OS DEC 2023 Answers

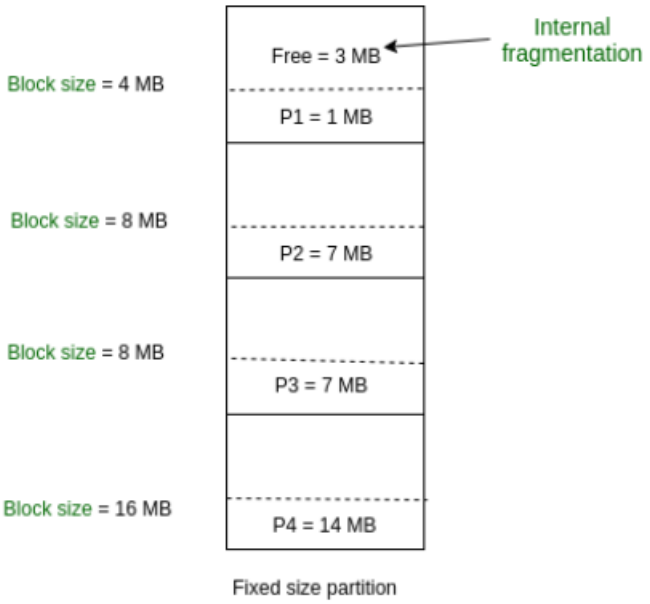
Q1

a) Explain memory fragmentation.

Memory fragmentation occurs when free memory gets broken into small pieces over time, making it difficult to allocate a large continuous block of memory even though enough total memory is available. It reduces the efficiency of memory usage and can eventually lead to allocation failures.

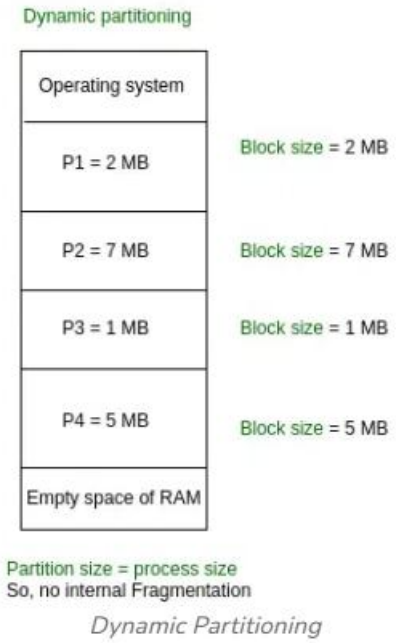
There are two types of fragmentation:

1. **Internal Fragmentation**:
   - **Description:** Unused memory spaces are scattered across the system, making it difficult to allocate contiguous memory even though sufficient space may exist.
   - **Cause:** Dynamic memory allocation (e.g., when processes request and release memory).
   - **Example:** A process needs 5 MB of memory, but the available free spaces are 2 MB, 1 MB, and 2 MB—totaling 5 MB but not contiguous.



2. External Fragmentation:

- **Description:** Occurs when allocated memory blocks are larger than what is required by the process, leaving unused space within the block.
- **Cause:** Fixed-sized memory allocation (e.g., paging).
- **Example:** A page of 4 KB is allocated to a process that needs only 2 KB, leaving 2 KB unused.



Dynamic Partitioning

Ways to Minimize Fragmentation

1. **Compaction:** Rearranges memory to consolidate free spaces into one large block (effective for external fragmentation).
2. **Paging and Segmentation:** Allows non-contiguous allocation of memory to reduce external fragmentation.
3. **Dynamic Allocation Techniques:** Allocates memory more flexibly to minimize wasted space.

d) Explain process state model.

The **Process State Model** describes the various states a process can be in during its lifecycle. The operating system (OS) manages processes by transitioning them between these states based on CPU scheduling, resource availability, and system conditions.

1. Process States

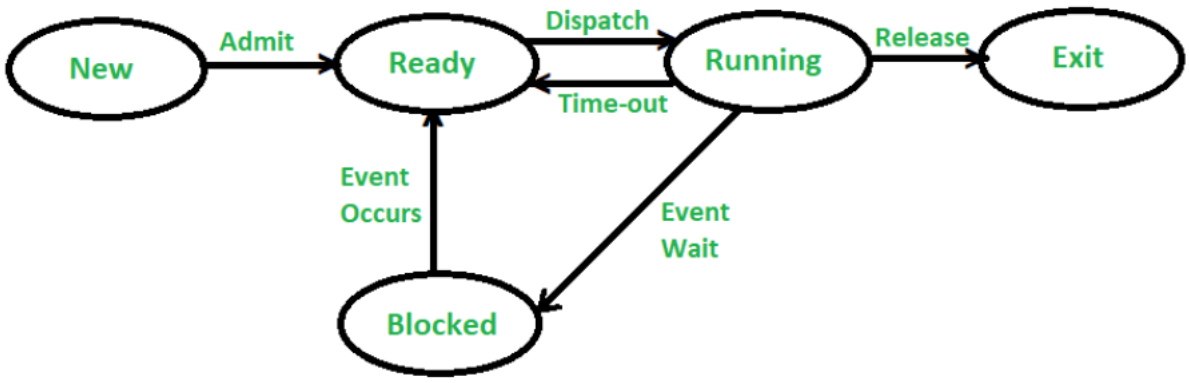
1. **New**
   - A process is created but is not yet ready for execution.
   - It is in the process of being initialized (loading into memory, setting up resources, etc.).

2. **Ready**
   - The process has all the required resources except the CPU.
   - It is waiting in the **ready queue** for the CPU to execute it.
   - The process is selected by the CPU scheduler when it gets a turn.

3. **Running**
   - The process is currently executing on the CPU.
   - At any time, only **one** process per CPU core is in this state.

4. **Blocked (Waiting)**
   - The process is waiting for an **I/O operation** or some event (e.g., user input, file access).
   - It **cannot proceed** until the requested event is completed.

5. **Terminated (Exit):** The process has finished execution or is forcibly stopped. All allocated resources are released



Five State Process Model

Transitions Between States

- **New → Ready:** After creation, the process moves to the ready state.
- **Ready → Running:** The scheduler picks the process for execution and assigns the CPU.
- **Running → Blocked:** The process requests I/O or waits for an event.
- **Blocked → Ready:** The event occurs, and the process is ready for execution.
- **Running → Terminated:** The process completes execution.

e) Explain about IPC.

Inter-Process Communication (IPC) refers to a set of mechanisms that allow processes to communicate and share data with each other.

Key Features of IPC

1. **Data Exchange**: Enables processes to send and receive data to coordinate tasks.
2. **Synchronization**: Ensures that processes cooperate without interfering with each other (e.g., avoiding race conditions).
3. **Resource Sharing**: Allows multiple processes to access shared resources like files or memory.

Common IPC Mechanisms

1. **Pipes**:
   - Unidirectional communication channel between two processes.
   - Data flows in one direction (e.g., from producer to consumer).
   - Example: A program sending output to another program as input (| in UNIX).

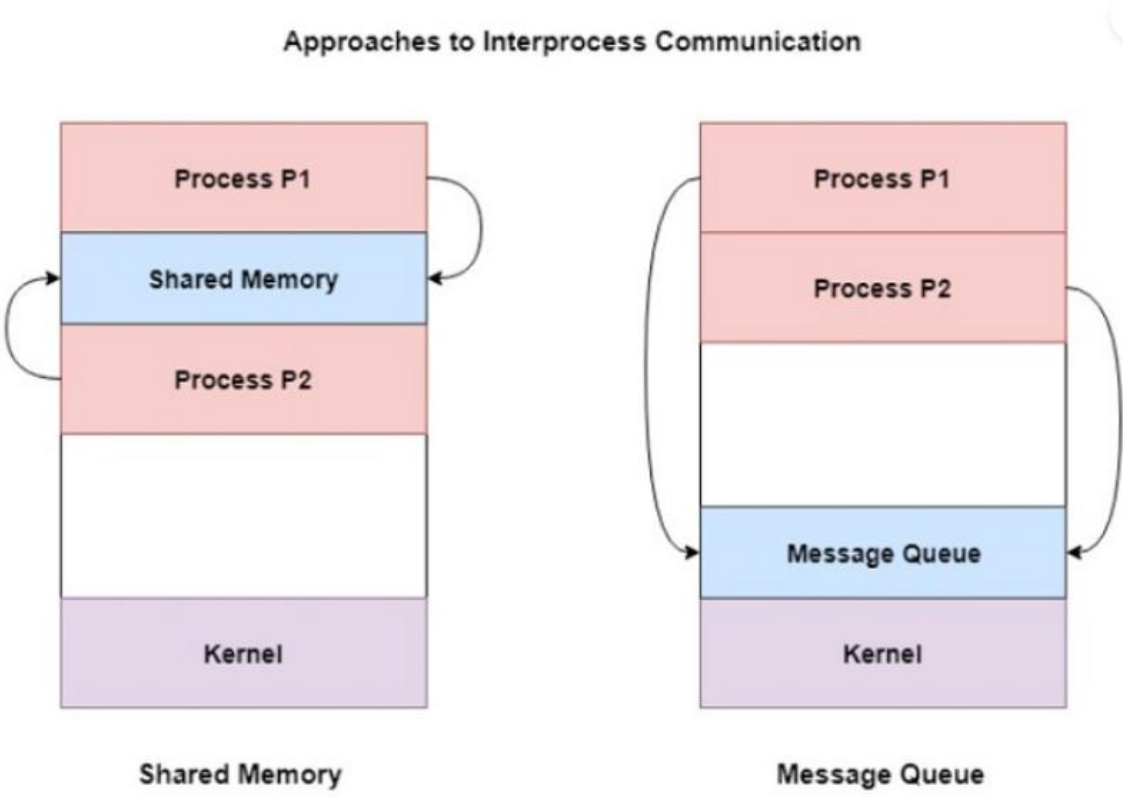
2. **Shared Memory**:
   - Allows multiple processes to access the same memory space.
   - Fast as processes directly read/write into memory, but synchronization (e.g., using semaphores) is required.

3. **Semaphores**:
   - Used for synchronization to ensure processes don't simultaneously access shared resources.
   - Prevents issues like race conditions or deadlocks.

4. **Sockets**:
    - Facilitates communication between processes, even if they're on different machines in a network.
    - Commonly used in client-server models (e.g., web browsers and servers).

Examples of IPC in Action

1. **Client-Server Model**: A web server and browser communicate using sockets.
2. **Shared Memory for Real-Time Systems**: Fast communication is achieved by directly reading/writing to shared memory.



Q2

b) What is a process? Explain Process control block in detail.

A **process** is a program in execution. It consists of program code, data, and system resources required for execution. The operating system (OS) manages processes using **process scheduling** to ensure efficient multitasking.

Characteristics of a Process

- A process requires **CPU time**, **memory**, and **I/O resources**.
- It has a **lifecycle** consisting of different **states** (New, Ready, Running, Blocked, Terminated).
- Multiple processes can execute concurrently through **multitasking**.

The answer to Process Control Block is in [**Q6 – f >> OS MAY 2023**]

Q3

a) What are different types of process scheduling algorithms? Explain anyone scheduling algorithm with example.

Types of Process Scheduling Algorithms

Process scheduling algorithms determine the order in which processes execute in a multitasking system. These algorithms are categorized into two types:

1. Preemptive Scheduling
The CPU can be **taken away** from a process if a higher-priority process arrives or its time slice expires.

Examples:
- **Round Robin (RR)**
- **Shortest Remaining Time First (SRTF)**
- **Priority Scheduling (Preemptive)**

2. Non-Preemptive Scheduling
Once a process gets the CPU, it **cannot be interrupted** until it completes or enters a waiting state.

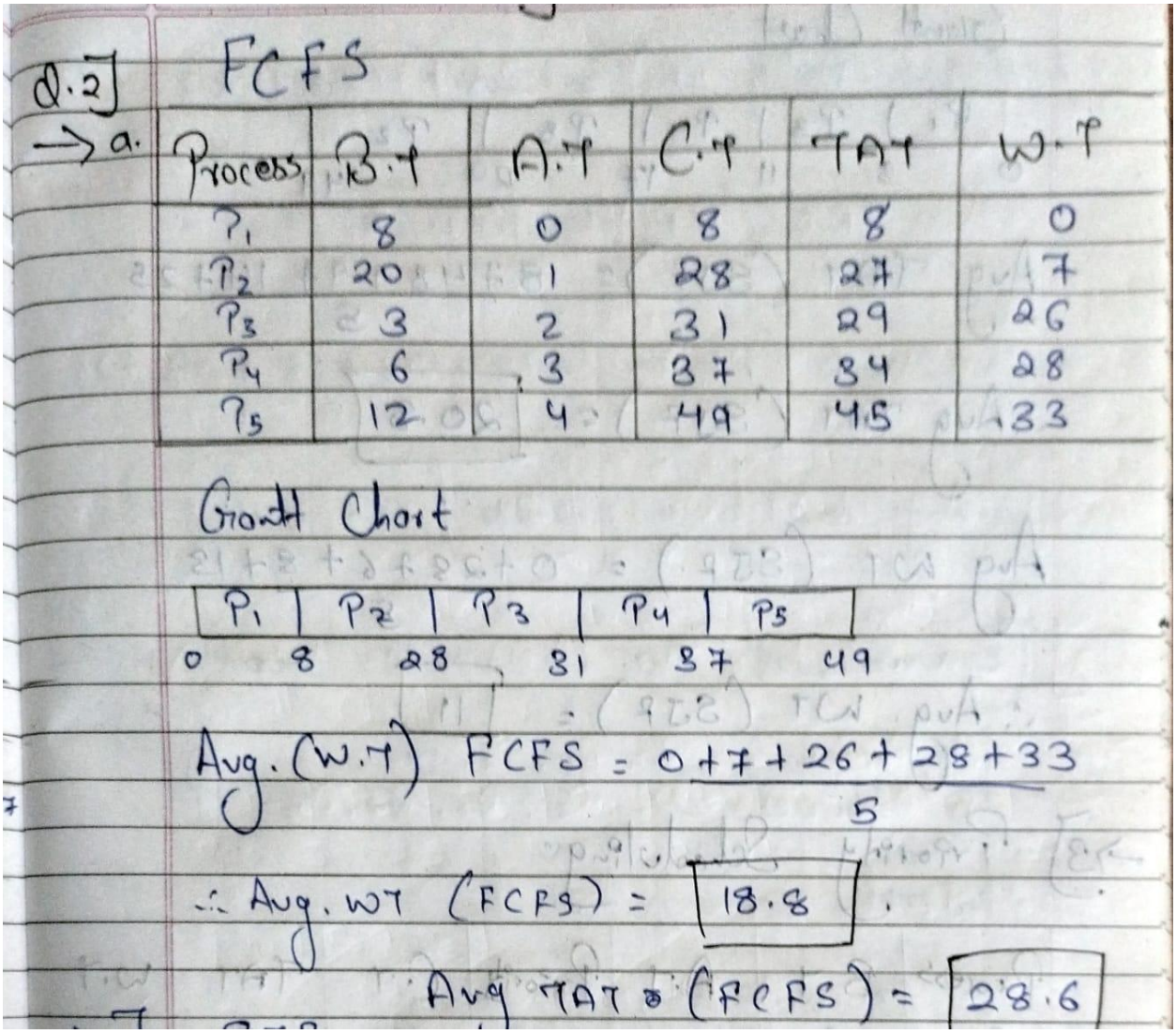
Examples:

- **First-Come, First-Served (FCFS)**
- **Shortest Job Next (SJN) / Shortest Job First (SJF)**
- **Priority Scheduling (Non-Preemptive)**

Example: First-Come, First-Served (FCFS) Scheduling

Definition:

- The simplest scheduling algorithm.
- Processes are executed in the **order they arrive** (like a queue).
- **Non-preemptive**: A process runs **until completion** before the next process starts.



Q.2] FCFS

→ a.

Process	B.T	A.T	C.T	TAT	W.T
P_1	8	0	8	8	0
P_2	20	1	28	27	7
P_3	3	2	31	29	26
P_4	6	3	37	34	28
P_5	12	4	49	45	33

Gantt Chart

P_1	P_2	P_3	P_4	P_5

0 8 28 31 37 49

Avg. (W.T) FCFS = $\frac{0+7+26+28+33}{5}$

∴ Avg. WT (FCFS) = 18.8

Avg TAT (FCFS) = 28.6

b) What is a critical region? Explain necessary conditions for deadlock.

A **critical region** (or critical section) is a part of a program where shared resources (e.g., variables, files, or data structures) are accessed or modified by multiple processes or threads. To ensure data consistency and prevent **race conditions**, only one process or thread is allowed to execute in the critical region at any given time.

Key Concepts:

1. **Mutual Exclusion:** Only one process can enter the critical region at a time to avoid interference.
2. **Synchronization Mechanisms:** Techniques like semaphores, mutexes, and monitors are used to control access.
3. **Race Condition Prevention:** Ensures that shared data is accessed safely without conflicts.

The answer to Necessary Conditions is in [**Q2 – a >> OS MAY 2023**]

Q4

b) What is an Operating System? Explain structure of Operating System.

An **Operating System (OS)** is a system software that acts as an **interface between the user and the computer hardware**.
It manages computer resources like CPU, memory, storage, and input/output devices, and provides services to users and applications to make the system easy and efficient to use.
In simple words, the OS is responsible for controlling and coordinating the use of hardware among various applications.

Structure of an Operating System
The structure of an OS defines its organization and functionality. Common structures include:

1. Monolithic Structure

- **Description:** All functionalities (e.g., process management, file management) are implemented in a single code base.
- **Features:**
  - High performance as all components are tightly integrated.
  - Difficult to debug and maintain due to lack of modularity.
- **Example:** UNIX.

2. Layered Structure

- **Description:** The OS is divided into layers, each providing specific services. Lower layers interface with hardware, while higher layers handle user interaction.
- **Features:**
  - Modularity simplifies debugging and maintenance.
  - Overhead due to layering.
- **Example:** Multics.

3. Microkernel Structure

- **Description:** Only essential services (e.g., inter-process communication, memory management) are in the kernel. Other components run in user space.
- **Features:**
  - Enhanced security and flexibility.
  - Lower performance due to communication overhead.
- **Example:** Minix.

4. Hybrid Structure

- **Description:** Combines features of monolithic and microkernel designs.
- **Features:**
  - Balances performance with modularity.
  - Supports scalability.
- **Example:** Windows.

a) Give detail comparison of user level and kernel level threads

Aspect	User-Level Threads (ULT)	Kernel-Level Threads (KLT)
Management	Managed by user-level libraries, without OS support.	Managed directly by the operating system (kernel).
Switching Speed	Fast thread switching as it does not involve kernel.	Slower switching because it involves system calls to the kernel.
OS Awareness	Operating System is unaware of user threads.	Operating System is fully aware of kernel threads.
Resource Usage	Requires less system resources.	Requires more system resources (each thread needs kernel support).
Blocking	If one thread blocks, the entire process may block.	If one thread blocks, other threads can still continue.
Portability	More portable across different operating systems.	Less portable, as it depends on OS-specific thread management.
Example	Java green threads, early Pthreads implementations.	Windows threads, Linux kernel threads.

Q5

a) Explain objectives and characteristics of modern operating system. Explain Network OS.

Objectives of Modern Operating Systems
Modern Operating Systems are designed with the following key objectives:

1. **Convenience**: Make computer systems easy and comfortable for users to operate.
2. **Efficiency**: Manage hardware resources (CPU, memory, storage, I/O devices) effectively for better performance.
3. **Resource Management**: Allocate resources like CPU time, memory space, and storage efficiently among multiple processes.
4. **Multitasking and Multiprogramming**: Allow multiple applications to run simultaneously by managing processes smartly.

Characteristics of Modern Operating Systems
Modern Operating Systems have the following important features:

- **Multitasking**: Can run multiple processes at the same time.
- **Multi-user Support**: Allows multiple users to use the system simultaneously or at different times.
- **Portability**: Can run on different types of hardware with minimal changes.
- **Security**: Protects files, data, and system resources.
- **Memory Management**: Handles memory allocation, paging, segmentation efficiently.
- **User Interface**: Provides a graphical or command-line interface for user interaction.

What is a Network Operating System (Network OS)?

A **Network Operating System (Network OS)** is a type of operating system that manages **network resources** like files, printers, and applications across multiple computers connected via a network. It enables computers to communicate, share data, and work together.

Key Features of Network OS:
- **File Sharing**: Allows sharing of files among connected computers.
- **Resource Sharing**: Printers, servers, and applications can be accessed over the network.
- **Remote Access**: Users can access files and resources even if they are physically located elsewhere.
- **Backup and Recovery**: Provides network-wide backup and recovery options.

Examples of Network OS:
- Novell NetWare
- Microsoft Windows Server
- UNIX/Linux-based server OS (with network capabilities)

b) List page replacement algorithms? Explain anyone page replacement algorithms with example.

List of Page Replacement Algorithms

Page replacement algorithms are used in virtual memory systems to decide which page to replace in case of a **page fault** (when the requested page is not in memory). Common algorithms include:

1. **FIFO (First-In-First-Out) Algorithm**
2. **Optimal Page Replacement Algorithm**
3. **LRU (Least Recently Used) Algorithm**
4. **LFU (Least Frequently Used) Algorithm**
5. **Clock (Second-Chance) Algorithm**

Explanation of FIFO Page Replacement Algorithm

The **FIFO (First-In-First-Out)** page replacement algorithm replaces the page that has been in memory the longest. It uses a simple queue to track the order in which pages were loaded into memory.

Working Steps:

1. When a page fault occurs, check if there is space in the memory for the new page:
    - If there is space, the page is loaded into memory.
    - If the memory is full, remove the **oldest page** (the page at the front of the queue).
2. Add the new page to the queue.

Q.2] Given: String → 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

→ b

→ i. LRU (Least Recently Used)

Note: Replacing the least recently used page in the past.

frames

f_4				2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
f_3			1	1	1	1	1	4	4	4	4	4	4	1	1	1	1	1	1
f_2		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_1	7	7	7	7	7	3	3	3	3	3	3	3	3	3	3	3	7	7	7
Strings →	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	×	×	×	×	Hit	×	Hit	×	Hit	Hit	Hit	Hit	Hit	×	Hit	Hit	Hit	×	Hit	

Ans: Page faults = 8

Page Hits = 12

→ ii. FIFO (First In First Out)

f_4				2		2		2		2			1	1			1		
f_3			1	1		1		1		0			0	0			0		
f_2		0	0	0		0		4		4			4	4			7		
f_1	7	7	7	7		3		3		3			3	2			2		
Strings →	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	×	×	×	×	✓	×	✓	×	✓	✓	×	✓	✓	×	×	✓	✓	×	✓	✓

Ans: Page faults = 10

Page Hits = 10

Q5

a) Deadlock Recovery

Deadlock recovery is the process by which the operating system **handles and resolves a deadlock** situation after it has been detected.
In a deadlock, a group of processes gets stuck, each waiting for resources held by the others, and none of them can proceed.
After detecting a deadlock, the system must take some action to recover and make progress.

Methods of Deadlock Recovery:

1. Process Termination:

- **Abort one or more processes** involved in the deadlock to break the cycle.
- Two approaches:
    - **Abort all deadlocked processes** (easier but harsh).
    - **Abort one process at a time** and check after each termination if the deadlock is resolved (less harsh but slower).

2. Resource Preemption:

- Temporarily **take resources away** from some processes and give them to others to break the deadlock.
- The process whose resources are taken is **rolled back** to a safe state and restarted later.

3. Process Rollback:

- Roll back one or more processes to a previous safe checkpoint (if available) and restart them.

b) Virtual Memory

Virtual Memory is a memory management technique used by operating systems to **make the computer appear to have more RAM than it actually does**.
It allows programs to use a large logical address space even if the physical memory (RAM) is small.
The operating system uses **secondary storage (like a hard disk or SSD)** as an extension of RAM by moving data between RAM and disk as needed.

Key Features of Virtual Memory:

- **Provides an illusion of large memory** to users and applications.
- **Enables multiprogramming** by allowing multiple programs to be loaded into memory at once.
- **Isolates programs** from one another for better protection and security.

Techniques Used:

- **Paging**: Divides memory into fixed-size blocks called pages.
- **Segmentation**: Divides memory into variable-size segments based on logical divisions like code, data, and stack.

Example:

If a computer has **4 GB RAM**, but a program needs **6 GB**, the operating system uses **virtual memory** to store the extra 2 GB on the hard disk, and moves data back and forth as required.

c) Android

Android is an **open-source operating system** primarily designed for **mobile devices** like smartphones, tablets, smart TVs, and wearable devices.
It is based on the **Linux kernel** and developed by **Google**.
Android provides a rich platform for developers to build powerful applications using the **Java**, **Kotlin**, or **C++** programming languages.

Key Features of Android:

- **Open Source**: Free to use and modify by manufacturers and developers.
- **Customizable Interface**: Users and manufacturers can modify the look and behavior.
- **Multitasking**: Allows running multiple apps at the same time.
- **Google Services Integration**: Access to apps like Gmail, Google Maps, Play Store.

Android Architecture:

- **Linux Kernel**: Provides low-level hardware interaction and security.
- **Libraries and Android Runtime**: Offers core functionalities like media playback and application execution.
- **Application Framework**: Supports high-level app features like activity management.
- **Applications**: User-installed apps like WhatsApp, Instagram, etc.

CNND DEC 2022 Answers

Q1

a) Explain about file attributes, file operations, and file types

File Attributes
File attributes are metadata associated with a file that describe its properties and behavior. Common attributes include:

1. **Name:** The file's name for identification.
2. **Type:** Indicates the file's format (e.g., .txt, .jpg).
3. **Location:** Path specifying where the file is stored.
4. **Size:** The file's size in bytes.
5. **Protection:** Permissions defining read, write, and execute rights.

File Operations
File operations enable users and programs to interact with files. Key operations include:

1. **Create:** Creates a new file.
2. **Open:** Makes a file available for access.
3. **Read:** Retrieves data from a file.
4. **Write:** Updates or adds data to a file.
5. **Delete:** Removes a file from storage.
6. **Seek:** Moves the file pointer to a specific location for reading or writing.

File Types
Files are categorized based on their content and usage. Common types include:

1. **Text Files (.txt):** Store plain text data.
2. **Binary Files (.bin):** Contain data in binary format, such as compiled programs.
3. **Executable Files (.exe):** Contain machine code for execution.
4. **Image Files (.jpg, .png):** Store graphic data.
5. **Audio Files (.mp3, .wav):** Hold sound data.
6. **Video Files (.mp4, .avi):** Contain multimedia content.
7. **Archive Files (.zip, .rar):** Store compressed files for easy transfer.