EXAM-2081/2082 CHAITRA/BAISHAKH

1.**What is operating system? Explain its functions in brie.**

An Operating System is a collection of programs that manages computer hardware and software resources. It provides a platform for application programs to run and ensures efficient utilization of system resources while maintaining stability and security.

Functions of an Operating System

1. **Process Management**
- Handles creation, scheduling, and termination of processes.
- Ensures fair CPU allocation among programs.

2. **Memory Management**
- Allocates and deallocates memory space to applications.
- Prevents memory conflicts and optimizes usage.

3. **File Management**
- Organizes, stores, and retrieves files on storage devices.
- Provides permissions and access control.

4. **Device Management**
- Controls input/output devices (keyboard, printer, disk drives).
- Uses device drivers to ensure smooth communication.

5. **Security & Access Control**
- Protects data and resources from unauthorized access.
- Implements authentication (passwords, biometrics) and encryption.

6. **User Interface**
- Provides command-line or graphical interface for user interaction.
- Makes the system user-friendly.

3. **What is producer consumer problem? Write the solution of producer consumer problem.**

The **Producer-Consumer Problem** (also known as the **Bounded-Buffer Problem**) is a classic synchronization challenge in operating systems. It involves two types of processes—the **Producer** and the **Consumer**—who share a common, fixed-size storage space called a **buffer**.

Solution of Producer–Consumer Problem

There are multiple ways to solve it. The most common are Semaphore-based solutions.

Semaphore Solution (Brief Code-like Explanation)

We use three semaphores:

• mutex→ ensures mutual exclusion while accessing buffer.

•empty→ counts empty slots in buffer.

• full → counts filled slots in buffer.

Producer Process.

```
wait(empty);     // check if buffer has empty slot
wait(mutex);     // lock buffer
  add item to buffer
signal(mutex);   // unlock buffer
signal(full);    // increase count of filled slots.
```

Consumer Process.

```
wait(full);      // check if buffer has filled slot
wait(mutex);     // lock buffer
  remove item from buffer
signal(mutex);   // unlock buffer
signal(empty);   // increase count of empty slots.
```

Consumer Process.

```
wait(full);      // check if buffer has filled slot
wait(mutex);     // lock buffer
  remove item from buffer
signal(mutex);   // unlock buffer
signal(empty);   // increase count of empty slots
```

Explanation of Solution

1. Producer waits if buffer is full, otherwise produces and adds item.
2. Consumer waits if buffer is empty, otherwise consumes item.
3. Mutex ensures only one process modifies buffer at a time.
4. Empty/Full counters synchronize producer and consumer actions.

4. Define the term paging. Explain different memory allocation strategies.
Paging is a memory management technique used by operating systems to manage how processes are stored in RAM. In paging, the physical memory is divided into fixed-size blocks called frames, and the logical memory (process) is divided into blocks of the same size called pages.

Memory Allocation Strategies.
Memory allocation strategies decide how processes are assigned to memory blocks. The main strategies are:

1. Single Contiguous Allocation
- Entire process is loaded into a single contiguous block of memory.
- Simple but inefficient, as it suffers from fragmentation.

2. Multiple Contiguous Allocation (Partitioning)
- Memory is divided into fixed or variable partitions.
- Each partition holds one process.
- Types:
- Fixed Partitioning: Memory divided into equal-sized partitions.
- Variable Partitioning: Partitions created dynamically based on process size.

3. Paging (Non-Contiguous Allocation)
- Memory divided into fixed-size frames; process divided into pages.
- Pages can be stored in any available frame.
- Eliminates external fragmentation.

4. Segmentation
- Memory divided into variable-sized segments based on logical divisions (code, data, stack).
- Provides logical view of memory but suffers from external fragmentation.

5. Dynamic Allocation Strategies (used in partitioning)
When variable partitions are used, the OS applies strategies to decide where to place a process:
- First Fit: Allocates the first available block large enough for the process.
- Best Fit: Allocates the smallest available block that fits the process (minimizes wasted space).
- Worst Fit: Allocates the largest available block (leaves big leftover space for future use).
- Next Fit: Similar to first fit but starts searching from the last allocated position

5. Differentiate between internal and external fragmentation.

Key	Internal Fragmentation	External Fragmentation
Definition	When there is a difference between required memory space vs allotted memory space, problem is termed as Internal Fragmentation.	When there are small and non-contiguous memory blocks which cannot be assigned to any process, the problem is termed as External Fragmentation.
Memory Block Size	Internal Fragmentation occurs when allotted memory blocks are of fixed size.	External Fragmentation occurs when allotted memory blocks are of varying size.
Occurrence	Internal Fragmentation occurs when a process needs more space than the size of allotted memory block or use less space.	External Fragmentation occurs when a process is removed from the **main memory**.
Solution	Best Fit Block Search is the solution for internal fragmentation.	Compaction is the solution for external fragmentation.
Process	Internal Fragmentation occurs when **Paging** is employed.	External Fragmentation occurs when **Segmentation** is employed.

.6Explain deadlock in OS. Explain necessary conditions for deadlock in OS.

A deadlock in an operating system is a situation where two or more processes are permanently blocked because each is waiting for resources held by the other. It occurs when four necessary conditions (mutual exclusion, hold and wait, no preemption, and circular wait) exist simultaneously.

Deadlock is a state in which a set of processes cannot proceed because each process is waiting for a resource that another process in the set is holding.

- Example: Process P1 holds Resource R1 and requests R2, while Process P2 holds Resource R2 and requests R1. Neither process can continue → deadlock.

Necessary Conditions for Deadlock (Coffman Conditions)

Deadlock can occur only if all four conditions hold true:

1. **Mutual Exclusion**
- At least one resource must be held in a non-shareable mode.
- Example: A printer can only be used by one process at a time.

2. **Hold and Wait**
- A process holding at least one resource is waiting to acquire additional resources held by other processes.
- Example: A process holds memory and waits for CPU.

3. **No Preemption**
- Resources cannot be forcibly taken away; they must be released voluntarily by the process holding them.
- Example: A process using a file must release it after completion.

4. **Circular Wait**
- A set of processes are waiting for resources in a circular chain.
- Example: P1 waits for R2, P2 waits for R3, and P3 waits for R1.

7.What is deadlock detection? Explain banker's algorithm with example.

Definition:

Deadlock detection is the process used by an operating system to identify whether a deadlock has occurred among processes competing for resource**s.**

Banker's Algorithm

Definition:

The Banker's Algorithm is a deadlock avoidance algorithm developed by Edsger Dijkstra.

- It works like a banker giving loans: resources are only allocated if the system can remain in a safe state.
- A safe state means there exists at least one sequence of process execution where all processes can finish without deadlock.

Steps of Banker's Algorithm

1. **Data Structures:**
- Available: Vector showing number of available resources of each type.
- Max: Matrix showing maximum demand of each process.
- Allocation: Matrix showing resources currently allocated to each process.
- Need: Matrix showing remaining resource needs (Need = Max – Allocation).

2. **Safety Check:**
- Find a process whose Need ≤ Available.
- If found, assume it finishes → release its allocated resources back to Available.
- Repeat until all processes finish (safe state) or no process can proceed (unsafe state).

3. **Resource Request:**
- When a process requests resources, the system checks if granting them keeps the system in a safe state.
- If yes → allocate.
- If no → process must wait.

Example

Suppose we have 3 resource types (A, B, C) and 3 processes (P1, P2, P3).

Available: A = 3, B = 2, C = 2.

Max Matrix:.

P1: 7 5 3

P2: 3 2 2

P3: 9 0 2

Allocation Matrix:.

P1: 0 1 0

P2: 2 0 0

P3: 3 0 2

Need Matrix (Max – Allocation):

P1: 7 4 3

P2: 1 2 2

P3: 6 0 0

Available = (3,2,2).

- P2's Need (1,2,2) ≤ Available (3,2,2) → P2 can finish.
- After finishing, Available = Available + Allocation(P2) = (3,2,2) + (2,0,0) = (5,2,2).

Now P1's Need (7,4,3) > Available (5,2,2) → cannot run.

- P3's Need (6,0,0) ≤ Available (5,2,2)? No → cannot run.

👉 System is unsafe → deadlock may occur.

8. **Explain different file allocation techniques in OS.**
fferent File Allocation Techniques
1. **Contiguous Allocation.**
 Pros: Very fast read/write speeds because the disk head moves minimally; supports both sequential and direct (random) access.
 Cons: Leads to external fragmentation (wasted gaps between files); difficult to grow a file if the next block is already taken by another file.
..Concept: Each file occupies a set of contiguous blocks on the disk.
- **Advantages**:
- Simple to implement.
- Fast access (sequential and direct).
- **Disadvantages:**
- Suffers from external fragmentation (free space scattered).
- Difficult to grow files (no guarantee of contiguous free blocks).
- Example: If a file needs 5 blocks, it is stored in blocks 10–14 continuously.

2. **Linked Allocation**
Concept: Each file is a linked list of disk blocks. Blocks may be scattered anywhere on disk, but each block contains a pointer to the next block.
 Pros: No external fragmentation; files can grow easily as long as there is any free block available.
 Cons: Only supports **sequential access** (to get to the 10th block, you must read the previous 9); if a single pointer is corrupted, the rest of the file is lost.
Advantages:
- No external fragmentation.
- Easy to grow files dynamically.

Disadvantages:
- Random access is slow (must follow pointers).
- Extra space needed for pointers.
- Example: File stored in blocks 5 → 11 → 2 → 20 (linked together).

3. **Indexed Allocation**
• Concept: Each file has an index block that contains pointers to all its disk blocks.
 Pros: Supports **direct access** (random access) efficiently; no external fragmentation.
 Cons: Wasted space for the index block itself, especially for very small files; for massive files, one index block might not be enough (requiring multilevel indexin
• **Advantages**:
- Supports both sequential and random access efficiently.
- No external fragmentation.

• **Disadvantages:**
- Overhead of maintaining index blocks.
- If file is very large, multiple index blocks may be needed.
- Example: Index block points to blocks [4, 7, 9, 12] where file data is stored.

9. Define cryptography. Explain different security attacks.

Definition

Cryptography is the science of securing information by converting it into an unreadable format using mathematical techniques, so that only authorized parties can access it.

Security Attacks in OS

Security attacks are generally classified into two main categories: **Passive Attacks** and **Active Attacks**.

1. Passive Attacks

In a passive attack, the intruder monitors or eavesdrops on the communication without altering the data. These are very difficult to detect because the system continues to function normally.

- **Release of Message Content:** The attacker simply reads a sensitive email or file that wasn't meant for them.
- **Traffic Analysis:** Even if the data is encrypted, the attacker observes the frequency and length of messages to guess the nature of the communication or identify the parties **involved.**

Examples:

- Eavesdropping: Listening to private communication.
- Traffic Analysis: Studying communication patterns to infer sensitive information.
- Impact: Hard to detect because no data is changed.

2. Active Attacks

In an active attack, the intruder disrupts the normal operation of the system or modifies the data. These are usually easier to detect but cause immediate damage.

- **Masquerade:** The attacker pretends to be a legitimate user to gain unauthorized access (e.g., using stolen login credentials).
- **Modification of Messages:** The attacker intercepts a message and alters it before it reaches the recipient (e.g., changing "Pay $100" to "Pay $10,000").
- **Replay Attack:** The attacker captures a valid data transmission (like a login request) and retransmits it later to trick the system into granting access again.
- **Denial of Service (DoS):** The attacker floods a system with so much traffic that it crashes or becomes unavailable to legitimate users

Examples:

- Masquerade Attack: Attacker pretends to be an authorized user.
- Replay Attack: Reusing captured data packets to gain access.
- Modification Attack: Altering messages during transmission.

10. Write short notes on.

a) Race Condition

Definition:

A race condition occurs when two or more processes access shared resources concurrently, and the final outcome depends on the order of execution.

Key Points:

- Happens in multi-threaded or multi-process systems.

- Leads to unpredictable results and errors.
- Common in critical sections (shared variables, files).
- Prevented using synchronization techniques (locks, semaphores, monitors).

Example:
Two processes updating a bank account balance at the same time → incorrect result if not synchronized.

💻 b) Multiprogramming

Definition:
Multiprogramming is the technique of running multiple programs simultaneously on a single processor by managing them in memory.
Key Points:

- Increases CPU utilization by keeping it busy.
- When one program waits for I/O, another program uses the CPU.
- Requires memory management and scheduling.
- Forms the basis of modern operating systems.

Example:
While editing a document, the system can also play music and download files in the background.

⚙️ c) System Call

Definition:
A system call is the interface between a user program and the operating system. It allows programs to request services from the OS.
Key Points:

- Provides controlled access to hardware and OS functions.
- Types of system calls:

1. Process Control (create, terminate processes).
2. File Management (open, read, write files).
3. Device Management (request or release devices).
4. Information Maintenance (get system time, attributes).
5. Communication (send/receive messages).

- Essential for program execution and resource management.

Example:
read and write system calls in **UNIX/Linux.**

BACK EXAM – 2081/2082 CHAITRA/BAISHAKH

1. What is operating system? What are functions of operating system? Explain.

An operating system is system software that works as an interface between the user and computer hardware. It manages hardware resources and provides a platform to run application programs

Functions of an Operating System

An OS performs several vital tasks behind the scenes to keep the system running smoothly:

- **Processor Management:** The OS decides which program gets to use the CPU (the computer's brain) and for how long. This is called **scheduling**, and it's why you can listen to music while typing a document.
- **Memory Management:** The OS manages the system's primary memory (RAM). It keeps track of every byte and decides which process gets how much memory. When you close a program, the OS "reclaims" that memory for other apps.
- **File Management:** It creates a structured system for storing, retrieving, and naming files. It acts like a digital librarian, keeping your folders and documents organized on your hard drive or SSD.
- **Device Management:** The OS communicates with your hardware (printers, keyboards, monitors) through special software called **drivers**. It handles the "input" and "output" so the computer knows how to talk to each device.
- **User Interface (UI):** This is the part you actually interact with. It can be a **GUI** (Graphical User Interface) with icons and windows like Windows 11 or macOS, or a **CLI** (Command Line Interface) where you type text commands.
- **Security:** The OS protects your data from unauthorized access through passwords and firewalls. It also ensures that one program cannot "crash" or interfere with another program's data.
- **Error Detection:** It constantly monitors the system for hardware or software problems. If something goes wrong, the OS tries to fix it or alerts you with an error message instead of letting the whole system crash.

2. . Define process and thread. Describe different process states with necessary diagram.

1. Process vs. Thread

- **Process:** A process is a **program in execution**. When you double-click an app, the OS loads its code into memory, assigns it resources (RAM, files, etc.), and starts running it. It is considered a "heavyweight" entity because each process has its own dedicated memory space.
- **Thread:** A thread is a **subset of a process**, often called a "lightweight process." It is the smallest unit of execution that can be scheduled by the OS. Multiple threads of the same process share the same memory and resources but execute different tasks simultaneously (like a browser process having one thread for rendering the page and another for downloading a file.

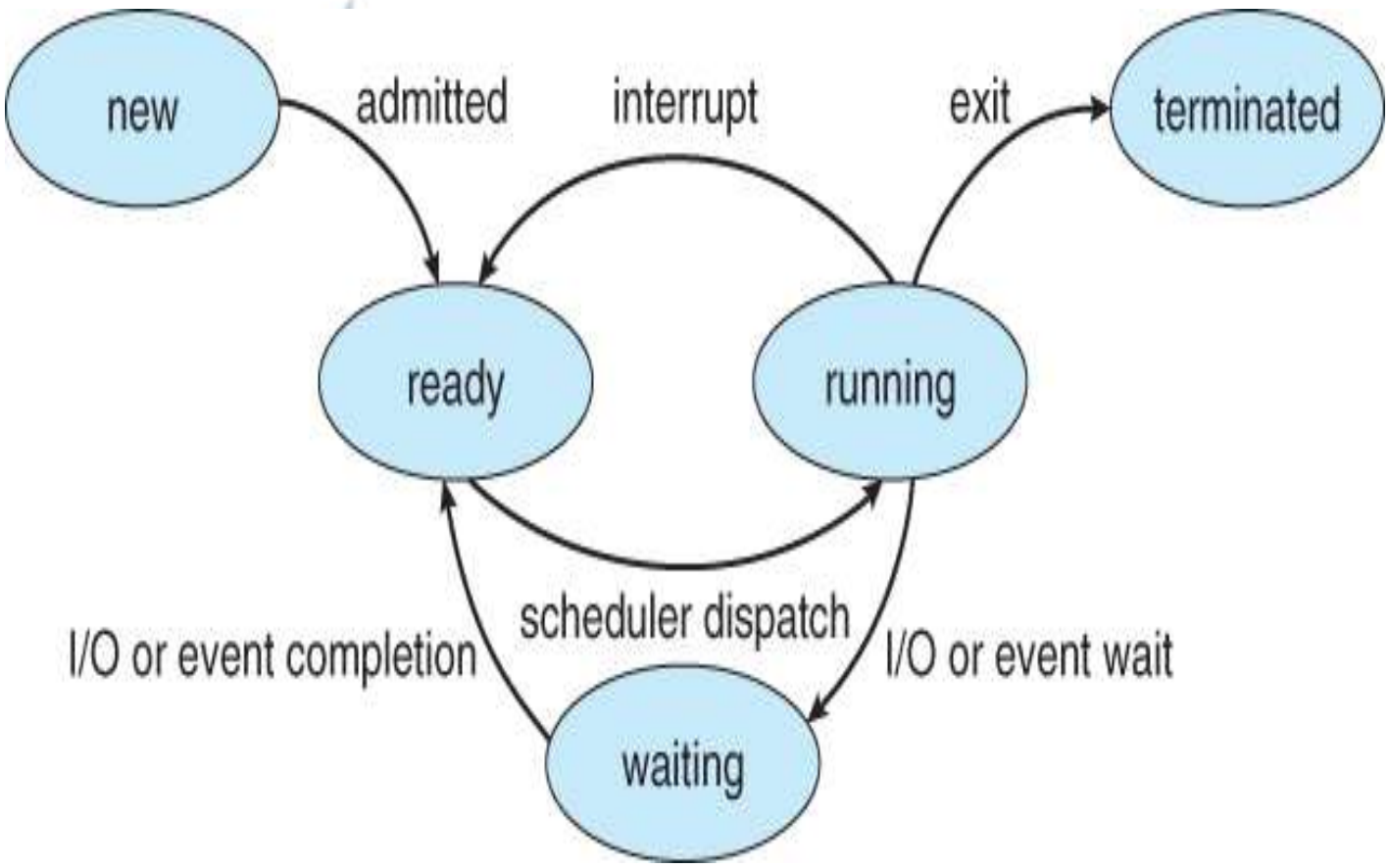
Process States (Life Cycle)

As a process moves from being a file on your disk to finished work, it passes through several distinct states. This is often represented by the **Process State Transition Diagram**.

The Five Primary States:

1. **New:** The process is being created but hasn't been loaded into the main memory yet.
2. **Ready:** The process is loaded into the main memory and is waiting for the CPU to become available. Many processes can be in the "Ready Queue" at once.
3. **Running:** The CPU is currently executing the instructions of this process. In a single-core system, only one process can be in the "Running" state at a time.
4. **Waiting (Blocked):** If a process needs to wait for an event (like a user clicking a button or a file being read from the disk), it moves here. It cannot use the CPU until that event is finished.
5. **Terminated (Exit):** Once the process completes its execution or is killed by the OS, it moves to this final state where its resources are released.

NECESSARY DIAGRAM.



3. What is inter-process communication? Explain Peterson's solution to mutual exclusion with busy waiting.

Inter-Process Communication is a way for processes to exchange data and coordinate actions. It helps them work together smoothly by sharing information or sending messages. Common methods include shared memory, message passing, pipes, and semaphores.

Peterson's Solution to Mutual Exclusion

Peterson's Solution is a classic software-based algorithm used to solve the **Critical Section Problem** for exactly two processes.[7] It ensures that only one process enters its critical section at a time (**Mutual Exclusion**) using two shared variables:[8]

1. **flag[2]:** A boolean array where flag[i] = true means Process [9]i wants to enter its critical section.[10]
2. **turn:** An integer that indicates whose "turn" it is to enter.[11]

The Logic

When Process i wants to enter its critical section:

1. It sets flag[i] = true.[12]
2. It sets turn = j (giving the other process, [13]j, a chance to go first).[14]
3. **Busy Waiting:** It then enters a while loop, constantly checking if the other process wants to enter AND if it is the other process's turn.[15]

Note: "Busy Waiting" means the process stays in a loop, consuming CPU cycles just to check the condition, rather than going to sleep.[16]

4. Once the condition is false, it enters the **Critical Section**.
5. After finishing, it sets flag[i] = false to let the other process in.

Pseudo-code for Process P_i

```
do {
   flag[i] = true;          // I am interested
   turn = j;                // You can go first
   while (flag[j] && turn == j); // Busy wait (spin)

   // --- CRITICAL SECTION ---
   // Access shared data here

   flag[i] = false;         // I am done

   // --- REMAINDER SECTION ---
} while (true);
```

Why it works

- **Mutual Exclusion:** If both try to enter, turn can only be [17]0 or [18]1, never both.[19] One process will always be stuck in the while loop while the other proceeds.
- **Progress:** A process not interested in the critical section (flag == false) will never block the other.[20]
- **Bounded Waiting:** Since the processes "give away" the turn, one process won't be stuck waiting forever while the other enters repeatedly

4.

5. What are the necessary conditions for deadlock? Differentiate between preemptive and non-preemptive scheduling.

Necessary Conditions for Deadlock (4 conditions)
Deadlock occurs when processes are stuck waiting for resources forever. The four necessary conditions are:
1. Mutual Exclusion – At least one resource must be held in a non-shareable mode (only one process can use it at a time).
2. Hold and Wait – A process holding at least one resource is waiting to acquire additional resources.
3. No Preemption – Resources cannot be forcibly taken away; they are released only voluntarily by the process.
4. Circular Wait – A set of processes are waiting for resources in a circular chain (P1 waits for P2, P2 waits for P3, …, Pn waits for P1).

Preemptive vs. Non-Preemptive Scheduling – Key Differences

Feature	Preemptive Scheduling	Non-Preemptive Scheduling
Basic Idea	OS **can interrupt** a running process before completion to give CPU to another process.	Once a process starts, it runs **until it finishes or voluntarily blocks** (e.g., for I/O).
Who Controls CPU	**OS** has full control.	**Process** keeps control until it releases CPU.
Context Switching	**Frequent** – happens when a process is preempted.	**Infrequent** – only when process terminates or blocks.
Overhead	**High** – due to frequent context switches and scheduling decisions.	**Low** – less context switching.
Responsiveness	**High** – good for interactive/time-sharing systems; short jobs get quick response.	**Low** – a long CPU burst can block all others.
Starvation Risk	Possible – low-priority jobs may wait indefinitely if higher-priority jobs keep arriving.	Possible – if a process has a very long CPU burst.
Scheduling Algorithms	• Round Robin (RR) • Shortest Remaining Time First (SRTF) • Preemptive Priority	• First Come First Serve (FCFS) • Non-Preemptive Shortest Job First (SJF) • Non-Preemptive Priority

Feature	Preemptive Scheduling	Non-Preemptive Scheduling
	• Multilevel Queue (with preemption)	
Flexibility	**More flexible** – OS can adapt to changing situations (e.g., new high-priority process).	**Less flexible** – scheduling decision made only at process start or end.

6.Explain about segmentation with its importance and drawbacks.

Segmentation is a memory management technique in which the computer's memory is divided into different segments based on the logical divisions of a program.

Unlike **Paging**, which divides memory into fixed-size blocks (physical view), segmentation divides memory into variable-sized chunks that correspond to how a programmer views a program (logical view). For example, a program is naturally divided into a main function, stack, library functions, and data arrays; each of these becomes a "segment."

How it Works

Each segment has a **name** (or number) and a **length**. To keep track of these, the CPU uses a **Segment Table**, which contains:

1. **Base:** The starting physical address of the segment in memory.
2. **Limit:** The length or size of that specific segment.

When the CPU generates a logical address, it consists of a pair: <segment-number, offset>. The hardware checks that the offset is less than the limit before adding it to the base address to find the actual location in RAM.

Importance of Segmentation

- **Logical Organization:** It mirrors the way a programmer thinks. Data, code, and stack are kept in separate areas, making the program's structure clear.
- **Protection:** You can easily assign permissions to specific segments. For instance, the "Code Segment" can be marked as *Read-Only*, while the "Data Segment" is *Read-Write*.
- **Sharing:** It is very efficient for sharing common code. For example, multiple users running the same text editor can share one single "Library Segment" in memory.
- **Variable Size:** Segments grow and shrink according to the needs of the program, which is more flexible than fixed pages.

Drawbacks of Segmentation

- **External Fragmentation:** As processes are loaded and removed, memory becomes broken into small, unusable gaps between segments. If a new segment is larger than any available gap, it cannot be loaded, even if the total free space is sufficient.
- **Complexity:** Managing a segment table with variable lengths is more complex for the operating system compared to the simple, fixed-size math used in paging.
- **Difficulty in Allocation:** Finding a contiguous hole in physical memory that fits a specific segment size can be difficult (using algorithms like First-fit or Best-fit).
- **Compaction Required:** To fix external fragmentation, the OS sometimes has to "shuffle" segments around in memory to create one large free block, which consumes significant CPU time.

7.What is page fault? Describe Not Recently Used Page replacement algorithm with suitable example.

Definition:

A page fault occurs when a program tries to access a page (block of memory) that is not currently present in the main memory (RAM). The operating system must then fetch the required page from secondary storage (disk) into RAM.

Not Recently Used (NRU) Algorithm

When the RAM is full and the OS needs to bring in a new page, it has to "kick out" an old one. The **NRU algorithm** is like a manager who fires the least productive employees first.

To decide who to kick out, the OS uses two "status bits" for every page:

1. **R (Referenced):** Was this page read or used recently? (1 = Yes, 0 = No)
2. **M (Modified):** Was the data in this page changed (edited)? (1 = Yes, 0 = No)

The 4 Priority Classes

The OS puts every page into one of these four categories:

- **Class 0:** $R=0, M=0$ (Not used, Not changed). **The "Lazy" Page.** Best one to kick out.
- **Class 1:** $R=0, M=1$ (Not used recently, but was changed). This usually happens when the R bit is cleared by a timer.
- **Class 2:** $R=1, M=0$ (Used recently, but "Clean"). It's being used, but hasn't been edited.
- **Class 3:** $R=1, M=1$ (Used and Changed). **The "Busy" Page.** Most important; don't kick this one out unless you have to.

The Rule: When a Page Fault happens, the OS picks a random page from the **lowest-numbered class** to remove.

3. Example (Simple Scenario)

Suppose your RAM is full with 4 pages, and you need to bring in **Page P5**.

Page	R bit	M bit	Class	Action
P1	1	1	3	High Priority - Keep it.
P2	0	1	1	Medium Priority - Keep it.
P3	**0**	**0**	**0**	**Lowest Priority - EVICT!**
P4	1	0	2	High Priority - Keep it.

8. Explain contiguous and linked list allocation in file system.

. **Contiguous Allocation**

Definition:

In contiguous allocation, each file is stored in a set of consecutive blocks on the disk.

Explanation:

- The directory entry maintains the starting block and the length of the file.
- Access is fast because blocks are sequentially located.
- Problem: External fragmentation occurs when free space is scattered.
- Difficult to grow files dynamically because adjacent blocks may not be free.

Suitable Example:

Suppose a file requires 5 blocks. In contiguous allocation, it will be stored in blocks 10–14 consecutively. Accessing block 10 automatically leads to sequential reading of 11, 12, 13, and 14.

2. Linked List Allocation

Definition:

In linked list allocation, each file is stored in scattered blocks across the disk, with each block containing a pointer to the next block.

Explanation:

- Directory entry stores the starting block.
- Each block has a pointer to the next block, forming a linked list.
- Eliminates external fragmentation.
- Slower access because pointers must be followed sequentially.
- Random access is difficult.

Suitable Example:

Suppose a file requires 5 blocks. In linked allocation, it may be stored in blocks 10 → 25 → 3 → 40 → 18, with each block pointing to the next. The directory entry points to block 10, and the chain continues until the file ends.

9. Compare memory mapped IO and 10 mapped IO. Explain first come first service disk scheduling algorithm with suitable example.

In simple terms, **Memory Mapped I/O** treats hardware devices like they are just addresses in RAM, while **I/O Mapped I/O** gives hardware its own "special VIP lane."

Feature	Memory Mapped I/O	I/O Mapped I/O (Isolated)
Address Space	Same address space for memory and I/O devices.	Separate address spaces for memory and I/O.
Instructions	Uses standard instructions (like LOAD, STORE) to talk to hardware.	Uses special instructions (like IN, OUT) to talk to hardware.
Simplicity	Easier to program because hardware looks like memory.	Slightly more complex as it requires special CPU instructions.
Efficiency	Can use a lot of "memory addresses," leaving less room for actual RAM.	Saves memory addresses for RAM by keeping I/O separate.
Hardware	Simpler CPU logic.	Requires a special "I/O Read/Write" line in the hardware.

Explain first come first service disk scheduling

In simple terms, **FCFS (First Come First Serve)** is the most basic disk scheduling algorithm. It works exactly like a queue at a movie theater—whoever asks for a disk track first gets served first. The disk arm doesn't try to be "smart" or find the closest track; it simply follows the order of arrival.

How it Works

- The requests are addressed in the **order** they are received in the disk queue.
- It is a **non-preemptive** algorithm.
- There is no logic to optimize the movement of the disk head (the "needle").

Suitable Example

Imagine a disk with tracks ranging from **0 to 199**.

- **Current position of the Disk Head:** 50
- **Request Queue (in order):** 82, 170, 43, 140, 24, 16, 190

To find the efficiency, we calculate the **Total Head Movement (Total Seek Time)**:

1. **From 50 to 82:** $|82 - 50| = 32$
2. **From 82 to 170:** $|170 - 82| = 88$
3. **From 170 to 43:** $|43 - 170| = 127$
4. **From 43 to 140:** $|140 - 43| = 97$
5. **From 140 to 24:** $|24 - 140| = 116$
6. **From 24 to 16:** $|16 - 24| = 8$
7. **From 16 to 190:** $|190 - 16| = 174$

Total Head Movement = $32 + 88 + 127 + 97 + 116 + 8 + 174 = \mathbf{642 \text{ cylinders}}$

10. Write short notes on:

a. **Banker's Algorithm**

• A deadlock avoidance method that allocates resources only if the system stays in a safe state.
Each process declares its maximum need; the OS simulates allocation to check safety.
If a safe sequence exists, resources are granted—otherwise, the process waits.

• Explanation: It works like a banker giving loans — resources are allocated only if the system can remain in a safe state.

- Process:
- Each process declares its maximum resource need.
- The OS checks if resources can be safely allocated without leading to deadlock.
- If yes → resources are granted; if not → process waits.
- Importance: Prevents deadlock by ensuring that at least one safe sequence of process execution exists.

2. Types of Operating System

- Batch OS: Executes jobs in batches without user interaction.
- Time-Sharing OS: Allows multiple users to share system resources simultaneously.
- Real-Time OS: Provides immediate response, used in critical systems (aircraft, medical devices).
- Distributed OS: Manages multiple computers as a single system.
- Network OS: Provides services to computers connected in a network.
- Embedded OS: Designed for specialized devices like mobiles, appliances, and IoT.

c. Memory Hierarchy

Memory hierarchy organizes storage into levels by speed, cost, and capacity.
From fastest/smallest to slowest/largest: registers → cache → RAM → SSD/HDD → archival.Keeping hot data in faster levels reduces access time and improves overall performance

- Structure (Top → Bottom):

1. Registers – fastest, smallest.
2. Cache Memory – very fast, stores frequently used data.
3. Main Memory (RAM) – moderate speed, larger capacity.
4. Secondary Storage (Hard Disk/SSD) – slower, very large capacity.
5. Tertiary Storage (Magnetic tapes, optical disks) – slowest, used for backups.

- Importance: Provides a balance between speed and cost. Frequently used data is kept in faster memory, while bulk data is stored in slower, cheaper memory.

4. FIFO Scheduling

FIFO (First In First Out) scheduling executes processes in the **order of their arrival**.
The process that comes first gets the CPU first.
It is simple and easy to implement.
However, it may cause **long waiting time** for short processes.
FIFO is also known as **FCFS (First Come First Serve)** scheduling.
First-In, First-Out (FCFS) scheduling serves processes or disk requests in the order they arrive.

- Simple and fair, but does not consider burst time or priority.
- In CPU scheduling: the process that arrives first is executed first.

REGULAR EXAM-2081 JESTHA/ASHADH

1. a. Define operating system. Explain the function of OS.

An Operating System (OS) is system software that acts as an interface between the user and computer hardware. It manages hardware resources (CPU, memory, I/O devices) and provides services to application programs.

Functions of Operating System

1. Process Management
- Creates, schedules, and terminates processes.
- Allocates CPU time fairly among processes.

2. Memory Management
- Allocates and deallocates memory to processes.
- Prevents memory conflicts and optimizes usage.

3. File Management
- Organizes, stores, and retrieves files on storage devices.
- Provides permissions and access control.

4. Device Management
- Controls input/output devices (keyboard, printer, disk drives).
- Uses device drivers for communication.

5. Security & Protection
- Protects data and resources from unauthorized access.
- Implements authentication (passwords, biometrics).

6. User Interface
- Provides command-line or graphical interface for user interaction.
- Makes the system user-friendly.

b. What are the types of operating system? Explain.

1. b. Types of Operating Systems

Based on how they handle tasks and users, operating systems are classified into several types:

1. Batch Operating System

In this type, the user does not interact with the computer directly. Instead, the user prepares their "job" (program + data) and submits it to a computer operator. The operator groups similar jobs into **batches** and runs them together to speed up processing.

- **Example:** Payroll systems, Bank statements.

2. Multiprogramming Operating System

The goal here is to keep the CPU busy at all times. Multiple programs are loaded into the main memory (RAM). If the current program stops to wait for an I/O task (like waiting for a printer), the OS immediately switches the CPU to another program.

3. Time-Sharing (Multitasking) Operating System

This is a logical extension of multiprogramming. Each task is given a specific "slice" of CPU time, called a **Time Quantum**. The CPU switches between tasks so rapidly that the user feels like all programs are running at the same time.

- **Example:** Windows, macOS, Linux.

4. Real-Time Operating System (RTOS)

These are used in environments where **timing is everything**. The OS must process data and respond within a very strict time limit (deadline). If it fails to meet the deadline, the system fails.

- **Hard Real-Time:** Missing a deadline is a total system failure (e.g., Satellite control, Airbag systems).
- **Soft Real-Time:** Missing a deadline is bad but not a disaster (e.g., Video streaming).

5. Distributed Operating System

This type connects multiple independent computers and makes them appear as a single system to the user. It allows users to access resources (like files or processing power) located on different machines across a network.

- **Example:** Large research networks..

2. a. Define process control block (PCB). Differentiate between The Process Control
Block (PCB) is a data structure used by the Operating System to store all the information about a specific process. You can think of it as the "Identity Card" or "Data Sheet" of a process process and program.

Comparison: Program vs. Process

Basis	**Program**	**Process**
Basic Definition	A static file containing a set of instructions.	A program in a state of execution.
Nature	**Passive Entity:** It just sits there.	**Active Entity:** It performs tasks.
Lifespan	Long-lasting (exists until deleted from disk).	Temporary (exists only while running).
Address Space	No address space assigned.	Has its own **Logical Address Space**.
Components	Contains code and static data.	Contains Stack, Heap, Data, and Text segments.
Resources	Occupies only disk space.	Occupies CPU, RAM, and I/O resources.
OS Awareness	OS is not aware of it until it's loaded.	OS tracks it via a **Process Control Block (PCB)**.
Changes	The file content stays the same.	The state, variables, and registers change constantly.
Relationship	One program can create many processes.	A process is an instance of a specific program.

b.

3.a. What is memory management? Explain different types of memory allocation with example.

Memory Management is the process of controlling and coordinating a computer's main memory (RAM). It ensures that the OS keeps track of every memory byte—whether it is allocated to a process or is free.

Types of Memory Allocation

There are two main ways the OS assigns memory to processes:

1. Contiguous Memory Allocation In this method, each process is contained in a single, unbroken block of memory.

- **Fixed Partitioning:** RAM is divided into fixed-size slots. Even if a process is small, it takes up the whole slot.
- **Variable Partitioning:** The OS allocates exactly as much space as the process needs.
- *Example:* If Process A needs 50MB, the OS finds a 50MB continuous gap and puts it there.

2. Non-Contiguous Memory Allocation The process's data is allowed to be scattered in different parts of the RAM. This is much more efficient.

- **Paging:** Breaking the process into fixed-size "pages" and the RAM into "frames."
- **Segmentation:** Breaking the process into logical parts (code, data, stack).
- *Example:* Process B is 10MB. The OS puts 4MB in one corner of the RAM and 6MB in another corner.

b. Describe multiprogramming. Differentiate between internal and external fragmentation.

Multiprogramming is an OS technique where multiple processes are loaded into main memory simultaneously and the CPU switches between them to maximize utilization. When one process waits for I/O, another gets CPU time.

Internal vs. External Fragmentation

S.No	Basis of Comparison	Internal Fragmentation	External Fragmentation
1	**Definition**	Occurs when memory blocks are of fixed size and the process is smaller than the block.	Occurs when total free memory is enough for a process, but it isn't contiguous (side-by-side).
2	**Location**	The wasted space is **inside** the allocated memory partition.	The wasted space is **between** different allocated partitions.
3	**Main Cause**	It happens due to **Fixed Partitioning** (Static allocation).	It happens due to **Variable Partitioning** (Dynamic allocation).
4	**Process Size**	The process is smaller than the memory hole provided.	The process is larger than any single free hole, but smaller than the total free space.
5	**Occurrence**	Occurs when the OS assigns a process to a pre-defined fixed slot.	Occurs when processes of various sizes are loaded and removed over time, leaving "gaps."
6	**Impact**	Memory is wasted even though it is technically "assigned" to a process.	Memory is wasted because it is too scattered to be assigned to any new process.
7	**Solution**	Can be solved by using **Variable Partitioning** or **Paging**.	Can be solved by using **Compaction** (shuffling) or **Segmentation**.
8	**Predictability**	It is easier to predict if you know the block sizes.	It is unpredictable as it depends on the arrival and departure of processes.

4. a. What is file? Explain about file allocation methods.

A file is basically a named collection of data stored on your hard disk, SSD, pen drive, etc. It can be anything – text document, photo, video, song, program, PDF, whatever.

File allocation methods (how the OS gives space on disk to a file)

1. **Contiguous Allocation** The whole file is stored in one continuous block of space on the disk (like consecutive seats in a row). The directory just remembers the starting block number and how many blocks the file uses. **Good points**: super fast to read or write (head doesn't have to jump around),

very simple. **Bad points**: lots of external fragmentation (small free spaces get scattered, hard to find big continuous space for new or growing files). You also need to know file size in advance sometimes.

2. **Linked Allocation** File blocks can be anywhere on the disk. Each block has a pointer telling where the next block is (like a chain or linked list). Directory only keeps the address of the first block. **Good points**: no external fragmentation, easy to make file bigger (just add new block at the end of chain). **Bad points**: random access is very slow (have to start from first block and follow pointers every time). If one pointer gets corrupted, the rest of the file is lost.
3. **Indexed Allocation** Every file gets its own index block (or inode). This index block has a list of pointers to all the data blocks of the file. Directory points to this index block. **Good points**: random access is fast (just look at index and jump directly), almost no external fragmentation, handles very large files well (using multi-level indexing). **Bad points**: small files waste space because of the extra index block.

4(b) Terms

i. Disk Formatting

- The process of preparing a disk for use by an operating system.
- It creates the file system structure (like FAT, NTFS, ext4).
- Formatting erases all existing data and organizes the disk into sectors and tracks.
- Example: Formatting a USB drive before storing files.

ii. Directory System

- A directory is a special type of file that contains information about other files.
- It organizes files into a hierarchical structure (folders and subfolders).
- Functions:
- Stores file names, attributes, and locations.
- Provides path for accessing files (e.g.,).
- Example: Windows Explorer showing folders and files.

iii. Disk Arm Scheduling

- Refers to algorithms used to schedule disk I/O requests to minimize seek time (movement of disk arm).
- Common algorithms:
- FCFS (First Come First Serve): Requests served in order of arrival.
- SSTF (Shortest Seek Time First): Chooses request closest to current head position.
- SCAN (Elevator Algorithm): Disk arm moves back and forth across disk serving requests.
- C-SCAN (Circular SCAN): Arm moves in one direction only, then jumps back.
- Example: If requests are at tracks 10, 20, 30, SSTF will serve the nearest track first.

5.a.Define deadlock. Explain deadlock handling strategies.

Definition

A deadlock is a situation in an operating system where a set of processes are permanently blocked because each process is waiting for a resource that another process in the set is holding.

Deadlock Handling Strategies

Operating systems use different approaches to deal with deadlocks. These are:

1. **Deadlock Prevention**
   - Strategy: Ensure that at least one of the four necessary conditions (mutual exclusion, hold and wait, no preemption, circular wait) never occurs.
   - Techniques:
     - Avoid hold and wait → require processes to request all resources at once.
     - Allow preemption → forcibly take resources from processes.
     - Break circular wait → impose ordering of resource requests.
   - **Drawback:** May lead to low resource utilization and reduced system efficiency.
2. **Deadlock Avoidance**
   - Strategy: Dynamically examine resource allocation requests and decide whether granting them could lead to deadlock.
   - Technique: **Banker's Algorithm** (ensures system stays in a safe state).
   - **Drawback:** Requires advance knowledge of maximum resource needs of each process.
3. **Deadlock Detection and Recovery**
   - Strategy: Allow deadlock to occur, then detect and fix it.
   - Detection: Use resource allocation graphs or algorithms to check for cycles.
   - Recovery:
     - Terminate one or more processes involved in deadlock.
     - Preempt resources from processes.
   - **Drawback:** Recovery can be costly and may lead to loss of work

5B.

6.a. What is security attack? Explain active and passive attacks.

A security attack is any malicious attempt to compromise the confidentiality, integrity, or availability of computer systems, networks, or data. In simpler terms, it's when someone tries to break into, damage, or steal from a computer system without permission**.**

. **Passive Attacks.**

Definition:

Passive attacks occur when an attacker only monitors or observes communication without altering the data.

Goal:

- To steal information secretly.
- Compromise confidentiality of data.

Characteristics:

- No modification of data.
- Very difficult to detect because communication appears normal.
- Prevention is more important than detection.

Examples:

- Eavesdropping: Listening to private communication (like intercepting emails or phone calls).
- Traffic Analysis: Studying communication patterns (who talks to whom, frequency, size of messages) to infer sensitive information.

Impact:

- Information leakage.
- Loss of privacy.
- Can be used for planning further active attacks.

2. Active Attacks

Definition:

Active attacks occur when an attacker modifies, disrupts, or injects data into communication.

Goal:

- To cause damage, gain unauthorized access, or disrupt services.
- Compromise integrity and availability of data.

Characteristics:

- Involves modification of data streams.
- Easier to detect but more harmful.
- Requires strong defense mechanisms.

Examples:

- Masquerade Attack: Attacker pretends to be an authorized user.
- Replay Attack: Reusing captured data packets to gain access.
- Modification Attack: Altering messages during transmission.
- Denial of Service (DoS): Flooding a system with requests to make it unavailable.

Impact:

- Data corruption or loss.
- System downtime.
- Unauthorized access to resources.

b. Write short notes on:

i. FIFO (First-In, First-Out) Page Replacement

FIFO is the simplest page replacement algorithm. It follows the logic of a queue: "The oldest page in the RAM is the first one to be kicked out."

- **How it works:** The Operating System keeps track of all pages in a queue. When a **Page Fault** occurs and the RAM is full, the OS removes the page that has been in memory the longest (at the "head" of the queue).
- **Pros:** * Extremely easy to understand and program.
  - Low overhead (no complex math required).
- **Cons:**
  - **Inefficient:** It might kick out a page that is used very frequently just because it arrived early.
  - **Belady's Anomaly:** This is a famous weirdness where giving the system *more* RAM (more frames) can actually result in *more* page faults.

ii. Paging

Paging is a memory management scheme that allows the physical address space of a process to be **non-contiguous**. It solves the problem of "External Fragmentation."

- **The Logic:** 1. **Logical Memory (Process):** Divided into fixed-size blocks called **Pages**. 2. **Physical Memory (RAM):** Divided into fixed-size blocks called **Frames**.
- **How it works:** When a process needs to run, its pages are loaded into any available empty frames in the RAM. These frames don't have to be side-by-side.
- **Page Table:** Since the pages are scattered, the OS uses a "map" called a **Page Table** to keep track of which page is in which frame.
- **Advantages:**
  - No **External Fragmentation** (every tiny hole in RAM can be used).
  - Simplifies memory allocation.
- **Disadvantages:**
  - Can cause **Internal Fragmentation** (if the last page of a process doesn't perfectly fill a frame).
  - Requires extra memory for the Page Table itself.

BACK EXAM-2080, MAGH/PHAGUN.

1. **What is operating system? Explain types of operating system in detail.**

perating System (OS) is the most important software that runs on a computer. It acts as a "middleman" or bridge between the computer hardware and the user. Without an OS, you would have to write complex code just to tell the computer to move the mouse or save a file.

Types of Operating Systems in Detail

Here are the main types of OS, categorized by how they process tasks and handle users:

1. Batch Operating System

In the early days, users didn't interact with the computer directly. They would prepare their "jobs" (on punch cards) and give them to a human operator. The operator would group similar jobs into **batches** to speed up processing.

- **Pros:** Shifts the burden of management from the user to the operator.
- **Cons:** No direct interaction; if one job fails, the whole batch might be delayed.
- **Example:** Payroll systems or bank statements.

2. Multiprogramming Operating System

The main goal here is to keep the CPU busy. In this system, multiple programs are loaded into the RAM at once. If the "Active" program needs to wait for something (like a printer or a user to type), the OS immediately switches the CPU to another program.

- **Key Idea:** The CPU is never sitting idle if there is work to do.

3. Time-Sharing (Multitasking) Operating System

This is what we use today on our laptops and phones. The OS gives each task a tiny "slice" of time (called a **Time Quantum**). The CPU switches between tasks so fast (in milliseconds) that it feels like all programs are running at the same time.

- **Example:** Windows, macOS, Linux.

4. Real-Time Operating System (RTOS)

These are used when timing is critical. In an RTOS, the "Response Time" must be guaranteed. If a task isn't finished within a specific deadline, the whole system could fail.

- **Hard Real-Time:** Missing a deadline is a disaster (e.g., Airbags, Missile control).
- **Soft Real-Time:** Missing a deadline is bad but not fatal (e.g., Video streaming).

5. Distributed Operating System

This system connects a group of independent computers and makes them appear as one single computer to the user. It distributes the workload across different machines.

- **Example:** Large-scale research grids or telecommunication networks.

2.

3. Using Banker's algorithm, explain if the state is in deadlock state or in safe state.

Banker's algorithm is a method used in operating systems to avoid deadlock. It works like a bank giving loans, where resources are given only if the system can still remain in a safe state. The algorithm checks whether the current allocation of resources can allow all processes to finish one by one without getting stuck.

1. The Safe State

A system is in a **Safe State** if the Operating System can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock. This "order" is called the **Safe Sequence**.

- **How to identify it:** If you can find at least one sequence (e.g., $P_2 \rightarrow P_1 \rightarrow P_3$) where every process can finish its work and release its resources for the next one, the system is safe.
- **Analogy:** It's like a bank that has enough cash to satisfy the withdrawal needs of its customers if they don't all come at the exact same second.

2. The Unsafe State

An **Unsafe State** is not necessarily a deadlock, but it is a "danger zone." In this state, the OS cannot guarantee that all processes will finish. It only takes one specific set of requests from the processes to turn an unsafe state into a deadlock.

- **The Rule:** All Deadlock states are Unsafe, but not all Unsafe states are Deadlocks.

3. The Deadlock State

A system is in a **Deadlock State** when a group of processes are stuck in a circular loop, waiting for resources held by each other, and none of them can move forward.

How to identify a Deadlock (The 4 Conditions):

For a state to be a Deadlock, all four of these must be true:

1. **Mutual Exclusion:** Only one process can use a resource at a time.
2. **Hold and Wait:** A process is holding at least one resource and waiting for another.
3. **No Preemption:** Resources cannot be forcibly taken from a process.
4. **Circular Wait:** Process A waits for B, B waits for C, and C waits for A

4. What is segmentation? Why is it important? Write its advantages.

Segmentation is a memory management technique in which a process is divided into different logical segments such as code, data, and stack. Each segment can vary in size depending on the requirements of the process. Unlike paging, segmentation divides memory based on the logical structure of the program rather than fixed-size blocks.

Why is Segmentation Important?

- Marketing Context:
- Helps businesses understand customer needs better.
- Allows targeted strategies for different groups.
- Improves efficiency in resource allocation.
- Memory Management Context:
- Provides logical division of programs.
- Simplifies handling of code, data, and stack separately.
- Supports protection and sharing of segments.

Advantages of Segmentation

Marketing Segmentation Advantages:

1. Better customer satisfaction through tailored products/services.
2. Efficient use of marketing resources.
3. Competitive advantage by focusing on profitable segments.
4. Easier product positioning and differentiation.
5. Helps in strategic planning and forecasting.

5. What is thread? Differentiate between thread and process.

A thread is the smallest unit of CPU execution within a process.

It represents a single sequence of instructions that can run independently.

Threads share the same resources of a process but execute concurrently.

Differences: Process vs. Thread

S.No	Basis of Comparison	Process	Thread
1	**Definition**	An executing instance of a program.	A small unit of execution within a process.
2	**Nature**	Considered a **"Heavyweight"** entity.	Considered a **"Lightweight"** entity.
3	**Memory Space**	Each process has its own independent address space.	Threads share the address space of the parent process.
4	**Creation Time**	Takes more time to create because it needs its own memory.	Takes much less time to create.
5	**Context Switching**	Switching between processes is slow (high overhead).	Switching between threads is very fast (low overhead).
6	**Communication**	Requires **IPC** (Inter-Process Communication) like pipes or signals.	Can communicate directly as they share the same memory variables.
7	**Dependency**	Processes are independent; one crashing won't affect others.	Threads are dependent; if one thread crashes, the whole process may die.
8	**System Resources**	Consumes more resources (more RAM and CPU cycles).	Consumes significantly fewer resources.

6. Define file management. Explain file structure and file types.

File Management:

File management means handling files in a computer system. It includes creating, saving, naming, moving, and deleting files so that data is organized and easy to find. Basically, it's how the operating system helps us manage our documents, images, and programs.

File Structure:

File structure is the logical arrangement of data inside a file. It decides how information is stored, accessed, and updated.

- **Types of File Structures:**
- **Sequential File Structure:**
- Data stored one after another in order.
- Example: A text file where lines are written sequentially.
- Best for reading large amounts of data in order.
- **Indexed File Structure:**
- Uses an index (like a table of contents) to quickly locate records.
- Example: Database files where each record has an index pointer.
- Faster than sequential when searching specific data.
- **Hashed File Structure:**
- Uses a hash function to calculate the location of data.
- Example: Password storage systems.
- Very fast for random access.
- Importance: The choice of file structure affects speed, efficiency, and how easily data can be updated or retrieved.

📄 **File Types.**

File types indicate the format and purpose of a file. They are usually identified by extensions (like txt ,jpeg ,exe).

- **Categories of File Types:**

1. Text Files (.txt, .docx): Store readable characters. Used for documents, notes, coding scripts.
2. Image Files (.jpg, .png, .gif): Store graphical data. Used in design, photography, web content.
3. Audio/Video Files (.mp3, .mp4, .avi): Store multimedia content. Used for entertainment and communication.
4. Executable Files (.exe, .bat): Contain instructions for the computer to run programs.
5. Compressed Files (.zip, .rar): Reduce file size and bundle multiple files together.
6. System Files (.dll, .sys): Used by the OS to run properly.• Role in OS: File types help the operating system decide which application should open the file. For example, opens in MS Word, in a media player.

7. What are the goals of I/O software? Explain DMA operation

Goals of I/O Software

I/O (Input/Output) software is part of the operating system that manages communication between the computer and its devices (like keyboard, mouse, printer, disk, etc.). Its main goals are:

1. **Device Independence**

• Programs should work with any device without needing to know hardware details.

• Example: A text editor can save files on HDD, SSD, or USB without changing code.

2. **Uniform Naming**

• All devices and files should be accessed in a standard way (like using file names or paths).

3. **Error Handling**

• Detect and manage errors (like printer out of paper, disk read error) without crashing the system.

4. **Buffering & Caching**

• Temporarily store data to match speed differences between fast CPU and slower I/O devices.

• Example: Keyboard input buffered before being processed.

5. **Concurrency (Multiplexing)**

• Allow multiple processes to use I/O devices simultaneously without conflict.

• Example: Printing while downloading a file.

6. **Security & Protection**

• Prevent unauthorized access to devices and data.

• Example: Restricting access to system files.

DMA (Direct Memory Access) Operation

DMA is a technique that allows devices (like disk controllers or network cards) to transfer data directly to/from main memory without involving the CPU for each byte.

• Why it's needed: CPU is very fast compared to I/O devices. If CPU handled every byte transfer, it would waste time. DMA frees the CPU to do other tasks.

🔄 Steps in DMA Operation

1. CPU Setup:

• CPU tells the DMA controller: source address, destination address, and amount of data to transfer.

2. DMA Controller Takes Over:

• DMA controller manages the transfer directly between device and memory.

• CPU is not involved in the actual data movement.

3. Data Transfer:

• Device sends/receives data directly to/from memory via the system bus.

4. Completion & Interrupt:

• Once transfer is done, DMA controller sends an interrupt to CPU.
CPU resumes processing with the new data available.

8. Define transaction. Describe transaction look aside buffer (TBL)

A Transaction is a single logical unit of work that consists of one or more operations (like reading or writing data). In a database or an operating system, a transaction must be treated as an "all or nothing" deal.

Transaction Lookaside Buffer (TLB)

A Transaction Lookaside Buffer (TLB) is a special high-speed cache used in computer memory management. It stores recent translations of virtual addresses to physical addresses.

Purpose:

- Speeds up memory access during transactions.
- Avoids repeated lookups in the page table (which is slower).
- Ensures efficient execution of processes that rely on virtual memory.

How it works:

1. When CPU generates a virtual address, it first checks the TLB.
2. If the translation is found (TLB hit), physical address is returned quickly.
3. If not found (TLB miss), OS/page table is consulted, and the result is stored in TLB for future use.

Benefits:

- Faster memory access.
- Reduces CPU overhead during transactions.
- Improves overall system performance.

9. Write short notes on

🔔 **Interrupts**

- Definition: An interrupt is a signal sent to the CPU by hardware or software indicating that immediate attention is needed.
- Types:

1. Hardware interrupts → triggered by devices (keyboard, mouse, printer).
2. Software interrupts → triggered by programs or system calls.

- Process:
- CPU pauses current execution.
- Saves the state of the program.
- Executes the interrupt service routine (ISR).
- Resumes the original task.
- Importance:
- Allows multitasking.
- Improves efficiency by letting CPU respond to events immediately.
- Example: Keyboard input interrupt while CPU is processing another task.

b) 💿 Disk Scheduling

- Definition: Disk scheduling is the method used by the operating system to decide the order in which disk I/O requests are served.
- Need: Disk access time is slow compared to CPU, so scheduling improves performance.
- Common Algorithms:

1. FCFS (First Come First Serve): Requests handled in arrival order.
2. SSTF (Shortest Seek Time First): Chooses request closest to current head position.
3. SCAN (Elevator Algorithm): Head moves back and forth across disk, serving requests in order.
4. C-SCAN (Circular SCAN): Head moves in one direction, then jumps back to start.

- Goal: Minimize seek time, reduce latency, and improve throughput.

c) 📦 FIFO (First In, First Out)

- Definition: FIFO is a scheduling or queue management technique where the first request that arrives is the first to be served.

Characteristics:

- Simple and fair.
- Works like a queue in real life (first person in line gets served first).

Applications:

- CPU Scheduling: Processes executed in order of arrival.
- Disk Scheduling: Requests handled in arrival order.
- Memory Management: Pages replaced in the order they entered the memory (FIFO page replacement).

Limitation:

- May cause convoy effect (slow process delays all others).
- Not always optimal for performance..

REGULAR/BACK EXAM-2080 BHADRA

1. Define operating system. What are the function of operating system? Explain OS as resource manager.

Definition of Operating System

An Operating System (OS) is system software that acts as an interface between the user and computer hardware. It manages hardware resources and provides services so that application programs can run smoothly.

Functions of Operating System

The main functions of an OS are:

1. Process Management
- Handles creation, scheduling, and termination of processes.
- Ensures fair CPU time distribution.
2. Memory Management
- Allocates and deallocates memory to processes.
- Uses techniques like paging and segmentation.
3. File Management
- Manages files and directories on storage devices.
- Provides operations like create, read, write, delete.
4. Device Management
- Controls input/output devices through drivers.
- Ensures smooth communication between hardware and software.
5. Security & Protection
- Protects data and resources from unauthorized access.
- Provides user authentication and access control.
6. User Interface
- Provides CLI (Command Line Interface) or GUI (Graphical User Interface).
- Makes interaction with the system easier.
7. Networking & Communication
- Supports data exchange between computers.
- Provides protocols and connectivity.

🛠️ **OS as Resource Manager**

- The OS is often called a resource manager because it manages all the resources of a computer system:
- CPU (processor time)
- Memory (RAM)
- Storage (files, disks)
- I/O devices (keyboard, printer, etc.)
- Role:
- Allocates resources to different processes fairly and efficiently.
- Prevents conflicts when multiple processes request the same resource.
- Ensures maximum utilization of resources without wastage.

2. Define process, program and threads. Compare program and process.

1. Definitions

Program

A **Program** is a passive entity. It is simply a file stored on your disk (like chrome.exe or vlc.app) that contains a set of instructions written by a programmer. It stays "asleep" until you double-click it.

Process

A **Process** is an active entity. It is a **program in execution**. When you run a program, it is loaded into the RAM, and the OS creates a process for it. A process includes the program code, its current activity (tracked by the Program Counter), and its assigned resources (memory, files, etc.).

Thread

A **Thread** is a "lightweight process" or a subset of a process. It is the smallest unit of execution that the CPU can handle. A single process can have multiple threads running at once, sharing the same memory space but performing different tasks (like one thread playing music while another thread downloads a file).

2. Program vs. Process: The Key Differences

While they are related, they are fundamentally different in terms of how they exist and how the system treats them.

Basis	Program	Process
Nature	It is a **Passive** entity.	It is an **Active** entity.
Life Span	It is permanent (stored on disk until deleted).	It is temporary (starts when opened, ends when closed).
Resources	It does not require any RAM or CPU resources.	It requires RAM, CPU time, and I/O resources to run.
Storage	It is stored on the **Secondary Memory** (HDD/SSD).	It is loaded into the **Primary Memory** (RAM).
Relationship	One program can create many processes.	A process cannot exist without a program.
Definition	A collection of instructions.	An instance of a program being executed.

Export to Sheets

3. Define process model. Explain five state process model with states and transitions

In Operating Systems, a **Process Model** is a conceptual framework that describes the life cycle of a process. It defines how a process moves from its creation to its completion and how the OS tracks its progress using various states.

Five-State Process Model

The five-state model is a standard way to represent the behavior of a process. When a process is executed, it changes its state depending on its needs (like waiting for a user to click a button) and the availability of the CPU.

The Five States Explained

1. **New:** The process is being created or loaded from the disk into the primary memory. It hasn't been admitted to the "Ready" pool yet.
2. **Ready:** The process is loaded into the RAM and is waiting for its turn to use the CPU. It has all the resources it needs except for the processor itself.
3. **Running:** The CPU is currently executing the instructions of this process. In a single-core system, only one process can be in this state at a time.
4. **Waiting (Blocked):** The process cannot continue until a specific event occurs (like waiting for I/O completion or a signal from another process).[1]
5. **Terminated (Exit):** The process has finished its execution or has been killed by the OS.[2] It is now being removed from the system's active list.[3]

Transitions (The Movements)

The "lines" connecting these states in a diagram represent **Transitions**.[4] Here is how they happen:

- **Admitted (New $\rightarrow$ Ready):** The OS accepts the process and moves it to the ready queue.
- **Scheduler Dispatch (Ready [5]$\rightarrow$ Running):** The OS chooses this process from the queue and gives it the CPU.[6]
- **Interrupt (Running $\rightarrow$ Ready):** The OS takes the CPU away from a process (maybe because its time is up) and puts it back in the queue to give another process a turn.
- **I/O or Event Wait (Running [7]$\rightarrow$ Waiting):** The process pauses itself because it needs to wait for something (like reading a file from the disk).[8]
- **I/O or Event Completion (Waiting [9]$\rightarrow$ Ready):** The event the process was waiting for has finished, so it moves back to the "Ready" queue to wait for the CPU again.[10]
- **Exit (Running $\rightarrow$ Terminated):** The process finishes its last instruction and tells the OS it is done.

4. **What is virtual memory? Explain any two page replacemen replacement algorithms**.

Definition: Virtual memory is a memory management technique that gives an application the illusion of having a large, continuous block of memory, even if the physical RAM is smaller

Page Replacement Algorithms

When a process requests a page that isn't in RAM (a **Page Fault**), and the RAM is completely full, the OS must choose an existing page to "kick out" to make room for the new one. This choice is made by a **Page Replacement Algorithm**.

Here are two of the most common ones:

1. FIFO (First-In, First-Out)

This is the simplest algorithm. The OS keeps track of all pages in a queue.

- **The Logic:** The page that has been in memory the longest is the one selected for replacement.
- **Pros:** Very easy to implement using a queue data structure.
- **Cons:** It can be inefficient because a page that is used very frequently might be kicked out simply because it was loaded a long time ago.
- **Belady's Anomaly:** This algorithm suffers from a strange phenomenon where increasing the number of page frames (more RAM) can actually *increase* the number of page faults.

2. LRU (Least Recently Used)

LRU is much smarter than FIFO. It looks at the **recent history** of page usage to predict the future.

- **The Logic:** The page that has not been used for the longest period of time is replaced. It assumes that if you haven't used a page in a while, you probably won't need it soon.
- **Pros:** Generally performs much better than FIFO and does not suffer from Belady's Anomaly. It is considered a very efficient "real-world" algorithm.
- **Cons:** It is harder to implement because the OS must keep track of exactly when every page was last accessed (this requires special hardware support like counters or stacks).

.

5. Define deadlock. What are the condition for deadlock? Explain deadlock prevention.

a Deadlock is a situation where a set of processes are stuck in a "traffic jam" because each process is holding a resource and waiting for another resource held by someone else.[1] None of the processes can move forward, and the system hangs.

1. Necessary Conditions for Deadlock

For a deadlock to occur, all four of these conditions (known as **Coffman Conditions**) must be true at the exact same time.[2] If you can break even one of these, you can prevent a deadlock.

1. **Mutual Exclusion:** At least one resource must be held in a non-sharable mode.[3] Only one process can use the resource at a time (e.g., a printer).
2. **Hold and Wait:** A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.[4]
3. **No Preemption:** Resources cannot be forcibly taken from a process.[5] They can only be released voluntarily by the process holding them.[6]
4. **Circular Wait:** A set of processes [7]$\{P_0, P_1, ..., P_n\}$ must exist such that [8]P_0 is waiting for a resource held by [9]P_1, [10]P_1 is waiting for [11]P_2, and [12]P_n is waiting for [13]P_0.[14]

2. Deadlock Prevention

Deadlock Prevention works by ensuring that at least one of the four conditions mentioned above **cannot** occur.[15] Here is how the OS tackles each one:

- **Preventing Mutual Exclusion:** This is difficult because some resources (like a tape drive) are naturally non-sharable. However, for some devices, we use **Spooling** (like a print spooler) so multiple processes can "share" a virtual version of the device.[16]
- **Preventing Hold and Wait:** The OS can require a process to request all its required resources **at once** before it starts. If it can't get everything, it gets nothing. Alternatively, it can only allow a process to request a resource if it currently holds none.
- **Preventing No Preemption:** If a process is holding some resources and requests another that cannot be immediately allocated, the OS **forcibly releases** all resources currently held by that process. The process will have to start over and request them all again later.
- **Preventing Circular Wait:** The OS assigns a **linear numerical order** to all resource types (e.g., Disk = 1, Printer = 2, RAM = 3). A process must request resources in increasing order. It cannot request a "1" if it already holds a "3." This mathematically breaks the possibility of a circle.

6. What is process scheduling? Explain RR scheduling with an example.

Process Scheduling is the mechanism the Operating System uses to decide which process in the "Ready" queue gets to use the CPU and for how long. Since a CPU can usually only execute one task at a time, the scheduler ensures that every process gets a fair share of processor time, maximizing efficiency and minimizing waiting time.

Round Robin (RR) Scheduling

Round Robin is one of the most popular scheduling algorithms, designed specifically for time-sharing systems.

- **The Logic:** Each process is assigned a small, fixed unit of time called a **Time Quantum** (or Time Slice).
- **How it works:** 1. The CPU picks the first process from the queue and runs it.

2. If the process finishes before the time quantum expires, it leaves the system.
3. If the process is still running when the time quantum ends, the OS **interrupts** it (preemption), moves it to the back of the "Ready" queue, and gives the CPU to the next process.

- **Key Characteristic:** It is **Preemptive**—the OS doesn't wait for a process to finish; it forces a switch when time is up.

Example of RR Scheduling

Let's assume we have 3 processes and a **Time Quantum of 4 ms**.

Process	Burst Time (Total time needed)
P1	10 ms
P2	3 ms

Process	Burst Time (Total time needed)
P3	5 ms

The Execution Order (Gantt Chart):

1. **0 - 4 ms: P1** runs for the first 4 ms. It still has 6 ms left. It is moved to the back of the queue.
2. **4 - 7 ms: P2** runs. Since it only needs 3 ms (which is less than the 4 ms quantum), it finishes early and leaves the system.
3. **7 - 11 ms: P3** runs for 4 ms. It still has 1 ms left. It is moved to the back of the queue.
4. **11 - 15 ms: P1** returns and runs for another 4 ms. It has 2 ms left.
5. **15 - 16 ms: P3** returns and runs for its final 1 ms. **P3 is finished.**
6. **16 - 18 ms: P1** returns and runs for its final 2 ms. **P1 is finished.**

7. Describe contiguous allocation. Explain how link list allocation helps to overcome problems of contiguous allocation.

1. Contiguous Allocation

Contiguous Allocation requires each file to occupy a set of adjacent blocks on the disk.[2] For example, if a file starts at block 100 and has a length of 5, it must occupy blocks 100, 101, 102, 103, and 104.[3]

- **How it works:** The directory entry for each file only needs to store the **Starting Block Address** and the **Length** of the file.[4]
- **Advantages:**
  - **High Performance:** Since all blocks are next to each other, the disk head moves very little, making it extremely fast for sequential reading.[5]
  - **Supports Direct Access:** You can easily calculate the address of the [6]n^{th} block (e.g., [7]$Start + n$).[8]
- **Major Disadvantages:**
  - **External Fragmentation:** As files are created and deleted, the free space gets broken into small, non-contiguous holes.[9] You might have enough total free space for a new file, but if it isn't in one continuous block, the file cannot be stored.[10]
  - **File Growth Issues:** If you want to add data to a file, but the block immediately after it is already taken by another file, you have to move the entire file to a new, larger hole.[11]

2. Linked List Allocation

Linked List Allocation solves the problems of contiguous allocation by allowing a file to be scattered anywhere on the disk.[12]

- **How it works:** Each file is a linked list of disk blocks.[13] Every block contains two things:
  1. The actual **Data**.
  2. A **Pointer** (address) to the next block in the file.[14]
- The directory entry only needs to store the address of the **First Block** and (optionally) the **Last Block**.

How Linked List Overcomes Contiguous Allocation Problems

1. **Eliminates External Fragmentation:** Since any free block can be used for any file, no space is wasted.[15] The OS doesn't need a single "big hole"; it can use 100 small, scattered holes to store one 100-block file.
2. **No Need to Pre-declare File Size:** In contiguous allocation, you often have to tell the OS how big the file will be when you create it. In Linked List, a file can grow as long as there is at least one free block anywhere on the disk.[16]
3. **Easy File Expansion:** If you need to add data, the OS simply finds any empty block, writes the data, and updates the pointer in the previous "last block" to point to this new one.

8. Define DMA. Explain DMA controller data transfer mode Differentiate between programmed I/O and interrupt driver I/O .

1. Definition of DMA (Direct Memory Access)

Direct Memory Access (DMA) is a hardware-driven technique that allows an I/O device to send or receive data directly to or from the main memory (RAM) without involving the CPU for every byte transferred.[1]

2. DMA Controller Data Transfer Modes

The DMA controller can transfer data in three primary ways, depending on how it "borrows" the system bus from the CPU:

- **Burst Mode (Block Transfer):** The DMA controller takes control of the bus and stays in control until the *entire* block of data is transferred.[4]
  - *Pro:* Very fast data transfer.
  - *Con:* The CPU is completely blocked from accessing memory until the DMA is finished.[5]
- **Cycle Stealing Mode:** The DMA controller transfers one byte (or word) of data and then releases the bus back to the CPU.[6] it "steals" one bus cycle at a time.
  - *Pro:* The CPU doesn't have to wait long; it can keep working alongside the DMA.[7]
- **Transparent Mode:** The DMA controller only uses the system bus when it knows the CPU is busy with internal operations and doesn't need the bus.
  - *Pro:* It has zero impact on CPU performance.
  - *Con:* It is the slowest mode because it has to wait for "gaps" in CPU activity.

3. Differentiate: Programmed I/O vs. Interrupt-Driven I/O

These are the two traditional ways the CPU handles I/O before we move up to advanced methods like DMA.

Basis of Comparison	Programmed I/O	Interrupt-Driven I/O
CPU Involvement	**Maximum:** The CPU stays in a loop checking if the device is ready.	**Minimum:** The CPU starts the I/O and goes back to other work.
Efficiency	**Inefficient:** CPU wastes time "polling" the device (Busy Waiting).	**Efficient:** CPU only reacts when the device signal "interrupts" it.
Status Check	CPU repeatedly asks the device "Are you done yet?"	Device tells the CPU "I am done!" via a signal.
Speed	Slow; depends on the speed of the I/O device.	Faster; better utilization of CPU cycles.
Hardware	Simple; requires no special interrupt hardware.	Complex; requires an interrupt controller and lines.

9. Write short notes on:

a. Peterson's Solution

Peterson's Solution is a classic software-based solution to the **Critical Section Problem**.[1] It is designed to ensure that two processes can share a single resource without conflict (Mutual Exclusion).[2]

- **How it Works:** It uses two shared variables:
  1. flag[2]: A boolean array where flag[i] = true means process [3]i wants to enter the critical section.[4]
  2. turn: An integer that indicates whose turn it is to enter.[5]
- **The Logic:** Before entering the critical section, a process sets its flag to true and "graciously" sets the turn to the other process.[6] It only enters the critical section if it is its turn **or** if the other process does not want to enter.
- **Requirements Met:** It successfully satisfies all three requirements: **Mutual Exclusion**, **Progress**, and **Bounded Waiting**.[7]
- **Limitation:** It is a "busy-waiting" solution (the CPU wastes cycles looping while waiting) and is primarily restricted to two processes.[8]

b. Importance of Segmentation

As we discussed earlier, **Segmentation** is a memory management scheme that supports the **user's view of memory**.[9] Instead of seeing memory as a linear array of bytes, the user sees it as a collection of logical modules (segments) like main program, stack, function, etc.

Key Importance:

1. **Logical Protection:** Since segments correspond to logical units, the OS can easily set permissions.[10] For example, a "Code Segment" can be marked as read-only, preventing accidental overwriting.[11]
2. **Sharing:** It is easier to share modules between processes.[12] Two users can share the same "Utility" segment while having their own "Data" segments.
3. **No Internal Fragmentation:** Segments are sized exactly to the data they hold, so no memory is wasted inside a segment.
4. **Dynamic Growth:** Segments like the **Heap** or **Stack** can grow and shrink independently without needing to move other parts of the program.[13]

c. FCFS (First-Come, First-Served) Disk Scheduling

FCFS is the simplest form of disk scheduling.[14] The disk service requests in the exact order they arrive in the I/O queue.[15]

- **Mechanism:** If the disk head is at cylinder 50 and the queue is $\{100, 20, 150\}$, the head moves to 100, then all the way back to 20, and then out to 150.
- **Advantages:**
  - **Fairness:** Every request is handled in the order it was received.
  - **Simplicity:** It is very easy to program and requires no complex logic.
  - **No Starvation:** Every request will eventually be served.
- **Disadvantages:**
  - **Inefficient:** It does not try to optimize the movement of the disk head.[16] This leads to a high **"Seek Time"** because the head might have to "zig-zag" across the disk surface.
  - **Low Throughput:** Because of the excessive head movement, the overall speed of data retrieval is slow compared to algorithms like SCAN or SSTF.

REGULAR/BACK EXAM-2079 CHAITRA/2080 BAISHAKH

1. "Operating System Acts as Resource Manager" and "Operating System Acts as Extended Machine". Justify these statements.

1. OS as a Resource Manager

This is the Bottom-Up view. A computer has many resources (CPU, RAM, Disks, Network interfaces, I/O devices) that are finite and expensive. The OS acts as a manager to ensure these resources are used efficiently.

Justification:

- Resource Allocation: When multiple programs run, the OS decides which process gets the CPU and for how long (Scheduling).
- Conflict Resolution: If two users try to print at the same time, the OS manages the queue so they don't overlap.
- Efficiency: It tracks which parts of the memory are in use and which are free to prevent wastage.
- Security: It ensures that one process cannot steal or corrupt the resources belonging to another.

2. OS as an Extended Machine

This is the Top-Down view. Physical hardware is incredibly complex and "ugly" to program. For example, reading a sector from a hard drive involves complicated commands for motor speed, head positioning, and error checking.

Justification:

Abstraction: The OS hides the messy hardware details and presents the user with a "clean" abstract version. Instead of seeing disk sectors, the user sees "Files".

- Ease of Use: It provides a set of System Calls that make it easy for programmers to perform complex tasks without knowing how the hardware works.
- Virtualization: It creates an "Extended Machine" (or Virtual Machine) that is much easier to operate than the bare metal.
- Uniform Interface: Whether you are using a Seagate HDD or a Samsung SSD, the OS makes them both look exactly the same to your softwar

2. Define process scheduling. Explain first come first served and shortest remaining time next with example.

Process Scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular

strategy. It is a fundamental part of multiprogramming operating systems, allowing the computer to switch the CPU among processes to make the system more productive.

1. First Come First Served (FCFS)

This is the simplest scheduling algorithm. It follows the **Non-Preemptive** policy, meaning once a process gets the CPU, it holds it until it terminates or requests I/O.

- **Logic:** Processes are assigned the CPU in the order they request it (using a FIFO queue).
- **Problem:** It can suffer from the **Convoy Effect**, where short processes have to wait a long time for one big process to finish.

Example of FCFS:

Consider three processes:

Process	Arrival Time	Burst Time
P1	0	10
P2	1	2
P3	2	1

Gantt Chart:

- **0 to 10:** P1 runs to completion.
- **10 to 12:** P2 runs.
- **12 to 13:** P3 runs.
- **Avg Waiting Time:** $(0 + (10-1) + (12-2)) / 3 = 6.33$ ms.

2. Shortest Remaining Time Next (SRTN)

SRTN is the **Preemptive** version of Shortest Job First (SJF) scheduling.

- **Logic:** The OS always chooses the process that has the **shortest remaining execution time**.
- **Preemption:** If a new process arrives with a burst time shorter than the remaining time of the current running process, the current process is stopped (preempted) and the new one starts.

Example of SRTN:

Using the same processes:

Process	Arrival Time	Burst Time
P1	0	10
P2	1	2
P3	2	1

Gantt Chart:

1. **0 to 1: P1** starts (Remaining: 9).
2. **1 to 2: P2** arrives (Burst: 2). Since 2 is less than P1's remaining 9, **P2 takes over**. (P2 Remaining: 1).
3. **2 to 3: P3** arrives (Burst: 1). Since P3's 1 is equal to P2's remaining 1, we continue with P2 or switch to P3 (let's say P3 starts). **P3 runs to completion**.
4. **3 to 4: P2** finishes its last 1 ms.
5. **4 to 13: P1** finishes its remaining 9 ms.

- **Avg Waiting Time:** P1 waits 4ms, P2 waits 0ms, P3 waits 0ms. $(4+0+0)/3 = 1.33$ms.

3. Define IPC. What is mutual exclusion and race condition? Explain Peterson's solution.

. What is IPC (Inter-Process Communication)?

IPC is a set of programming interfaces that allow processes to communicate with each other and synchronize their actions.[2] Since processes often have separate memory spaces, they cannot naturally "see" each other's data.

There are two main models of IPC:

- **Shared Memory:** Processes agree on a region of memory that they can both read from and write to.[3]
- **Message Passing:** Processes communicate by sending messages to each other through the OS (useful for processes on different computers).[4]

2. Race Condition

A **Race Condition** occurs when two or more processes access and manipulate shared data at the same time, and the final outcome depends on the **timing** or order of execution.[5]

Example: If two processes try to increment a shared counter ([6]$count = count + 1$) simultaneously, they might both read the value "5," increment it to "6," and save it.[7] The counter becomes 6 instead of 7. One increment is "lost" because of the "race."

3. Mutual Exclusion

Mutual Exclusion is the requirement that if one process is executing in its **Critical Section** (the part of the code where shared data is accessed), no other process is allowed to enter their critical section at the same time.[8] It is the primary solution to prevent race conditions.

4. Peterson's Solution

Peterson's Solution is a classic, software-based algorithm used to achieve **Mutual Exclusion** for two processes.[9] It ensures that only one process enters its critical section at a time.

The Logic

It uses two variables to keep track of the state:

1. **flag[2]**: A boolean array.[10] If flag[i] is true, process i is ready to enter the critical section.
2. **turn**: An integer that holds the ID of the process whose "turn" it is.[11]

How it works (Step-by-Step):

1. **Request:** Process [12]P_i sets flag[i] = true and sets turn = j (giving the other process a chance).[13]
2. **Wait:** P_i enters a while loop. It waits as long as the other process wants to enter (flag[j] == true) AND it is the other process's turn (turn == j).
3. **Critical Section:** Once the other process finishes or gives up, P_i enters the critical section.
4. **Exit:** After finishing, [14]P_i sets flag[i] = false, allowing the other process to enter.[15]

4. What is deadlock? Write down conditions for deadlock to occur and deadlock prevention techniques.

a **Deadlock** is a "gridlock" in an operating system. It occurs when a set of processes are blocked because each process is holding a resource and waiting for another resource held by another process in the same set. Since everyone is waiting and no one is releasing, the system comes to a standstill.

1. Necessary Conditions for Deadlock

For a deadlock to occur, these four conditions (known as the **Coffman Conditions**) must hold true simultaneously. If you can break any one of these, a deadlock cannot happen.

1. **Mutual Exclusion:** At least one resource must be held in a non-sharable mode. Only one process can use the resource at a time (e.g., a printer or a tape drive).
2. **Hold and Wait:** A process is currently holding at least one resource and is waiting to acquire additional resources that are currently being held by other processes.
3. **No Preemption:** Resources cannot be taken away from a process by force; they can only be released voluntarily by the process after it has finished its task.
4. **Circular Wait:** A closed chain of processes exists such that each process holds at least one resource needed by the next process in the chain.

2. Deadlock Prevention Techniques

Deadlock prevention works by ensuring that **at least one** of the four necessary conditions listed above is never met.

Breaking Mutual Exclusion

- **Shareable Resources:** Make resources shareable whenever possible. For non-shareable resources (like printers), use **Spooling**. This allows the OS to handle the resource while the process thinks it has exclusive access.

Breaking Hold and Wait

- **Resource Request at Start:** Require processes to request all their required resources at the very beginning. The process only starts if *all* resources are available.
- **No Resources Held:** A process can only request a new resource if it currently holds none. It must release its current resources before requesting new ones.

Breaking No Preemption

- **Resource Preemption:** If a process is holding some resources and requests another that cannot be immediately allocated, all its currently held resources are **preempted** (taken away). The process is restarted only when it can regain its old resources plus the new ones it requested.

Breaking Circular Wait

- **Hierarchical Ordering:** Assign a unique numerical integer to every resource type (e.g., Disk=1, Printer=2, RAM=3).
- **Strict Order:** Processes must request resources in a strictly increasing (or decreasing) order. A process holding resource "2" cannot request resource "1." This mathematically prevents a circle from ever forming.

5. Explain memory management with fixed and variable partition briefly.

Definition

Memory management is the process of efficiently handling a computer's primary memory.It allocates, tracks, and frees memory space for processes during execution.

Its goal is to maximize utilization, avoid fragmentation, and ensure smooth multitasking.

Functions of Memory Management

- Allocation: Assigns memory to processes when they are created.
- Protection: Ensures one process does not interfere with another's memory.
- Sharing: Allows controlled sharing of memory among processes.
- Relocation: Adjusts memory addresses when processes are moved.
- Deallocation: Frees memory when a process terminates.

1. Fixed Partitioning (Static)

In this method, the main memory is divided into a set of non-overlapping regions called **partitions** at the time of system generation or boot-up.

- **How it Works:** Each partition can hold exactly **one** process. Even if a partition is larger than the process, it remains occupied until the process finishes.
- **Internal Fragmentation:** This is the biggest disadvantage. If a process is smaller than the allocated partition, the leftover space inside that partition is wasted and cannot be used by any other process.
- **Limit on Multiprogramming:** The number of processes that can be in memory at once is strictly limited by the number of partitions created.

2. Variable Partitioning (Dynamic)

In this method, partitions are created dynamically as processes arrive. There are no pre-defined boundaries; the OS allocates exactly the amount of memory a process requests.

- **How it Works:** Initially, all memory is treated as one large block (a "hole"). When a process arrives, the OS carves out a block specifically for its size.
- **External Fragmentation:** As processes finish and leave, they leave behind small holes. Over time, these holes become so scattered that even if the total free space is large, you might not have a single **contiguous** block big enough for a new process.
- **Compaction:** To solve external fragmentation, the OS uses **Compaction** (shuffling processes to one side of the RAM to merge all small holes into one large block), but this is very time-consuming.

6. Explain contiguous and linked list allocation of file using FAT.

1. Contiguous Allocation

In this method, each file occupies a set of **adjacent (consecutive)** blocks on the disk.

- **How it works:** To store a file, the OS looks for a single "hole" of free blocks large enough to fit the entire file. The directory entry only needs to store the **Starting Block Address** and the **Length** of the file.
- **Advantages:**
  - **High Performance:** Since blocks are next to each other, the disk head doesn't have to move much, making sequential reading very fast.
  - **Direct Access:** You can quickly jump to any part of the file (e.g., to find block 5, you just calculate $Start + 5$).
- **Disadvantages:**
  - **External Fragmentation:** As files are deleted, the disk becomes "checkered" with small gaps that might be too small for new files.
  - **File Growth:** If you want to expand a file but the next block is already taken, the whole file must be moved to a bigger location.

2. Linked List Allocation using FAT

A standard Linked List allocation stores a pointer to the next block *inside* the data block itself. However, **FAT (File Allocation Table)** improves this by moving all those pointers into a separate table.

- **What is FAT?** The FAT is a table stored at the beginning of the disk partition. Each entry in the table corresponds to a physical block on the disk.
- **How it works:**
  1. The directory entry for a file stores only the **Starting Block Number**.
  2. To find the second block, the OS looks at the FAT entry for the first block. That entry contains the address of the next block.
  3. This continues until the OS hits a special **End-of-File (EOF)** marker in the table.

How FAT helps overcome standard Linked List problems:

- **Faster Searching:** In a standard linked list, the OS has to read every disk block just to find the pointer. With FAT, the OS can follow the entire chain of blocks just by looking at the table in memory, without moving the disk head.
- **Reliability:** Since the data block is 100% data (no pointers inside it), the file structure is cleaner.
- **No External Fragmentation:** Like a regular linked list, a file can be scattered anywhere on the disk. Any free block can be used, so no space is wasted.

7. What is DMA? Write differences between memory mapped IO and IO mapped 10.

Direct Memory Access (DMA) is a technique that allows devices to transfer data directly to/from memory.It bypasses the CPU during data transfer, reducing CPU overhead.DMA improves system performance, especially for large data transfers like disk or network I/O.

Comparison Between Memory Mapped I/O and I/O Mapped I/O

S.N.	Basis	Memory Mapped I/O	I/O Mapped I/O (Isolated I/O)
1	**Address Space**	I/O devices and RAM share the **same** address space.	I/O devices have a **separate** address space from RAM.
2	**Instructions**	Uses standard memory instructions like MOV, LOAD, and STORE.	Uses special I/O instructions like IN and OUT.
3	**Control Signals**	Uses common control signals (MEMR / MEMW) for both RAM and I/O.	Uses separate control signals (IOR / IOW) for I/O devices.
4	**CPU Pins**	No extra pins are required on the CPU for I/O addressing.	Extra pins/signals are required to distinguish between Memory and I/O.
5	**Memory Capacity**	It reduces the available memory space for programs (since some addresses are taken by I/O).	It does not affect RAM capacity; the full memory address space is available.
6	**Hardware Logic**	The hardware decoder logic for I/O devices is more complex.	The hardware decoder logic is simpler because of the dedicated address space.
7	**Ease of Programming**	Easier for programmers because all memory manipulation instructions work on I/O.	Requires specific knowledge of special I/O instructions.
8	**Bus Architecture**	Only one bus is used for both memory and I/O devices.	Can use a separate I/O bus or a dedicated control line for the I/O space.

8.a. Write differences between random access and sequential acces.

This is a classic comparison between how data is retrieved from storage media.

Basis of Comparison	Sequential Access	Random (Direct) Access
Access Method	Data is accessed in a specific linear order (one after another).	Data can be accessed directly from any location.
Efficiency	Highly efficient for reading large files in bulk (like a full backup).	Efficient for retrieving specific records without reading the whole file.
Search Speed	Slower; to read the 100th record, you must read the first 99.	Faster; the system jumps straight to the required address.
Media Examples	Magnetic Tapes, Cassettes, VHS tapes.	RAM, Hard Disk Drives (HDD), SSDs, DVDs.
File Structure	Usually variable length records are easier to handle.	Generally requires fixed-length records or an index to jump.
Pointer Movement	Pointer moves only to the adjacent address.	Pointer can jump to any arbitrary address.
Analogy	Like a **Cassette tape**: You must fast-forward to get to the 5th song.	Like a **CD or Spotify**: You can click on any track to play it instantly.
Application	Used for batch processing (e.g., monthly payroll).	Used for databases and real-time systems.

b . What is paging? Explain any one page replacement algorithm.

Definition: Paging is a memory management technique that divides physical memory into fixed-size blocks called frames, and logical memory into blocks of the same size called pages.

Page Replacement Algorithm: FIFO (First-In, First-Out)

Explanation:

• Pages are maintained in a queue.

• When a page fault occurs and memory is full, the page at the front of the queue is replaced.

• Simple to implement but may not always be optimal.

Page Replacement Algorithm: FIFO (First-In, First-Out)

Since physical memory is limited, when a new page is needed and no frames are free, the OS must replace an existing page. **FIFO** is the simplest algorithm for this.

- **Logic:** The OS keeps track of all pages in memory in a queue. The page that has been in memory the longest (the "oldest" page) is the first one to be replaced.
- **Example:**
    - **Reference String:** 1, 3, 0, 3, 5, 6
    - **Number of Frames:** 3

    1. **Page 1** arrives $\rightarrow$ Frame: [1, _, _] (Page Fault)
    2. **Page 3** arrives $\rightarrow$ Frame: [1, 3, _] (Page Fault)
    3. **Page 0** arrives $\rightarrow$ Frame: [1, 3, 0] (Page Fault)
    4. **Page 3** arrives $\rightarrow$ Frame: [1, 3, 0] (Hit - already there)
    5. **Page 5** arrives $\rightarrow$ Page 1 is the oldest, so it's replaced. Frame: [**5**, 3, 0] (Page Fault)
    6. **Page 6** arrives $\rightarrow$ Page 3 is the next oldest, so it's replaced. Frame: [5, **6**, 0] (Page Fault)

8. Write shot notes on.

a. Process States

A process is not just a program code; it is a dynamic entity that changes its state as it executes. The lifecycle of a process is generally represented by the **Process State Transition Diagram**.

- **New:** The process is being created.
- **Ready:** The process is loaded into memory and is waiting to be assigned to a processor by the scheduler.
- **Running:** The instructions are being executed by the CPU.
- **Waiting (Blocked):** The process is waiting for some event to occur (such as I/O completion or receiving a signal).
- **Terminated:** The process has finished execution and is being removed by the OS.

Key Note: A process can move from **Running** back to **Ready** if an interrupt occurs (e.g., time slice expires in Round Robin).

b. File System Layout

The file system layout describes how the OS organizes data on a physical disk or partition. It is usually divided into several key sections:

1. **MBR (Master Boot Record):** The very first sector of the disk used to boot the computer and store the partition table.
2. **Partition Boot Block:** The first block of a partition, containing a short program to load the OS from that partition.
3. **Superblock:** Contains critical information about the file system, such as the total number of blocks, number of free blocks, and the file system type.
4. **Free Space Management:** Keeps track of blocks not currently in use (using a bit map or linked list).
5. **I-nodes (Index Nodes):** A data structure that stores all information about a file (size, permissions, owner) except its name and actual data.
6. **Data Blocks:** The actual area where the content of the files is stored.

c. LOOK Disk Scheduling Algorithm

LOOK is an improved version of the **SCAN (Elevator)** algorithm. It is designed to reduce unnecessary head movement.

- **How it works:** Like SCAN, the disk head moves in one direction (e.g., toward the highest cylinder), servicing all requests in its path.
- **The "Look" Difference:** Unlike SCAN, which goes all the way to the very end of the disk (cylinder 0 or the maximum cylinder), LOOK checks if there are any more requests in that direction. If there are no more requests ahead, it **reverses direction immediately**.
- **Advantage:** It prevents the disk head from traveling to the extreme ends of the disk when no requests are waiting there, saving time and energy.
- **Example:** If the disk has 200 cylinders and the last request is at 180, LOOK will turn back at 180, whereas SCAN would continue to 199.

REGULAR/BACK EXAM-2079, BHADRA/ASHWIN

1. .Justify "Operating System as resource manager". Explain the types of operating system in brief.

Justification: "Operating System as a Resource Manager"

A computer has various resources like the CPU, memory, disk space, and I/O devices. The OS acts as a manager of these resources for the following reasons:

1. **Resource Allocation:** When multiple programs run simultaneously, they all compete for CPU time and RAM. The OS decides which process gets the resource, for how long, and at what time.
2. **Efficiency:** The OS ensures that resources are used to their maximum potential. For example, while one process waits for I/O, the OS switches the CPU to another process so the processor doesn't sit idle.
3. **Conflict Resolution:** If two programs try to use the printer at the same time, the OS manages the "traffic" to prevent a crash or corrupted output.
4. **Protection and Security:** It ensures that one process cannot steal or interfere with the memory resources allocated to another process.

Types of Operating Systems

Operating Systems have evolved based on how they handle tasks and users. Here are the main types:

1. Batch Operating System

Users do not interact with the computer directly. Instead, they prepare their jobs (on punch cards or tapes) and submit them to a computer operator. The operator groups jobs with similar needs into "batches" to speed up processing.

- *Example:* Payroll systems.

2. Multiprogramming Operating System

To increase CPU utilization, multiple programs are kept in the main memory at once. If the current program waits for I/O, the OS switches the CPU to another program.

3. Time-Sharing (Multitasking) OS

This is a logical extension of multiprogramming. The CPU switches between multiple jobs so frequently that users can interact with each program while it is running. Each user is given a small slice of time (Time Quantum).

- *Example:* Windows, macOS, Linux.

4. Real-Time Operating System (RTOS)

Used in environments where a rigid time constraint is required. The system must provide a response within a very specific time limit (deadlines).

- *Example:* Air traffic control, medical imaging, missile systems.

5. Distributed Operating System

This system manages a group of independent computers and makes them appear to be a single computer to the user. It distributes the workload across multiple "nodes."

- *Example:* Google's cluster management systems.

6. Network Operating System

Runs on a server and provides the capability to manage data, users, groups, security, and applications over a network.

- *Example:* Windows Server, UNIX.

7. Multi-processor System

Also known as Parallel systems, these use two or more CPUs within a single computer system to increase throughput and reliability.

2. Why process need to be preemptive? Explain long and short term brust.

1. Why do processes need to be Preemptive?

A scheduling system is **Preemptive** when the OS can interrupt a currently running process to give the CPU to another process. This is necessary for several reasons:

- **Responsiveness (Interactive Systems):** In a multitasking OS (like Windows or Linux), preemption ensures that no single process "hogs" the CPU. This allows the UI to stay responsive even if a background task is doing heavy calculations.
- **Priority Handling:** If a high-priority "urgent" process arrives, a preemptive system can stop a low-priority process immediately to handle the urgent task.
- **Fairness:** It ensures that every process gets a fair share of CPU time using techniques like **Time Slicing** (Round Robin).
- **Preventing Deadlocks/Hangs:** If a process enters an infinite loop in a non-preemptive system, the whole computer would freeze. Preemption allows the OS to regain control.

3. **Long-term vs. Short-term Schedulers**

Schedulers are the OS components that handle the transition of processes between various states.

Long-Term Scheduler (Job Scheduler)

The Long-term scheduler determines which programs are admitted to the system for processing.

- **Function:** It selects processes from a "Job Pool" (on the disk) and loads them into the "Ready Queue" (in RAM).
- **Frequency:** It executes relatively **infrequently** (seconds or minutes).
- **Goal:** Its main job is to maintain a good **Degree of Multiprogramming** (the number of processes in memory). It tries to balance a mix of I/O-bound and CPU-bound processes.

Short-Term Scheduler (CPU Scheduler)

The Short-term scheduler decides which of the processes in the "Ready Queue" should be executed next by the CPU.

- **Function:** It allocates the CPU to a specific process from the RAM.
- **Frequency:** It executes very **frequently** (milliseconds). Because it runs so often, it must be extremely fast to avoid wasting CPU time.
- **Goal:** Its main goal is to optimize system performance (maximize throughput and minimize waiting time).

3.

4. How starvation differ dead lock? Explain deadlock handling policies.

Basis	Starvation	Deadlock
Meaning	A process waits for a very long time because it never gets CPU or resources.	Two or more processes wait forever for each other's resources.
Reason	Caused by unfair scheduling or priority system.	Caused by circular waiting for resources.
Progress	Other processes keep running; only one (or some) suffers.	No process involved can move forward.
System state	System is still working.	System comes to a complete halt for those processes.
Solution	Can be solved by aging or fair scheduling.	Needs deadlock prevention, avoidance, or recovery.
Example	Low-priority process never gets CPU time.	P1 holds R1 and waits for R2, P2 holds R2 and waits for R

Deadlock Handling Policies

1. Deadlock Prevention

- Ensure at least one of the four deadlock conditions (mutual exclusion, hold & wait, no preemption, circular wait) never occurs.
- Example: Impose ordering on resource requests to avoid circular wait.

2. Deadlock Avoidance

- System checks resource requests dynamically and decides whether granting them keeps the system in a safe state.
- Example: Banker's Algorithm ensures safe allocation.

3. Deadlock Detection and Recovery

- Allow deadlock to occur, then detect it using algorithms (like wait-for graph).
- Recovery methods: terminate processes or preempt resources to break the deadlock.

4. Ignore Deadlock (Ostrich Policy)

- In some systems, deadlocks are rare and cost of handling is high.
- The OS simply ignores deadlocks (used in UNIX, Windows).

5. How safe state in achieved in banker algorithm?

How a Safe State is achieved (The Safety Algorithm)

To determine if a state is safe, the Operating System follows these logic steps:

1. Initialization

Let Work be a vector representing the currently **Available** resources and Finish be a boolean array of size n (number of processes) initialized to false.[2]

2. Finding an Executable Process

The OS looks for a process P_i that satisfies two conditions:

- Finish[i] == false (The process hasn't finished yet).
- $Need_i \leq Work$ (Its current needs are less than or equal to what is available).

3. Simulating Execution and Resource Release

If such a process is found:

- The OS assumes P_i will finish its task.
- When it finishes, it releases all the resources it was holding back into the pool:[3]

$Work = Work + Allocation_i$

- Mark Finish[i] = true.[4]
- Go back to **Step 2** to find the next process.[5]

4. Verification

- If **all** processes are marked Finish == true, then the system is in a **Safe State** because a safe sequence has been found.[6]
- If there is no process that meets the conditions in Step 2 and some processes are still unfinished, the system is in an **Unsafe State**.

The Role of the "Safe Sequence"

The Safe Sequence is the specific order (e.g., [7]P_1, P_3, P_2, P_4) in which processes can be executed without causing a deadlock.[8]

Important Note: An **Unsafe State is NOT a Deadlock**. It is simply a state that *could* lead to a deadlock. A Safe State, however, **guarantees** that a deadlock will not occur.

Resource Allocation Logic

When a process requests resources, the Banker's Algorithm performs a "Trial Allocation":[9]

1. It pretends to grant the request.
2. It runs the **Safety Algorithm** (described above).
3. If the resulting state is **Safe**, the resources are actually granted.[10]
4. If the resulting state is **Unsafe**, the process must wait, and the old state is restored.[11]

6. Define swapping. Differentiate between fixed and variable sized partitioning in multiprogramming
Definition of Swapping.
Swapping is a memory management technique where a process is moved temporarily out of the main memory (RAM) to a secondary storage (usually the Hard Disk) and then brought back into memory for continued execution.
Differentiate between fixed and variable sized partitioning in multiprogramming

Feature	Fixed Partitioning	Variable Partitioning
Partition Creation	Created during system configuration or boot-up.	Created dynamically during runtime as processes arrive.
Partition Size	Fixed and cannot be changed.	Varies according to the size of the process.
Fragmentation Type	Suffers from **Internal Fragmentation**.	Suffers from **External Fragmentation**.
Degree of Multiprogramming	Limited by the number of pre-defined partitions.	Limited by the total available RAM and process sizes.
Process Size	A process cannot be larger than the largest partition.	A process can be as large as the total free RAM.
Management	Very simple for the OS to track (using a table).	Complex (must track every "hole" and "allocated" block).
Efficiency	Lower (due to wasted space inside slots).	Higher (allocates exactly what is needed).
Compaction	Not required.	Required to merge small holes (to solve external fragmentation).

7.What is segmentation? Explain the importance and drawbacks of segmentation.
Segmentation is a memory management technique that divides a program's memory into logical segments such as code, data, and stack.
Each segment has a variable size and is identified by a segment number.
It provides a logical view of memory rather than a purely physical one.
Importance of Segmentation
- Logical Division: Separates program into meaningful parts (code, data, stack).
- Flexibility: Each segment can grow or shrink independently.
- Protection: Provides segment-wise protection and controlled sharing.
- Modularity: Supports modular programming and simplifies debugging.
- Efficient Use: Allows processes to use only the memory they need.

Drawbacks of Segmentation
- External Fragmentation: Free memory may be scattered, making allocation difficult.
- Complex Management: Requires segment tables and hardware support, increasing overhead.
- Variable Sizes: Handling variable segment sizes is more complicated than fixed partitions.
- Performance Overhead: Address translation (segment number + offset) can slow down execution.

8. What is semaphore? Describe Peterson's algorithm.

Definition

A semaphore is a synchronization tool used in operating systems to manage concurrent processes.It is an integer variable that controls access to shared resources.
Semaphores prevent race conditions by using two atomic operations: wait (P) and signal (V)..

Peterson's Algorithm.

Peterson's Algorithm is a classic software-based solution to the critical section problem for **two processes**.[12] It does not require special hardware instructions and works by using two shared variables.[13]

The Shared Variables:

1. **boolean flag[2]**: An array used to indicate if a process is ready to enter the critical section.[14] (flag[i] = true means Process [15]i wants to enter).[16]
2. **int turn**: Indicates whose turn it is to enter the critical section.[17]

The Algorithm Structure (for Process i):

```
do {
   flag[i] = true;          // Process i is ready
   turn = j;                // Give turn to the other process j
   while (flag[j] && turn == j); // Wait if j is ready and it's j's turn

   // CRITICAL SECTION
   // (Access shared data here)

   flag[i] = false;         // Exit Section: Process i is done

   // REMAINDER SECTION
} while (true);
```

Why it works (Requirements Met):

- **Mutual Exclusion:** Only one process can enter.[18] If both try, the turn variable will eventually settle on one value, forcing one process to wait in the while loop.
- **Progress:** A process outside the critical section cannot block another process from entering.[19] If Process j doesn't want to enter (flag[j] = false), Process i can enter immediately.
- **Bounded Waiting:** A process will not have to wait forever. Once Process j finishes its critical section, it sets flag[j] = false, allowing Process i to enter.

9. Write short notes on:

. Process Scheduling

Process Scheduling is the OS activity that handles the removal of a running process from the CPU and the selection of another process based on a specific strategy. It is the heart of **Multiprogramming**.[1]

- **Objective:** To keep the CPU busy at all times and minimize the response time for users.[2]
- **Types of Schedulers:**
  - **Long-term (Job Scheduler):** Brings processes from the disk to the Ready Queue in RAM.[3]
  - **Short-term (CPU Scheduler):** Selects which process from the Ready Queue gets the CPU next.[4]
- **Scheduling Criteria:** Algorithms are judged based on **Throughput** (jobs per hour), **Turnaround Time** (submission to completion), and **Waiting Time**.
- **Common Algorithms:** FCFS (First Come First Served), SJF (Shortest Job First), and Round Robin (Time Slicing)

b. Virtual Memory

Virtual Memory is a memory management technique that creates an illusion for users that they have a very large main memory, even if the physical RAM is small.

- **How it works:** It allows the execution of processes that are not completely in the main memory. Only the "currently needed" parts (pages) are kept in RAM, while the rest stay on the hard disk.
- **Demand Paging:** This is the most common implementation where a page is brought into RAM only when it is actually requested (on demand).
- **Benefits:** * Allows programs larger than physical RAM to run.
  - Increases the degree of multiprogramming.[5]
- **Key Concept:** When a process tries to access a page not in RAM, a **Page Fault** occurs, and the OS fetches it from the disk.

c. Thread vs. Process

While both are units of execution, they differ significantly in how they share resources:

Feature	Process	Thread
Definition	An independent program in execution.	A "lightweight" unit of execution within a process.
Memory	Has its own independent memory space.	Shares the memory space of its parent process.
Creation	Expensive (takes more time/resource).	Economical (fast to create).
Communication	Needs IPC (Inter-Process Communication).	Can communicate directly (shared memory).
Isolation[6]	One process crashing doesn't affect others.[7]	If one thread crashes, the whole process may die.

---

d. Interrupt Handlers

An **Interrupt Handler**, also known as an **Interrupt Service Routine (ISR)**, is a specialized block of code in the OS that is executed when a specific interrupt occurs.

- **The Process:** 1. When a hardware device (like a keyboard or disk) needs attention, it sends an **Interrupt Signal** to the CPU.

2. The CPU suspends its current task and saves its state.[8]

3. The CPU looks up the **Interrupt Vector Table** to find the address of the correct Interrupt Handler.

4. The Handler executes (e.g., reads the key press).[9]
5. The CPU restores the saved state and resumes the previous task.[10]

- **Importance:** They allow the OS to handle asynchronous events (like I/O or timers) without making the CPU wait idly.

REGULAR/BACK EXAM-2078, KARTIK/MANGSIR

1.Define operating system. Explain OS as a resource manager.

An **Operating System (OS)** is a piece of system software that acts as an **intermediary (interface)** between a computer user and the computer hardware. It manages all the hardware and software components, providing a platform where application programs can run efficiently.

2. Operating System as a Resource Manager

In a computer system, resources like the CPU, RAM, and Disk space are limited, but multiple programs (processes) often want to use them at the same time.[4] The OS acts as a "Manager" or "Allocator" to handle these conflicting requests.[5]

The OS manages resources in two primary ways:[6]

- **Time Multiplexing (Sharing in Time):** Different programs take turns using a resource.[7]
  - *Example:* The CPU switches between different apps so fast that they seem to run simultaneously.
- **Space Multiplexing (Sharing in Space):** A resource is divided into parts, and each program gets a piece.[8]
  - *Example:* Main memory (RAM) is divided so multiple programs can stay in it at once.[9]

Key Resource Management Functions:

Resource	How the OS Manages It
Process (CPU)	The OS uses **Scheduling Algorithms** to decide which process gets the CPU, for how long, and when.
Memory (RAM)	The OS tracks which parts of memory are in use and by whom. It allocates and de-allocates space as programs start and close.
Device (I/O)	If two programs want to use the printer, the OS "spools" the jobs in a queue to prevent a crash. It manages all peripherals via device drivers.
Storage (File)	The OS organizes data into files and directories, manages disk space, and controls who has permission to read/write them.
Security	It ensures that one process cannot access the data or memory of another process without permission.

2 What are process and thread? Explain inter process communication.

1. Process and Thread

What is a Process?

A **Process** is an instance of a program in execution. It is a "heavyweight" unit that has its own independent address space, memory, and resources.

- **Isolation:** If one process crashes, it generally does not affect others.
- **Components:** It contains the program code, data, heap (for dynamic memory), and stack (for local variables).

What is a Thread?

A **Thread** is a "lightweight process" or a basic unit of CPU utilization. It exists **within a process**. A single process can have multiple threads, all performing different tasks at the same time.

- **Sharing:** Threads within the same process share the same code section, data section, and OS resources (like open files).
- **Efficiency:** Creating and switching between threads is much faster than processes because they don't need their own separate memory space.

3. Inter-Process Communication (IPC)

Since processes are isolated and cannot "see" each other's memory, the Operating System provides **IPC mechanisms** to allow them to exchange data and synchronize their actions.
There are two primary models of IPC:

A. Shared Memory Model

- The OS creates a specific region of memory that two or more processes can access.
- **Advantage:** It is the **fastest** method because once the memory is mapped, the OS is no longer involved in the data transfer.
- **Drawback:** Processes must manage their own synchronization (e.g., using **Semaphores**) to prevent two processes from writing to the same spot at once.

B. Message Passing Model

- Processes communicate by sending and receiving "messages" through a kernel-managed buffer (a Message Queue).
- **Advantage:** No synchronization is required by the programmer because the OS handles the queuing and delivery. It is great for small amounts of data or communication across a network.
- **Drawback:** It is **slower** than shared memory because every message requires a system call to the kernel.

3 .What is mutual exclusion? Explain Dekker's or Peterson's algorithm.

Mutual exclusion is a property of concurrent systems that ensures only one process/thread enters the critical section at a time.
It prevents race conditions when multiple processes access shared resources.

Achieved using synchronization techniques like semaphores, locks, or algorithms (Dekker's, Peterson's).

Dekker's Algorithm

Dekker's algorithm ensures that two processes do not enter their critical sections at the same time by using two variables: a turn variable and a flag array.

Shared Variables:

1. **boolean flag[2]**: If flag[0] is true, Process 0 wants to enter the critical section.
2. **int turn**: Holds the value (0 or 1) indicating which process has the "right of way" when both want to enter at the same time.

The Algorithm Logic (for Process 0):

C

```
flag[0] = true;            // Process 0 expresses interest
while (flag[1]) {          // Check if Process 1 also wants in
   if (turn != 0) {        // If it's not Process 0's turn...
      flag[0] = false;     // ...temporarily withdraw interest
      while (turn != 0);   // Wait until it's officially Process 0's turn
      flag[0] = true;      // Re-assert interest
   }
}
// --- CRITICAL SECTION ---

turn = 1;                  // After finishing, give turn to Process 1
flag[0] = false;           // Mark Process 0 as no longer interested
```

Dekker's Algorithm (Older Solution)

Definition:

Dekker's Algorithm was the first known software solution for mutual exclusion between two processes.

Key Idea:

- Each process sets its flag when it wants to enter.
- If both want to enter, the turn variable decides who goes first.
- Uses repeated checks and turn alternation to ensure fairness.

Drawback:

- More complex than Peterson's Algorithm.
- Rarely used today, but historically important.

4 Define deadlock. Explain Ostrich algorithm for deadlock handling. Describe deadlock detection and recovery

A **Deadlock** is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource that is being held by another process in the same set. In this state, none of the processes can ever complete, and they remain in a "Wait" state indefinitely until the OS or a user intervenes.

Ostrich Algorithm for Deadlock Handling.

- Definition: The Ostrich Algorithm is a deadlock handling strategy where the operating system ignores deadlocks instead of preventing or detecting them.
- Reasoning: Deadlocks are rare in many systems, and the cost of prevention/detection may be higher than the damage caused.
- Analogy: Like an ostrich burying its head in the sand, the OS pretends deadlocks don't exist.
- Example: UNIX and Windows often adopt this approach because deadlocks are infrequent and recovery is costly.
- Drawback: If a deadlock occurs, the system may freeze until manual intervention.

Deadlock Detection and Recovery

Detection

- The OS allows deadlocks to occur but uses algorithms to detect them.
- Methods:
- Wait-for Graph: Detects cycles in resource allocation.
- Resource Allocation Graph (RAG): If a cycle exists, deadlock may be present.
- Detection is done periodically or when resource requests fail.

Recovery

Once detected, the OS must break the deadlock:

1. Process Termination
- Kill one or more processes involved in deadlock until it breaks.
- Simple but may cause data loss.

2. Resource Preemption
- Temporarily take resources away from some processes and give them to others.
- Requires rollback and restart of affected processes.

3. Manual Intervention
- In some systems, administrators must manually terminate or restart processes..

5.Define process scheduling. Explain Round Robin Scheduling.

Definition

Process scheduling is the method by which the operating system decides which process runs on the CPU at a given time.It ensures efficient CPU utilization, fairness among processes, and responsiveness for users.

Round Robin Scheduling

Definition:

Round Robin (RR) is a preemptive scheduling algorithm where each process is given a fixed time slice (quantum) in cyclic order.

Explanation:

- The ready queue is maintained as a FIFO queue.
- Each process gets the CPU for one time quantum.
- If the process finishes within the quantum → it leaves the system.
- If not → it is preempted and placed at the end of the queue.
- This cycle continues until all processes are executed.

Advantages:

- Fair: Every process gets equal CPU time.
- Simple to implement.
- Good for time-sharing systems.

6.Describe virtual memory. Differentiate between contiguous and noncontiguous memory allocation

irtual memory is a memory management technique that gives processes the illusion of having a large, continuous memory space, even if physical RAM is limited.

It uses paging or segmentation to map logical addresses to physical memory.

This allows execution of large programs and efficient use of RAM.

Difference between Contiguous and Non-Contiguous Memory Allocation

Contiguous Allocation	**Non-Contiguous Allocation**
1. Entire process stays in one continuous memory block.	**Process is divided and stored in different memory locations.**
2.High memory waste due to internal and external fragmentation.	**Very low waste; no external fragmentation.**
3.Simple address mapping using base and limit registers.	**Complex mapping using page tables or segment tables.**
4.Faster access because of direct memory addressing.	**Slower due to address translation overhead.**
5.Poor memory utilization.	**Highly efficient memory utilization.**
6.Rigid; needs one large free contiguous block.	**Flexible; can use small free memory gaps.**
7.Fixed and variable partitioning.	**Paging and segmentatio**

7 .What is segmentation. Write down the importance and drawbacks of segmentation.
Definition.
Segmentation is a memory management technique where a program is divided into logical segments such as code, data, and stack.
Each segment has a variable size and is identified by a segment number + offset.
It provides a logical view of memory rather than a purely physical one.
Importance of Segmentation
- Logical Division: Breaks program into meaningful units (code, data, stack).
- Flexibility: Each segment can grow or shrink independently.
- Protection: Provides segment-wise protection and controlled sharing.
- Modularity: Supports modular programming and simplifies debugging.
- Efficient Use: Allocates only the memory needed for each segment.
- Supports Virtual Memory: Works well with demand paging and dynamic allocation.

Drawbacks of Segmentation
- External Fragmentation: Free memory may be scattered, making allocation difficult.
- Complex Management: Requires segment tables and hardware support, increasing overhead.
- Variable Sizes: Handling variable segment sizes is more complicated than fixed partitions.
- Performance Overhead: Address translation (segment number + offset) can slow execution.
- Less Popular Today: Modern systems often prefer paging due to simplicity.

8 Explain DMA. Differentiate between memory mapped IO and 10 mapped 10. Alright
Definition.
DMA is a technique that allows I/O devices to transfer data directly to/from main memory without continuous CPU involvement.The CPU only initiates the transfer, and the DMA controller manages the rest.
Differentiate between memory mapped IO and 10 mapped 10.

S.N.	Feature	Memory Mapped I/O	I/O Mapped I/O (Isolated)
1	**Address Space**	I/O devices and RAM share the **same** address space.	I/O devices have a **separate** address space from RAM.
2	**Instructions**	Uses standard memory instructions like **LOAD/STORE**.	Uses special I/O instructions like **IN/OUT**.
3	**Address Lines**	Uses the same set of address lines for both.	Requires a special control signal to distinguish I/O.
4	**RAM Capacity**	**Reduces** the total addressable RAM for programs.	**Does not affect** the available RAM capacity.
5	**Complexity**	Hardware design is **simpler**.	Hardware design is **more complex**.
6	**Flexibility**	More flexible; can use all memory manipulation tricks.	Limited to specific I/O instructions.

S.N.	Feature	Memory Mapped I/O	I/O Mapped I/O (Isolated)
7	**Bus Usage**	A single set of Read/Write control lines.	Separate Read/Write control lines for I/O.

9 Write short notes on

a) Disk Scheduling

Disk scheduling is the process by which the operating system manages the order of I/O requests arriving for the disk. Since the disk head moves slowly, the OS reorders requests to minimize "Seek Time."

- **Purpose:** To reduce the movement of the disk arm and improve system throughput.
- **Key Terms: Seek Time** (time to move the head to the right track) and **Rotational Latency** (time for the disk to rotate to the right sector).
- **Common Algorithms:**
  - **FCFS:** First-Come, First-Served (Fair but slow).
  - **SSTF:** Shortest Seek Time First (Selects the closest request; may cause starvation).
  - **SCAN (Elevator):** Moves head from one end to the other, servicing requests along the way.
  - **C-SCAN:** Circular SCAN; moves in one direction only, then jumps back to the start.

b) Optimal Page Replacement Algorithm

This is a theoretical algorithm used for page replacement in virtual memory. It is often used as a "benchmark" to measure how well other algorithms (like FIFO or LRU) are performing.

- **The Rule:** Replace the page that will **not be used for the longest period of time** in the future.
- **Advantages:** It provides the **lowest possible page-fault rate** for a fixed number of frames. It is guaranteed to be "optimal."
- **Drawback:** It is impossible to implement in a real-world system because the OS cannot predict the future memory requirements of a program.
- **Belady's Anomaly:** Unlike FIFO, the Optimal algorithm never suffers from Belady's Anomaly (where adding more frames actually increases page faults).

c) Memory Hierarchy

Memory hierarchy is a pyramid-like structure that organizes computer storage based on response time, cost, and capacity.

- **Top (Fastest/Smallest/Costliest):** Registers (inside CPU) and Cache Memory (L1, L2, L3).
- **Middle:** Main Memory (RAM).
- **Bottom (Slowest/Largest/Cheapest):** Secondary Storage (HDD, SSD) and Magnetic Tapes.
- **Goal:** To provide the CPU with the fastest possible access to data at the lowest average cost. As you move **down** the hierarchy, capacity increases but speed decreases.

d) File Operations

A file is an abstract data type managed by the OS. To handle files, the Operating System provides several system calls (operations):

1. **Create:** Finding space on the disk and adding an entry to the directory.
2. **Open:** Bringing the file's attributes and disk addresses into RAM for faster access.
3. **Read:** Taking data from the file at the current "file pointer" position.
4. **Write:** Adding new data to the file or overwriting existing data.
5. **Delete:** Removing the file from the directory and freeing up the disk space.
6. **Seek (Repositioning):** Moving the file pointer to a specific place within the file without reading or writing

REGULAR/BACK EXAM-2076, FALGUN/CHAITRA

1. Explain batch systems. Time sharing systems and real time systems in brief.

Batch Systems

• Definition: In batch systems, jobs are collected, grouped, and executed one after another without user interaction.

• Working: Users submit jobs → OS queues them → executes sequentially.

• Advantages: Efficient for large, repetitive tasks; reduces idle CPU time.

• Drawbacks: No immediate user interaction; debugging is difficult.

• Example: Payroll processing, bank cheque processing.

Time-Sharing Systems

• Definition: Time-sharing systems allow multiple users to share the CPU simultaneously by giving each process a small time slice (quantum).

• Working: CPU switches rapidly among processes, creating the illusion that all run at the same time.

• Advantages: Interactive, fair CPU distribution, supports multiprogramming.

• Drawbacks: Performance depends on quantum size; context switching overhead.

• Example: Modern multi-user systems, university servers.

Real-Time Systems

• Definition: Real-time systems are designed to respond to inputs or events within a strict time deadline.

Types:

• Hard Real-Time: Missing a deadline can cause system failure (e.g., flight control).

• Soft Real-Time: Occasional deadline misses are tolerable (e.g., video streaming).

• Advantages: Predictable, reliable, ensures timely response.

• Drawbacks: Complex design, expensive hardware/software.

• Example: Air traffic control, medical monitoring systems.

2. Define preemptive and non-preemptive scheduling. Differentiate between process and thread. Explain the states of a process.

1. Preemptive vs. Non-Preemptive Scheduling

These terms describe how the OS decides when to take the CPU away from a running process.

- **Non-Preemptive Scheduling:** Once a process starts running, it keeps the CPU until it either finishes (terminates) or switches to a "Waiting" state (e.g., for I/O).[1] The OS cannot force it to stop.[2]
    - *Example:* First-Come First-Served (FCFS).
- **Preemptive Scheduling:** The OS can interrupt a running process and give the CPU to another process at any time (e.g., when a higher-priority process arrives or a time slice ends).[3]

- *Example:* Round Robin (RR), Shortest Remaining Time First (SRTF).[4]

---

2. Differentiate between Process and Thread

Feature	Process	Thread
Definition	A program in execution (Heavyweight).	A segment of a process (Lightweight).
Memory	Has its own independent address space.	Shares memory with other threads of the same process.
Creation	Takes more time to create and terminate.	Much faster to create and terminate.
Context Switch	Slow (requires saving/loading all data).	Fast (minimal data to save).
Communication	Difficult; needs IPC (Message passing).	Easy; threads talk directly via shared memory.
Independence	One process crash does not affect others.	One thread crash can kill the whole process.

3. States of a Process (Process Life Cycle)

When a process is executed, it moves through different states. This is often called the **Process State Transition Diagram**.

1. **New:** The process is being created (e.g., loading from disk).
2. **Ready:** The process is in RAM and waiting for the CPU to pick it.
3. **Running:** The CPU is currently executing the process instructions.
4. **Waiting (Blocked):** The process is stuck waiting for an event, such as a user clicking a button or a file being read from the disk.
5. **Terminated:** The process has finished execution and is being removed from memory.[5]

Transitions:

- **Ready $\rightarrow$ Running:** The Scheduler picks the process.[6]
- **Running $\rightarrow$ Ready:** An interrupt occurs or the time slice ends (Preemption).[7]
- **Running $\rightarrow$ Waiting:** The process asks for I/O.
- **Waiting $\rightarrow$ Ready:** The I/O task is finished.

- **3 or 4 qeustion coming.**

5. Define logical and physical memory. Differentiate between fixed and variable partition multiprogramming.

Logical vs. Physical Memory.

- **Logical Memory (Virtual Address):** This is the memory address generated by the **CPU** while a program is running. The programmer or the compiler sees this as a flat, continuous space. It doesn't represent an actual location in the RAM.
- **Physical Memory (Physical Address):** This is the actual location in the **RAM hardware** where data is stored. It is the address seen by the memory unit (hardware).
- **Fixed vs. Variable Partitioning (Deep Comparison)**

S.N.	Feature	Fixed (Static) Partitioning	Variable (Dynamic) Partitioning
1	**Memory Division**	Memory is divided into partitions **before** any process is loaded.	Memory is divided **during** process loading based on demand.
2	**Internal Fragmentation**	**High**. Occurs when the process size is smaller than the partition.	**Zero**. Every process gets exactly the amount of memory it asks for.
3	**External Fragmentation**	**Low**. Only happens if a process is larger than any available slot.	**High**. Small, unusable holes are created as processes enter and exit.
4	**Degree of Multiprogramming**	**Fixed**. It is limited by the number of partitions (e.g., 5 slots = 5 processes).	**Variable**. Limited only by total RAM capacity.
5	**Process Size Limit**	A process cannot be larger than the **largest available partition**.	A process can be as large as the **total free memory**.
6	**Management Effort**	**Simple**. The OS only needs a table of "free" or "busy" slots.	**Complex**. The OS must track "holes" using algorithms like **First-Fit or Best-Fit**.
7	**Compaction**	Not applicable/Not needed.	**Required periodically** to merge small holes into one large bloc

6. Define segmentation. Explain briefly segmentation with paging.

Segmentation is a memory management technique where a program is divided into logical segments such as code, data, and stack.Each segment has a variable size and is identified by a segment number + offset.It provides a logical view of memory rather than a purely physical one.

Segmentation with Paging

Explain briefly segmentation with paging.

Segmentation with paging is a hybrid memory management technique that combines the benefits of segmentation and paging.

Explanation:

- Step 1 (Segmentation): Program is divided into logical segments (code, data, stack).
- Step 2 (Paging): Each segment is further divided into fixed-size pages.
- Mapping: Logical address → (segment number, page number, offset).
- Advantages:
- Eliminates external fragmentation (due to paging).
- Retains logical division (due to segmentation).
- Provides protection and sharing at segment level.

7. Explain about file allocation methods. Describe tertiary storage structure.

ile Allocation Methods

When files are stored on disk, the OS must decide how disk blocks are allocated. Common methods are:

1. Contiguous Allocation
- Each file occupies a set of contiguous blocks on disk.
- Advantages: Fast access (sequential and direct).
- Drawbacks: Suffers from external fragmentation; resizing files is difficult.

2. Linked Allocation
- Each file is a linked list of disk blocks scattered anywhere on disk.
- Directory entry stores pointer to first block; each block points to the next.
- Advantages: No external fragmentation, easy to grow files.
- Drawbacks: Slow random access; pointer overhead.

3. Indexed Allocation
- Each file has an index block containing pointers to all its disk blocks.
- Advantages: Supports direct access; efficient for large files.
- Drawbacks: Overhead of index block; may waste space if file is small.

Tertiary Storage Structure.

Tertiary Storage refers to a level in the storage hierarchy below secondary storage (HDDs/SSDs).[11] It involves high-capacity, removable media used for archiving data that is rarely accessed.

Key Components:

- **Removable Media:** The most common examples are **Magnetic Tapes** and **Optical Disks** (CDs, DVDs, Blu-rays).[12]
- **Robotic Libraries (Jukeboxes):** In large data centers, a robotic arm picks a tape or disk from a shelf and loads it into a drive when the data is requested.

Characteristics:

1. **Massive Capacity:** It can store petabytes ($1,000s$ of terabytes) of data.
2. **Low Cost:** It is much cheaper per gigabyte than SSDs or HDDs.
3. **High Latency:** It is very slow. It can take seconds or minutes to retrieve data because the media must be physically loaded and "spun up."
4. **Non-Volatile:** Data remains safe for decades without power.

8. Explain disk structure, scheduling and management of mass-storage device.

Disk Structure

• A disk is divided into platters, each with concentric circles called tracks.
• Tracks are further divided into sectors (smallest storage unit).
• Cylinder: Set of tracks aligned vertically across platters.
• Disk Addressing: Data is accessed using cylinder number, track number, and sector number.
• Access Time = Seek Time + Rotational Latency + Transfer Time.

Disk Scheduling

Disk scheduling decides the order in which disk I/O requests are serviced to minimize seek time.

Common Algorithms:

1. FCFS (First Come First Serve):
   - Requests served in arrival order.
   - Simple but may cause long average seek time.
2. SSTF (Shortest Seek Time First):
   - Chooses request closest to current head position.
   - Reduces seek time but may cause starvation.
3. SCAN (Elevator Algorithm):
   - Head moves back and forth across disk, servicing requests in order.
   - Fairer, reduces variance in response time.
4. C-SCAN (Circular SCAN):
   - Head moves in one direction only, then jumps back.
   - Provides uniform wait time.
5. LOOK / C-LOOK:
   - Similar to SCAN/C-SCAN but head only goes as far as last request, not end of disk.

Management of Mass-Storage Devices

The OS must manage the disk from the moment it is plugged in until it is full of files.

A. Disk Formatting

- **Low-Level Formatting (Physical):** Divides the disk into sectors that the disk controller can read.
- **High-Level Formatting (Logical):** The OS creates the file system structures (like the FAT or NTFS tables) and divides the disk into "clusters."

B. Boot Block

A tiny protected area of the disk that contains the **Bootstrap Loader**. This is the first code the computer runs to start the Operating System when you turn on the power.

C. Bad Block Management

Over time, parts of a disk can physically fail. The OS and the disk controller maintain a list of these "Bad Blocks" and map them to "Spare Sectors" so that the user never loses data due to a tiny scratch on the platter.

D. Swap-Space Management

The OS uses a portion of the disk as an extension of RAM. This is called **Swap Space**. It is managed differently than normal files to ensure maximum speed for Virtual Memory operations.

9. Explain Inter process communication (IPC) in brief. And Also explain classical IPC problems.

Inter-Process Communication (IPC)

Inter-Process Communication (IPC) is a mechanism that allows processes to exchange data and synchronize actions.

Since processes are independent, IPC ensures cooperation and safe sharing of resources.

It is essential in multiprogramming and distributed systems.

Methods of IPC:

- Shared Memory: Processes share a region of memory; fast but needs synchronization.
- Message Passing: Processes communicate by sending/receiving messages via OS.
- Pipes: Unidirectional communication channel between processes.
- Sockets: Communication across networks (client-server).
- Signals: Notifications sent to processes (e.g., interrupts).

Classical IPC Problems

1.Producer–Consumer Problem (Bounded Buffer)

• Producer puts items into a buffer, consumer takes them out.

• Problem: Producer must not overwrite full buffer, consumer must not read empty buffer.

•Solution: Use semaphores/locks to synchronize.

2.Readers–Writers Problem

• Many readers can read a database at the same time, but writers need exclusive access.

• Problem: Balance between readers and writers, avoid starvation.

• Solution: Synchronization ensures mutual exclusion for writers, shared access for readers.

3. **Dining Philosophers Problem**

- Five philosophers sit at a table with five chopsticks; each needs two to eat.
- Problem: If all pick one chopstick, deadlock occurs.
- Solution: Careful resource allocation or semaphore rules to avoid deadlock/starvation.

10. Write short notes on:

a) Threads (The "Lightweight" Workers)

Think of a **Process** like a whole factory, and a **Thread** like a single worker inside that factory. A process (like your Chrome browser) can have many threads (one for your YouTube video, one for the scroll bar, and one for the download).[1]

- **Why we use them:** They are much "lighter" than processes.[2] Since all workers (threads) are in the same factory (process), they share the same tools and materials (memory/RAM).
- **The Benefit:** It's way faster for the CPU to switch between two workers in the same room than to shut down one factory and open another. This makes your apps feel smooth and responsive.

- **The Risk:** If one worker (thread) makes a huge mistake and sets the factory on fire, the whole process crashes.

---

b) Direct Mapping (The Cache "Address Book")

The CPU is incredibly fast, but RAM is slow.[3] To fix this, we use a small, super-fast memory called **Cache**. **Direct Mapping** is the simplest way the CPU finds data in that cache.

- **How it works:** Imagine a huge library (RAM) and a small bookshelf (Cache). In Direct Mapping, every book in the library has one *specific* spot on the bookshelf where it is allowed to go.
- **The Logic:** We use a simple formula (usually [4]$Address\ mod\ Number\ of\ Cache\ Blocks$) to decide where a piece of data goes.[5]
- **The Upside:** It's very fast and cheap to build because the CPU knows exactly where to look for a specific address.[6]
- **The Downside:** If two different books (data) want the *same* spot on the bookshelf, they keep kicking each other out. This is called "Conflict Misses," and it can slow things down.

---

c) Swap-Space Management (The "Overflow" Area)

Think of **Swap-Space** as the "Emergency Room" on your Hard Drive or SSD.

- **The Purpose:** When your RAM gets completely full because you have too many tabs open, the Operating System doesn't want to crash. Instead, it takes the "oldest" or "least used" data in your RAM and moves it to a special hidden area on your disk called the Swap-Space.
- **Management:** The OS manages this space differently than your regular photos or files. It tries to make it as fast as possible, using large blocks of data so that moving things back and forth from RAM doesn't take forever.
- **The "Throttling" Effect:** If you've ever noticed your computer suddenly getting very slow and the hard drive light blinking like crazy, that's your OS "swapping." Since disks are much slower than RAM, your system feels like it's dragging

Regular/Back Exam-2075, Falgun/Chaitra

1.Describe operating system. Explain OS as virtual machine and resource manager.

An Operating System is system software that controls the hardware and provides services to application programs.It acts as a bridge between the user and the computer hardware. Without an OS, using a computer would be very difficult because users would have to deal directly with machine language and hardware details.

OS as a Virtual Machine (The "Top-Down" View)

In this context, the OS is often called an **Extended Machine**. It provides an abstraction layer that hides the complex, messy details of the hardware from the programmer and the user.

1. Hardware Abstraction

Physical hardware is difficult to interact with. For example, writing data to a hard drive involves controlling motor speeds, moving a physical disk head, and managing sectors. The OS abstracts this into a **"Virtual File System."** You simply see "Save Document," while the OS handles the mechanical complexity.

2. Simplification of Complexity

The OS provides a set of **System Calls** (an Interface). Instead of writing thousands of lines of code to talk to a network card, a programmer uses a simple command provided by the OS. This makes the computer feel like a much more "user-friendly" machine than it actually is.

3. The "Pseudo-Machine" Concept

The OS creates an environment where multiple programs can run simultaneously. To each program, it feels like it has:

- Its own dedicated CPU.
- Its own private memory space.
- Its own set of I/O devices. In reality, these are "virtual" versions of the hardware managed by the OS.

OS as a Resource Manager

The computer system has limited resources: CPU, memory, storage, and I/O devices.

The OS acts like a manager that controls and allocates these resources efficiently among multiple users and programs.

Functions of Resource Management

1. **CPU Management**

• The OS schedules processes using algorithms (FCFS, Round Robin, Priority Scheduling).

- Ensures fair CPU time and maximizes utilization.

2. **Memory Management**

- Allocates memory to processes.
- Uses techniques like paging, segmentation, and virtual memory.
- Prevents memory conflicts and ensures efficient usage.

3. **File Management**

- Organizes data into files and directories.
- Controls access rights (read, write, execute).
- Handles storage allocation and retrieval.

4. **Device Management**

- Uses device drivers to communicate with hardware.
- Manages input/output requests using buffering and spooling.

5. **Security & Protection**

- Ensures only authorized users can access resources.
- Provides mechanisms like passwords, encryption, and access control lists.

2.Differentiate between process and threads. Explain Dekker's Algorithm.

No.	Feature	Process	Thread
1	**Definition**	A program in execution with its own address space.	A segment or "unit of execution" within a process.
2	**Memory**	Has its own **private memory** (Stack, Heap, Data).	Shares the **parent process's memory** and resources.
3	**Weight**	**Heavyweight**: Creating a process requires high overhead.	**Lightweight**: Creating a thread is much faster and cheaper.
4	**Isolation**	**Independent**: If one process crashes, others continue.	**Interdependent**: If one thread crashes, the whole process may die.
5	**Switching**	**Slow Context Switching**: The OS must swap out entire memory maps.	**Fast Context Switching**: The OS only needs to swap registers and stack.
6	**Comm.**	Requires **IPC** (Inter-Process Communication) like pipes or signals.	Can communicate directly via **shared variables** in the same memory.
7	**Resources**	Uses more system resources (Memory, I/O handles).	Uses very few resources; shares the process's assets.
8	**ID**	Each has a unique **PID** (Process ID).	Each has a unique **TID** (Thread ID) but shares the same PID.

b. Explain Dekker's Algorithm.

Dekker's Algorithm was the first known correct solution to the **Critical Section Problem** for two processes. It allows two processes to share a single resource (like a printer or a variable) without conflict, ensuring **Mutual Exclusion**.

The Problem: Mutual Exclusion

When two processes want to access the same data at once, it can lead to a "Race Condition." We need a way to ensure that if Process 0 is in the "Critical Section," Process 1 must wait.

The Key Components

Dekker's uses three variables to coordinate:

1. wants_to_enter[0]: Boolean (True/False) – Does Process 0 want to enter?
2. wants_to_enter[1]: Boolean (True/False) – Does Process 1 want to enter?
3. turn: Integer (0 or 1) – Whose turn is it to break a "tie"?

How the Algorithm Works (The Logic)

If both processes want to enter at the exact same time, Dekker's uses the turn variable to decide who wins.

1. Process P_0 sets its wants_to_enter[0] to **True**.
2. It checks if P_1 also wants to enter.
3. If P_1 wants to enter, P_0 checks whose turn it is.
4. If it is P_1's turn, P_0 politely sets its own "want" to **False** and waits until the turn changes.
5. Once it is P_0's turn, it enters the Critical Section.
6. Upon leaving, it sets the turn to P_1 and sets its "want" to **False**.

Why it's important for exams:

Dekker's Algorithm satisfies the three main requirements for synchronization:

- **Mutual Exclusion:** Only one process enters at a time.
- **Progress:** If no one is in the critical section, someone can enter.
- **Bounded Waiting:** No process waits forever (no starvation).

3.Explain virtual exclusion, context switching, critical section, race condition with examples.

Mutual Exclusion (The Principle of Safety)

Mutual Exclusion is the core requirement for any synchronized system. It dictates that if one process is executing in its **Critical Section**, no other process can be allowed to enter its own critical section at the same time.

- **The Problem:** Without it, shared data becomes corrupted because multiple processes might try to write to the same memory address simultaneously.
- **The Solution:** Software solutions (like **Dekker's** or **Peterson's Algorithm**) or hardware solutions (like **Test-and-Set** instructions) are used to "lock" the section.
- **Key Requirements:** For a solution to be valid, it must provide:
    1. **Mutual Exclusion:** Only one process at a time.

2. **Progress:** If no one is in the section, the process that wants to enter should be allowed to.
3. **Bounded Waiting:** No process should wait forever to enter (prevents starvation).

2. Context Switching (The Mechanism of Multitasking)

Context Switching is the administrative work the OS performs to switch the CPU from one process to another. Since a CPU core can only handle one instruction stream at a time, the OS must "save and swap" very quickly to give the illusion of simultaneous execution.

- **The "Context":** This includes the CPU registers, Program Counter (which instruction is next), Stack Pointer, and Memory Management info. All of this is stored in the **PCB (Process Control Block)**.
- **The Steps:**
  1. Save the context of Process A into its PCB.
  2. Load the context of Process B from its PCB.
  3. The CPU starts executing Process B where it last left off.
- **Performance Impact:** Context switching is considered **overhead**. It consumes CPU cycles without doing "real" work. High frequency of switching can lead to **thrashing**, where the OS spends more time switching than actually running programs.

3. Critical Section (The Danger Zone)

In any multi-threaded or multi-process program, the **Critical Section** is the specific part of the code where shared resources are accessed.

- **Components of a Synchronized Process:**
  - **Entry Section:** The "waiting room." The process asks for a lock/permission here.
  - **Critical Section:** The actual code that modifies a shared variable, updates a database, or writes to a file.
  - **Exit Section:** The "release." The process gives up the lock so others can enter.
  - **Remainder Section:** The rest of the code that doesn't use shared resources.
- **Example:** In a printer-sharing program, the few lines of code that actually send data to the printer buffer represent the Critical Section.

4. Race Condition (The Conflict)

A **Race Condition** is an undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time, but because of their nature, the operations must be done in the proper sequence to be done correctly.

Detailed Example: The "Lost Update" Problem

Imagine two threads are trying to increment a shared variable Counter, which currently equals **10**.

1. **Thread A** fetches Counter (10).
2. **Thread B** fetches Counter (10).
3. **Thread A** increments it (11) and saves it.
4. **Thread B** increments its own copy (11) and saves it.

- **Result:** The counter is **11**, but it should be **12** (since two increments happened). Because Thread B finished its "race" slightly after Thread A, it overwrote Thread A's work.

4. Define preemptive and non- preemptive scheduling. Explain shortest job first algorithm with example.

Preemptive Scheduling

Preemptive scheduling is a mechanism where the OS can interrupt a currently running process to assign the CPU to another process, usually one with a higher priority. This ensures that no single task monopolizes the processor, allowing for better responsiveness in multi-user systems. However, it requires frequent context switching, which adds some computational overhead to the system.

Non-Preemptive Scheduling

Non-preemptive scheduling is a method where once a process starts executing, it holds the CPU until it finishes its task or voluntarily switches to a waiting state. The OS cannot force the process to stop, making it simpler to implement and reducing context-switching costs significantly. The downside is that a long-running process can cause others to wait, leading to "starvation."

Shortest Job First (SJF) Algorithm

Shortest Job First (SJF) is a scheduling policy that selects the process with the smallest execution time (Burst Time) for the next execution.

Key Characteristics:

- It can be **Non-Preemptive** (standard SJF) or **Preemptive** (known as Shortest Remaining Time First).
- It is **Optimal**: It gives the minimum average waiting time for a given set of processes.
- **Challenge:** It is difficult to implement in real-time because the OS doesn't always know exactly how long a process will take before it starts.

Example of Non-Preemptive SJF

Let's look at four processes arriving at the same time ($T=0$) with different burst times:

Process	Burst Time (ms)
P_1	6
P_2	8
P_3	7
P_4	3

Step 1: The Execution Order (Gantt Chart)

The SJF algorithm looks at the burst times and picks the shortest one first.

1. **P_4** (3ms) goes first.
2. **P_1** (6ms) goes second.
3. **P_3** (7ms) goes third.
4. **P_2** (8ms) goes last.

Step 2: Calculating Wait Time

Waiting time is the time a process spends in the ready queue before its execution starts.

- **P_4 Wait Time:** 0 ms
- **P_1 Wait Time:** 3 ms
- **P_3 Wait Time:** 3 + 6 = 9 ms
- **P_2 Wait Time:** 3 + 6 + 7 = 16 ms

Average Waiting Time: $\frac{0 + 3 + 9 + 16}{4} = \frac{28}{4} = \mathbf{7 \text{ ms}}$

6.Explain deadlock. Write short notes on starvation.

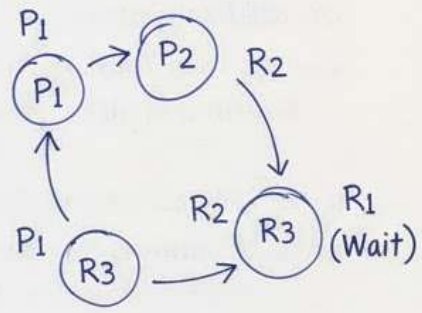
6. Explain deaddock. Write short notes on starvation.

1. Deadlock

Deadlock is state in ia multi-tasking system where two oire processes are pocesses ne indfinitely, indiitiely, each waiting for roree of resource held by anrother process. This causes a permiution, making a system system unpoopsive.

In a deadlock situation, processes enter depaddency, preventing cirreating cannom system resolve this own, often requiring often requiing rails performance, leading bor low resource and system hang.

This severely degetem cannoit any exceutin, acut nrolve in often requitem restaart restast laie low resource utilzation a system hang.



Resource Allocation Graph showing Circular Wait

Necessary Conditions for Deadlock

a) Mutual Exclusion: Resource can only used io one process in atime.
- b) Hold & Wait: A process holds least one one resource.
- b) No Preemption: Resources cannot forally taken; mother voltuarlly.
- d) No Preemttion: A closed chain of processs of ruch mistrty prcees an priting a other is move.
- d) Circular Wait: A processs needing exist. OS Example: Two procesive access and scanner management policies.

2. Starvation (Indeniite Blocking)

Starvation is problem where process repeately denied access of resoures, like continouls other resource or CPU time, even though other processes, favorir due of high-river to run if high-priority process might neve keep arriving, insluade unfair resource albaghir might never joive.

Causple : In Shortest Job First (SF) shechecbuling, if short jobs is short kep a long job might never execute.

Difference: Deadlock is permanent wait; Starvation is indenitte wait dee unfarness.