- Synchronous Messaging: Some message brokers support synchronous messaging patterns, where a service sends a message and waits for a response from another service.

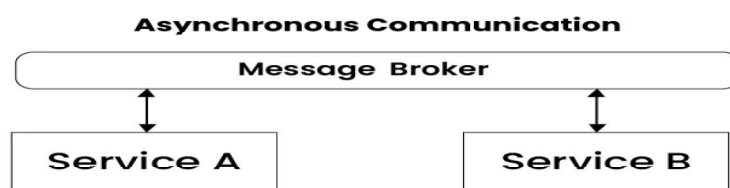
Applications of Synchronous Communication: Below are the applications of Synchronous Communication:

- Real-Time Messaging Applications: Used in chat apps like WhatsApp or Slack where messages are exchanged instantly between users.
- Database Operations: Suitable for operations requiring immediate confirmation, like reading or updating critical data in transactional systems.
- Payment Gateways: Ensures immediate feedback for payment authorization or failure in online transactions.
- APIs Requiring Immediate Response: Services like authentication APIs or search queries that require instant results.
- Video Conferencing and Calls: Applications like Zoom or Google Meet use synchronous communication for real-time audio and video data transfer.
- Remote Procedure Calls (RPCs): Often employed when one service needs an immediate response from another, as in micro services-based systems.

Challenges of Synchronous Communication: Below are the challenges of Synchronous Communication:

- **Latency:** Synchronous communication can introduce latency, especially if services are waiting for responses from slow or unresponsive services.
- **Blocking Nature:** Services can become blocked if they are waiting for a response, potentially leading to performance issues.
- **Complexity:** While simpler than asynchronous communication, synchronous communication can still add complexity, especially in large micro services architectures.
- **Error Handling:** Error handling in synchronous communication can be more challenging, as services need to deal with immediate failures.
- **Scalability:** Synchronous communication can be less scalable than asynchronous communication, as services need to handle more concurrent connections and requests.

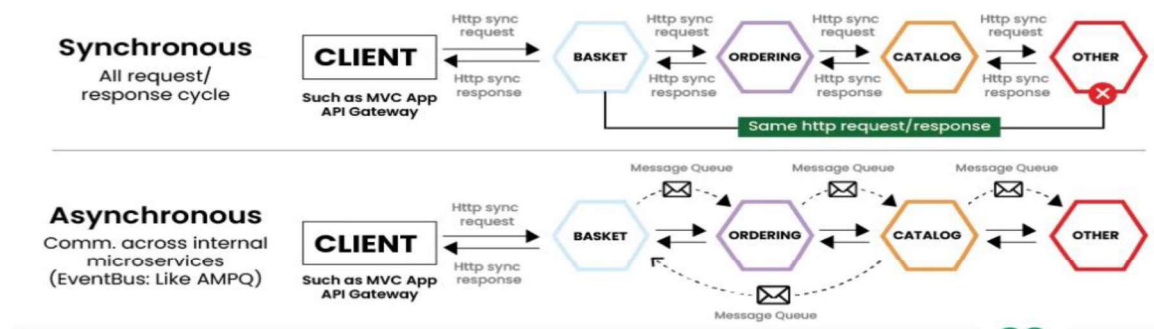
Asynchronous Communication: Asynchronous communication refers to a communication pattern where services exchange messages or data without waiting for an immediate response. This allows services to operate independently and asynchronously, enabling decoupling and scalability in distributed systems.



Example of Asynchronous Communication

- **Message Queues:** Message queues such as RabbitMQ or Kafka are used by services to transmit messages. Without waiting for a response, the sending service adds a message to the queue and carries on with its work. When the message is ready, it is processed by the receiving service.
- **Event-Driven Architecture:** Services use a publish-subscribe model to interact through events. Any interested services (subscribers) process the event asynchronously once it is published by the publishing service.
- **Background Processing:** Micro services can offload long-running or non-urgent tasks to background processes. For instance, a task may be queued for processing by a service and handled asynchronously by a distinct worker process.

Differences between Synchronous and Asynchronous Communication



Feature	Synchronous Communication	Asynchronous Communication
Definition	Real-time interaction where services wait for responses	Communication where services send messages without waiting
Waiting for Response	Services wait for responses before proceeding	Services do not wait for responses and continue immediately
Timing	Requires services to be available at the same time	Services can communicate at their convenience
Examples	HTTP Request-Response, RPC	Message Queues, Event-Driven Architecture, Background Processing
Flexibility	Less flexible, as services need to be available simultaneously	More flexible, as services can communicate independently
Complexity	Generally simpler to implement and understand	Can be more complex due to message buffering and error handling
Scalability	Can be less scalable, as services may block while waiting	More scalable, as services can handle multiple requests concurrently
Error Handling	Easier to handle immediate failures	Errors may be more challenging to handle due to asynchronicity
Use Cases	Suitable for real-time interactions and request-response patterns	Suitable for decoupling services and handling high loads

Q) Robustness and Design Issues.

Robustness testing is a quality assurance methodology focused on testing the robustness of the software and helps in removing the reliability of the software by finding the corner cases by inputting the data that mimics the extreme environmental conditions to determine whether or not the system is robust enough to deliver the required functionality to the user.

What is Robustness in Software testing?

Robustness is a measure of how well a software system can cope with invalid inputs or unexpected user interactions. A robust system is one that continues to function correctly even in the face of unexpected or invalid inputs. A software system that is robust is able to handle errors and unexpected inputs gracefully, without crashing or producing incorrect results. It is also able to adapt to changes in its operating environment, such as changes in the operating system, hardware, or other software components.

- Robustness is an important quality for any software system, but it is especially important for systems that are safety-critical or mission-critical.
- For these systems, a failure could have serious consequences, so it is essential that they be able to handle any unexpected inputs or conditions gracefully.

What is Robustness Testing?

Robustness testing is a type of testing that is performed to assess the ability of a system or component to function correctly when it is subjected to invalid or unexpected inputs, or when it is operating outside of its specified operating conditions. It is typically used to test for the presence of memory leaks or other types of errors that can cause a system to crash. Robustness testing is also sometimes referred to as reliability testing, stress testing, or endurance testing.

- The purpose of robustness testing is to identify the parts of the system that are most vulnerable to failure and to determine how the system can be made more resistant to failure.
- Robustness testing is typically conducted by subjecting the system to a variety of stressful conditions, such as high temperatures, high humidity, high pressure, and high levels of vibration.
- Robustness testing is typically conducted during the later stages of software testing after the software has been shown to work correctly under normal conditions.
- A common example of robustness testing is testing how a system responds to unexpected input values.

- For example, if a system is designed to accept numerical input values between 1 and 10, a robustness test would involve trying to input values outside of this range, such as 0, 11, or -5, to see how the system responds.
- Another example of robustness testing would be testing how a system responds to unexpected environmental conditions, such as excessive heat, cold, or humidity.

Why is Robustness Testing important?

- **Handle Unexpected Inputs:** Robustness testing is important because it helps ensure that a system can handle unexpected or abnormal inputs without crashing.
- **Uncover Potential Issues:** This type of testing can help uncover potential issues that could cause a system to fail in unexpected ways. By uncovering these issues early on, they can be fixed before the system is put into production.
- **Test Limits:** It allows developers to test the limits of their software and ensure that it can handle unexpected inputs and situations.
- **Uncover Hidden Bugs:** This type of testing can help uncover hidden bugs that could cause major problems in the field.
- **Stability:** This can help to prevent software failures and crashes, and can also help to improve the overall stability of your software.

Robustness Testing Scenarios

- **Navigation:** In this scenario, the tester will try to break the navigation of the application by trying to access pages or features that are not intended to be accessed by the user. This can be done by trying to access pages directly through the URL or by trying to access hidden features by manipulating the application's code.
- **Functionality:** In this scenario, the tester will try to break the functionality of the application by trying to input invalid data or by trying to perform actions that are not intended to be performed by the user. This can be done by trying to submit invalid data through the application's forms or by trying to access hidden features by manipulating the application's code.
- **Security:** In this scenario, the tester will try to break the security of the application by trying to exploit known vulnerabilities or by trying to gain access to sensitive data. This can be done by trying to inject SQL code into the application's forms or by trying to access restricted areas of the application.
- **Invalid input:** The system should be able to handle invalid input gracefully. For example, if a user enters an invalid command, the system should display an error message and prompt the user for another input.
- **Interfaces with new software modules:** The system should be able to interface with new software modules without any issues. For example, if a new software module is added to the system, the system should be able to communicate with it without any errors.

- **Changes in the environment:** The system should be able to function properly in different environments. For example, if the system is moved from one location to another, it should be able to function properly in the new location.
- **External environment changes affecting the hardware system:** The system should be able to function properly even if the external environment changes affect the hardware system. For example, if the temperature or humidity changes, the system should be able to function properly.

Types of Testing to ensure Robustness of Test Suites: There are many types of testing that can ensure the robustness of test suites. Below are some of the testing types that ensure the robustness of test suites:

- **Regression testing:** Regression testing is a type of testing that is used to find bugs in software that have already been fixed. This is done by running the software with different inputs and comparing the output to the expected output. This type of testing is used to verify that a software program continues to function properly after it has been modified or updated. This type of testing is typically performed after a new version of the software has been released, or after a change has been made to the code.
- **Functional testing:** Functional testing is a type of testing that is used to verify that a system or software performs as expected. Load testing is a type of testing that is used to verify that a system or software can handle a heavy load or traffic. This type of testing is typically performed by running the software through a series of tests that exercise the various functions of the software.
- **Load testing:** Load testing is a type of testing that is used to find bugs in software by running it with different inputs and checking if the output is as expected. It is used to verify that a software program is able to handle the load that is expected to be placed on it. This type of testing is typically done by running the software through a series of tests that simulate the load that the software will experience in production.
- **Stress testing:** Stress testing involves subjecting a system to intense or extreme conditions in order to see how well it holds up. This can help to identify potential issues that may only arise under high levels of stress or strain.
- **Negative testing:** Negative testing involves deliberately providing invalid or incorrect inputs to a system in order to see how it responds. This can help to uncover errors in input validation or handling that could lead to security vulnerabilities or data loss.
- **Fuzz testing:** A fuzz test (also known as a fuzzing test) is a software testing technique, usually automated or semi-automated, that involves providing invalid, unexpected, or random data to the inputs of a computer program. The program is then monitored for exceptions such as crashes, failing built-in code assertions, or potential memory leaks. Fuzz testing is effective in finding coding errors and security vulnerabilities. This involves feeding random or invalid data into a

system and seeing how it responds. This can help to find potential vulnerabilities in the system.

- **Use case testing:** Use case testing involves testing the system with realistic scenarios to see how it responds. This can help to identify potential usability issues.
- **Security testing:** Security Testing involves testing the system for security vulnerabilities. This can help to identify potential security risks.
- **Black-box testing:** Black-box testing is a method of software testing that examines the functionality of a software program without knowing the internal code structure. The tester is only aware of what the software is supposed to do but not how it does it. Black-box testing can be used to test the functionality of a software program, the usability of a user interface, and the compliance of a program with external standards.
- **Mutation testing:** Mutation testing is a type of software testing that involves modifying a program's source code or its inputs and then testing to see if the program still behaves as expected. The goal of mutation testing is to find faults in a program's code or inputs that can cause the program to produce incorrect results.
- **Fault injection testing:** Fault injection testing is a method of testing software by introducing faults into the software program to see if the program can detect and handle the faults. Fault injection can be used to test the robustness of a program's error-handling capabilities.

Robustness Testing Tools: Automation testing tools are used to automate the testing process. They can be used to test the functionality of an application, test the performance of an application, or test the security of an application.

- **JUnit:** JUnit is a popular open-source unit testing framework for Java. It is widely used in the Java development community and is well-suited for testing robustness. JUnit is easy to use and has a wide range of features, making it a good choice for robustness testing.
- **TestNG:** TestNG is another popular open-source unit testing framework for Java. Like JUnit, it is easy to use and has a wide range of features. TestNG is also well-suited for testing robustness.
- **Spock:** Spock is a Groovy-based testing framework that can be used for testing Java applications. It has a concise and expressive syntax that makes it easy to write tests. Spock is also well-suited for testing robustness.
- **JMeter:** JMeter is a popular open-source performance testing tool. It can be used to test the performance of web applications under load. JMeter is easy to use and has a wide range of features, making it a good choice for robustness testing.
- **LoadRunner:** LoadRunner is a commercial performance testing tool from HP. It can be used to test the performance of web applications under load. LoadRunner

is more expensive than JMeter but has more features and is better suited for enterprise applications.

Advantages

- **Increased test coverage:** More test cases can be run in a shorter amount of time, allowing for increased test coverage and improved quality assurance.
- **Reduced false positives:** By testing for a wider range of conditions, false positives can be reduced, saving time and resources.
- **Improved test accuracy:** Robustness testing can help to ensure that test cases are more accurate, leading to improved quality assurance.
- **Greater flexibility:** Robustness testing can be used to test for a wide range of conditions, making it more flexible than other types of testing.
- **Increased efficiency:** Robustness testing can save time and resources by allowing for increased test coverage in a shorter amount of time.
- **Improved software quality:** By identifying and addressing errors early on in the development process, robustness testing can help improve the overall quality of the software.
- **Reduced development costs:** By catching errors early, robustness testing can save on development costs by avoiding the need for expensive rework or redesign.
- **Increased customer satisfaction:** By ensuring that the software meets the customer's expectations in terms of functionality and reliability, robustness testing can help increase customer satisfaction.
- **Improved competitiveness:** By delivering a high-quality, robust software product, businesses can gain a competitive advantage over their rivals.

Limitations

- **Not a silver bullet:** There is no one testing technique that can catch all errors in a system.
- **Often Ineffective:** Many errors are not easily detected by testing, and even when they are, it is often difficult to reproduce the conditions under which the error occurs.
- **Time-consuming:** It can take a long time to test all the possible inputs to a system and to exhaust all the possible combinations of inputs.
- **Not perfect:** Even the most thorough testing cannot guarantee that all errors have been found and fixed.