What is an Operating System? Explain its objectives and functions.

An **Operating System (OS)** is a software that manages computer hardware, software, and resources. It acts as a bridge between the user and the system.

Objectives of an OS:

1. **Convenience:** Makes computing easier for users.
2. **Efficiency:** Optimizes hardware resource usage.
3. **Security:** Protects system data from unauthorized access.
4. **Ability to Evolve:** Supports hardware and software updates.

Functions of an OS:

1. **Process Management:** Controls execution of programs.
2. **Memory Management:** Allocates and deallocates memory efficiently.
3. **File System Management:** Organizes and secures stored data.
4. **Device Management:** Controls and coordinates hardware devices.
5. **Error Handling:** Detects and handles system errors.

2.

Types of Operating Systems:

1. **Batch Operating System**
2. **Time-Sharing Operating System**
3. **Distributed Operating System**
4. **Real-Time Operating System (RTOS)**

Explanation:

1. **Batch Operating System**
   - Jobs are collected, grouped into batches, and then executed sequentially.
   - Since **there is no user intervention**, execution is **automated and efficient**.
   - Used in environments where the same tasks (e.g., payroll processing, billing) are repeated regularly.
   - **Advantages:** Reduces CPU idle time and handles large volumes of repetitive tasks efficiently.
   - **Disadvantages:** Debugging is difficult, and turnaround time is high.
   - **Example:** IBM OS/360, early banking systems.
2. **Time-Sharing Operating System**
   - The CPU **allocates small time slices to multiple users**, allowing them to interact with the system at the same time.
   - Users get a **fast response**, and the system ensures fair CPU distribution.
   - It enables **multitasking** and **multi-user access**.
   - **Advantages:** High system utilization, interactive user experience.
   - **Disadvantages:** Requires a powerful CPU to handle multiple tasks efficiently.
   - **Example:** UNIX, Linux, Windows Server.
3. **Distributed Operating System**
   - The OS **manages multiple computers** in a network and makes them function as a **single unit**.
   - If one computer fails, others take over, ensuring **high reliability and load balancing**.
   - Used in cloud computing, supercomputers, and network-based applications.
   - **Advantages:** Improved performance, fault tolerance, and resource sharing.
   - **Disadvantages:** Requires a complex network setup and synchronization.
   - **Example:** LOCUS, Amoeba, Google Cloud OS.

4. **Real-Time Operating System (RTOS)**
    - Designed for applications where **timing and quick responses** are critical.
    - Ensures processes are completed within a **fixed deadline**.
    - Used in **medical devices, robotics, and air traffic control systems**.
    - **Types:**
        - **Hard RTOS:** Strict deadlines (e.g., pacemakers, flight navigation).
        - **Soft RTOS:** Delays are acceptable within limits (e.g., video streaming).
    - **Advantages:** Precise timing, predictable execution.
    - **Disadvantages:** Difficult to program and expensive.
    - **Example:** VxWorks, QNX, FreeRTOS.

3.

Detailed Explanation with Comparison Table

1. **Multiprogramming OS:**
    - Loads **multiple programs into memory** at once.
    - When one program is waiting for **I/O operations**, the CPU switches to another program.
    - Increases **CPU utilization** by ensuring it is never idle.
    - Example: **IBM OS/360, Batch Processing Systems.**
2. **Multitasking OS:**
    - Also called **Time-Sharing OS**, it allows multiple **tasks (processes)** to execute **simultaneously** by switching between them rapidly.
    - Each task gets a **small time slice** of CPU, giving the **illusion** that all tasks are running simultaneously.
    - Example: **Windows, Linux, macOS.**

Comparison Table:

Feature	Multitasking OS	Multiprogramming OS
Purpose	Enhances **user experience** by running multiple tasks interactively.	**Maximizes CPU usage** by loading multiple programs into memory.
Execution	Uses **time-sharing** to switch between tasks rapidly.	Executes one task at a time but switches when needed.
Process Handling	Multiple programs appear to run **simultaneously**.	CPU switches between programs **only when necessary**.
User Interaction	User interacts with multiple applications at the same time.	Designed for **batch processing** without user interaction.
Example	Windows, Linux, macOS	Early IBM systems like OS/360

Key Differences:

- **Multiprogramming** focuses on **CPU efficiency**, while **multitasking** focuses on **user experience**.
- **Multiprogramming OS** ensures that the CPU is always busy, while **multitasking OS** ensures that users can run multiple applications at once.
- **Multitasking is an extension of multiprogramming** with added user interaction.

Unit 2

Q4: What is a process? Describe the elements of Process Control Block (PCB) in detail. (4 Marks)

Definition of Process:

A **process** is an instance of a program in execution. It consists of the program code, data, and system resources needed for execution. The OS manages processes and ensures smooth execution by handling state transitions (New, Ready, Running, Waiting, Terminated).

Process Control Block (PCB):

A **Process Control Block (PCB)** is a data structure maintained by the OS to store essential details of a process. It helps in process scheduling and management.

Elements of PCB:

1. **Process ID (PID):** A unique identifier for each process.
2. **Process State:** Indicates if the process is New, Ready, Running, Waiting, or Terminated.
3. **Program Counter:** Holds the address of the next instruction to be executed.
4. **CPU Registers:** Stores register values when a process is switched out.
5. **Memory Management Information:** Tracks memory allocation details (base and limit registers).
6. **I/O Status Information:** Stores details of I/O devices assigned to the process.
7. **Accounting Information:** Keeps track of CPU usage, execution time, and process priority.

Diagram of PCB Structure:

```
 ---------------------------------
| Process ID (PID)              |
| Process State (New/Ready/Run) |
| Program Counter (Next Instr.) |
| CPU Registers                 |
| Memory Management Info        |
| I/O Status Info               |
| Accounting Information        |
 ---------------------------------
```

Q5: Explain the Two-State Model in detail with a diagram. (4 Marks)

Definition of Two-State Model:

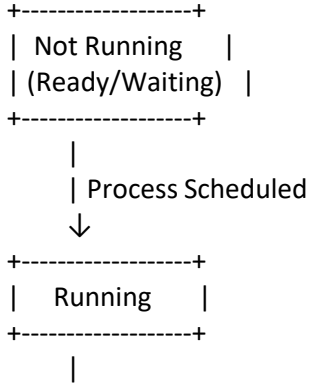
The **Two-State Model** represents process execution using two primary states:

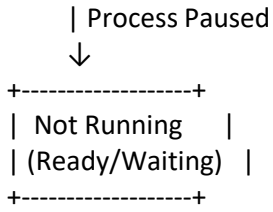
1. **Running State:** The process is currently executing on the CPU.
2. **Not Running (Waiting/Ready) State:** The process is waiting to be assigned CPU time.

Working of the Two-State Model:

- When a process is scheduled, it moves from **Not Running** → **Running**.
- When execution is paused (e.g., due to I/O operations), it moves from **Running** → **Not Running**.

Diagram:





Q6: Explain the conditions for creating new processes, process switching, and process termination. (4 Marks)

Conditions for Creating a New Process:

1. **User Request:** A user launches a new application.
2. **System Initialization:** OS starts background processes at boot.
3. **Parent Process Creation:** A process creates a child process.
4. **Service Request:** OS starts processes for system functions (e.g., print spooler).

Conditions for Process Switching:

1. **Time Slice Expiry:** In time-sharing systems, the process loses the CPU after a fixed time.
2. **I/O Request:** Process moves to the waiting state while waiting for input/output.
3. **Higher-Priority Process Arrives:** If a higher-priority process is ready, the OS preempts the current one.
4. **Completion:** Process finishes execution and releases the CPU.

Conditions for Process Termination:

1. **Normal Completion:** Process executes successfully and terminates.
2. **Error or Exception:** Due to illegal operations (e.g., divide by zero).
3. **Killed by Parent or OS:** A process may be terminated if it misbehaves or is no longer needed.
4. **Resource Unavailability:** The process cannot continue due to lack of memory or I/O.

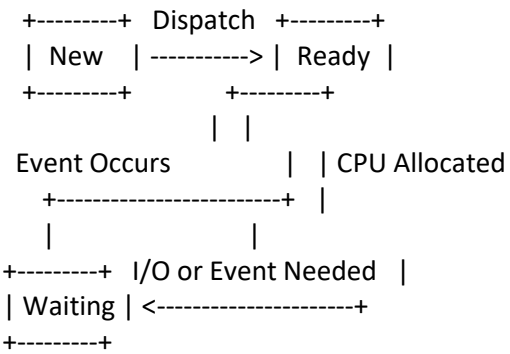
Q7: Explain the Five-State Process Model with a diagram. (4 Marks)

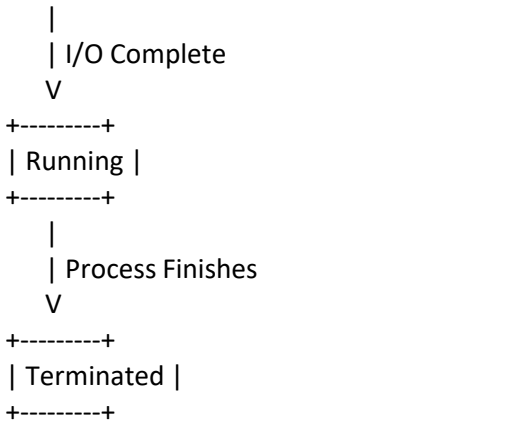
Five-State Model:

The **Five-State Model** represents different stages in process execution:

1. **New:** The process is created but not yet ready to execute.
2. **Ready:** The process is waiting in the queue to be assigned CPU.
3. **Running:** The process is currently executing on the CPU.
4. **Waiting:** The process is waiting for an I/O operation or event to complete.
5. **Terminated:** The process has finished execution and is removed from memory.

Diagram:





Q8: What is a suspended process? Explain the Process State Transition Diagram with suspended states. (4 Marks)

Definition of Suspended Process:

A **suspended process** is a process that is temporarily removed from memory and placed in secondary storage (disk). It is not immediately available for execution.

Reasons for Process Suspension:

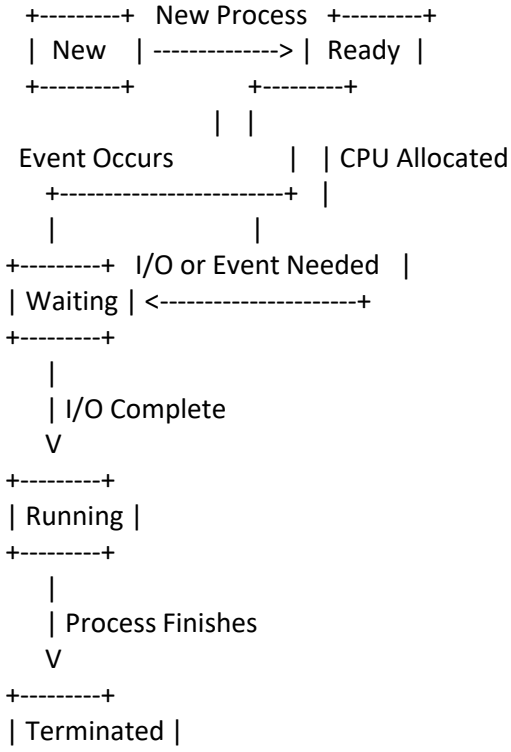
1. **Lack of Memory:** When RAM is full, inactive processes are moved to disk.
2. **User Request:** The user may pause or suspend a process manually.
3. **OS Management:** The OS may suspend background processes to free up resources.
4. **Waiting for I/O:** The process is waiting for input or an external event.

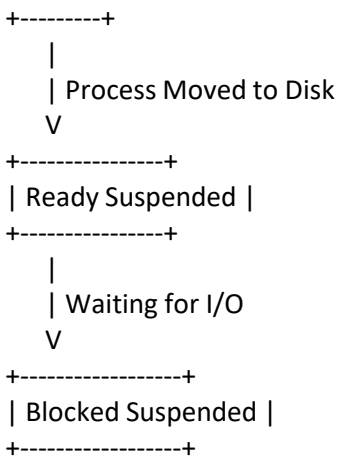
Seven-State Process Model (Including Suspended States):

The **seven-state model** extends the five-state model by adding:

1. **Ready Suspended:** A ready process moved to disk.
2. **Blocked Suspended:** A waiting process moved to disk.

Diagram:





Q9: Explain nine process states recognized by the UNIX SVR4 operating system with a suitable diagram. (4 Marks)

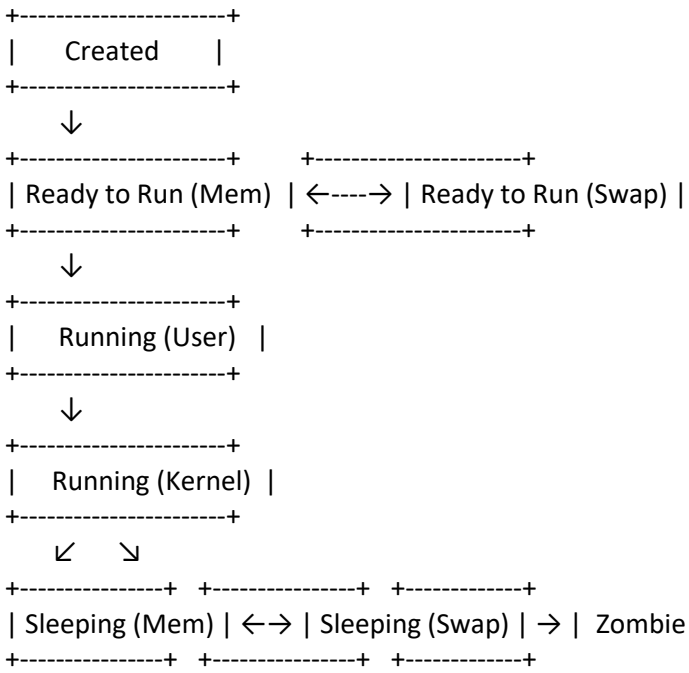
Nine Process States in UNIX SVR4:

The **UNIX SVR4** operating system extends the five-state model to **nine states** for better process management.

Process States:

1. **Created (New):** The process is created but not yet ready to execute.
2. **Ready to Run (in Memory):** The process is in RAM and waiting for CPU allocation.
3. **Ready to Run (Swapped):** The process is in secondary storage, waiting to be loaded into RAM.
4. **Running (User Mode):** The process is executing in user mode.
5. **Running (Kernel Mode):** The process is executing system calls in kernel mode.
6. **Sleeping (in Memory):** The process is waiting for an event and remains in RAM.
7. **Sleeping (Swapped):** The process is waiting for an event and is moved to secondary storage.
8. **Preempted:** A running process is forcibly stopped by the scheduler.
9. **Zombie:** The process has completed execution but still exists in the process table.

Diagram:



```
    ↘
 +----------------+
 |  Preempted     |
 +----------------+
```

Q10: Define multithreading. Differentiate between single-threaded and multi-threaded process models with a suitable diagram. (4 Marks)

Definition of Multithreading:

Multithreading allows a **single process** to have **multiple threads** running concurrently, improving efficiency and responsiveness.

Difference Between Single-Threaded and Multi-Threaded Models:

Feature	Single-Threaded Process	Multi-Threaded Process
Definition	Only one thread is executed per process.	Multiple threads run within a process.
Execution	If the thread is blocked, the entire process waits.	Other threads can continue execution if one thread is blocked.
Resource Usage	Requires more resources (separate stack, registers, etc.).	Threads share memory and resources, reducing overhead.
Performance	Slower because only one operation runs at a time.	Faster as multiple operations execute simultaneously.
Example	Old versions of MS Word.	Modern web browsers (Chrome, Firefox).

Diagram:

Single-Threaded Process:

```
+-----------+
| Process   |
| (Code, Data, Stack) |
| Thread 1  |
+-----------+
```

Multi-Threaded Process:

```
mathematica
CopyEdit
+------------------+
| Process          |
| (Code, Data)     |
| Thread 1         |
| Thread 2         |
| Thread 3         |
+------------------+
```

Q11: Explain Kernel-Level Threads (KLT) and User-Level Threads (ULT) with their advantages and disadvantages. (4 Marks)

1. User-Level Threads (ULT):

- Threads are managed by the user-space thread library, without kernel involvement.
- The OS treats all threads in a process as a single entity.

Advantages:

✓ Faster thread creation and management.

✓ No kernel-mode privilege required.

Disadvantages:

✗ If one thread is blocked, the entire process is blocked.

✗ Cannot utilize multiple processors effectively.

2. Kernel-Level Threads (KLT):

- Managed directly by the OS kernel, with each thread treated as an independent entity.
- OS schedules threads individually.

Advantages:

✓ If one thread is blocked, others continue execution.

✓ Can run on multi-core processors.

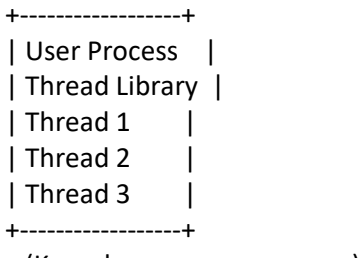
Disadvantages:

✗ Higher overhead due to kernel intervention.

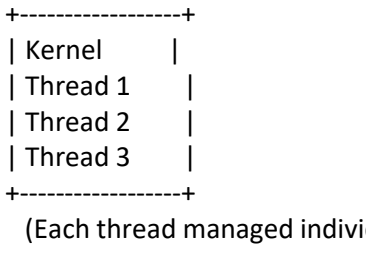
✗ Slower than ULTs.

Diagram:

User-Level Threads:



Kernel-Level Threads:



Q12: What is scheduling? Explain various short-term scheduling criteria. (4 Marks)

Definition of Scheduling:

Scheduling is the process of **selecting which process** will be assigned CPU time. **Short-term scheduling** determines which process gets CPU next.

Short-Term Scheduling Criteria:

1. **CPU Utilization:** The CPU should be kept as busy as possible.
2. **Throughput:** The number of processes completed per unit time.
3. **Turnaround Time:** The total time taken from process submission to completion.

4. **Waiting Time:** The time a process spends in the ready queue.
5. **Response Time:** The time taken for the system to start responding to user input.
6. **Fairness:** Ensuring no process is starved for CPU time.

Diagram:

Process Queue → Scheduler → CPU Execution → Process Completion

Q13: Explain the FCFS scheduling algorithm with an example. (4 Marks)

Definition of FCFS (First-Come, First-Served):

FCFS is a **non-preemptive scheduling algorithm** where the **first process that arrives gets executed first**.

Working of FCFS:

- Processes are **executed in the order they arrive**.
- Once a process starts execution, it runs **until completion** before moving to the next.

Example:

Process	Arrival Time	Burst Time
P1	0	4
P2	1	3
P3	2	2
P4	3	1

Gantt Chart:

| P1 | P2 | P3 | P4 |
0 4 7 9 10

Calculations:

- **Turnaround Time (TAT) = Completion Time - Arrival Time**
- **Waiting Time (WT) = Turnaround Time - Burst Time**

Process	Completion Time	Turnaround Time (TAT)	Waiting Time (WT)
P1	4	4 - 0 = 4	4 - 4 = 0
P2	7	7 - 1 = 6	6 - 3 = 3
P3	9	9 - 2 = 7	7 - 2 = 5
P4	10	10 - 3 = 7	7 - 1 = 6

Average Waiting Time (AWT) = (0+3+5+6) / 4 = 3.5 ms

Q14: Briefly define the Shortest Process Next (SPN) scheduling algorithm with an example. (4 Marks) SJF

Definition of SPN:

Shortest Process Next (SPN) is a **non-preemptive scheduling algorithm** where the process with the **shortest burst time** is executed first.

Working of SPN:

- The CPU selects the process with the **smallest execution time**.

- If two processes have the same burst time, the **one that arrived first is executed first**.
- **Non-preemptive**: Once a process starts execution, it runs **until completion**.

Example:

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	2

Gantt Chart:

```
| P1  | P3  | P4  | P2  |
0     7     8     10    14
```

Calculations:

Process	Completion Time	Turnaround Time (TAT)	Waiting Time (WT)
P1	7	7 - 0 = 7	7 - 7 = 0
P2	14	14 - 2 = 12	12 - 4 = 8
P3	8	8 - 4 = 4	4 - 1 = 3
P4	10	10 - 5 = 5	5 - 2 = 3

Average Waiting Time (AWT) = (0+8+3+3) / 4 = 3.5 ms

Q15: Briefly define the Round Robin (RR) scheduling algorithm with an example. (4 Marks)

Definition of Round Robin (RR):

Round Robin (RR) is a **preemptive scheduling algorithm** where each process gets a **fixed time quantum** for execution.

Working of RR:

- The CPU assigns **each process a time slice (quantum)**.
- If a process **does not finish within its quantum**, it is **moved to the end of the queue**.
- This ensures **fair CPU allocation**.

Example (Time Quantum = 3ms):

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

Gantt Chart:

```
| P1  | P2  | P3  | P1  | P3  |
0     3     6     9     11    14
```

Calculations:

Process	Completion Time	Turnaround Time (TAT)	Waiting Time (WT)
P1	11	11 - 0 = 11	11 - 5 = 6

P2	6	6 - 1 = 5	5 - 3 = 2
P3	14	14 - 2 = 12	12 - 8 = 4

Average Waiting Time (AWT) = (6+2+4) / 3 = 4 ms

Q16: Briefly define the Shortest Remaining Time (SRT) scheduling algorithm with an example. (4 Marks)

Definition of SRT:

Shortest Remaining Time (SRT) is a **preemptive version of SPN**, where the process with the **smallest remaining execution time** is always given the CPU.

Working of SRT:

- If a new process arrives with a **shorter remaining time**, the currently executing process is **preempted**.
- Ensures **shorter jobs complete faster** but may cause **starvation of longer processes**.

Example:

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	2

Gantt Chart:

```
| P1  | P2  | P3  | P2  | P1  |
0    1    3    5    7    11
```

Calculations:

Process	Completion Time	Turnaround Time (TAT)	Waiting Time (WT)
P1	11	11 - 0 = 11	11 - 8 = 3
P2	7	7 - 1 = 6	6 - 4 = 2
P3	5	5 - 2 = 3	3 - 2 = 1

Average Waiting Time (AWT) = (3+2+1) / 3 = 2 ms

Q17: Briefly define the Highest Response Ratio Next (HRRN) scheduling algorithm with an example. (4 Marks)

Definition of HRRN:

HRRN selects the process with the **highest response ratio**, calculated as:
Response Ratio=Waiting Time+Burst TimeBurst Time\text{Response Ratio} = \frac{\text{Waiting Time} + \text{Burst Time}}{\text{Burst Time}}Response Ratio=Burst TimeWaiting Time+Burst Time

Working of HRRN:

- Balances **waiting time** and **burst time** to prevent starvation.
- Processes with longer waiting times **get priority**.

Example:

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4

P3 2 2

Step 1: Calculate Response Ratio
At **time 2**, processes P1, P2, and P3 are available.

Process	Waiting Time	Response Ratio
P1	2	(2+8)/8 = 1.25
P2	1	(1+4)/4 = 1.25
P3	0	(0+2)/2 = 1.00

P1 and P2 have the highest ratio (1.25), so P1 runs first.
Gantt Chart:

```
| P1  | P2  | P3  |
0    8    12   14
```

Q18: What requirement is memory management intended to satisfy? (4 Marks)

Memory Management Requirements:

Memory management is responsible for **efficiently allocating and managing memory resources** in an OS.

1. **Process Isolation:** Ensures each process **does not interfere** with others.
2. **Automatic Allocation and Deallocation:** Allocates memory **dynamically** as processes start and end.
3. **Protection and Security:** Prevents **unauthorized access** to memory regions.
4. **Multiprogramming Support:** Enables multiple programs to run **simultaneously** by managing memory efficiently.
5. **Long-Term Storage:** Manages virtual memory and paging to **extend memory capacity**.

Q19: Give the difference between Fixed and Dynamic Partitioning. (4 Marks)

Definition:

Memory partitioning divides main memory into sections to manage processes efficiently. There are two types: **Fixed Partitioning** and **Dynamic Partitioning**.

Difference Between Fixed and Dynamic Partitioning:

Feature	Fixed Partitioning	Dynamic Partitioning
Definition	Memory is divided into fixed-size partitions at system boot.	Partitions are created dynamically based on process size.
Memory Utilization	May lead to **internal fragmentation** (unused space in partitions).	More efficient memory use, but **external fragmentation** may occur.
Flexibility	Not flexible; sizes are pre-defined.	More flexible; partitions adjust to process size.
Process Allocation	Processes can only fit into a partition of equal or larger size.	Processes can fit exactly into dynamically allocated partitions.
Example	IBM OS/360	Modern OS like Windows, Linux

Q20: What is Partitioning? Explain Memory Partitioning Techniques. (4 Marks)

Definition of Partitioning:

Partitioning is the process of **dividing main memory** into separate sections to manage multiple processes efficiently.

Types of Memory Partitioning Techniques:

1. **Fixed Partitioning:**
   - Divides memory into fixed-sized blocks.
   - Can cause **internal fragmentation** if processes are smaller than partition size.
2. **Dynamic Partitioning:**
   - Memory partitions are **created at runtime** to match process sizes.
   - Reduces internal fragmentation but may cause **external fragmentation**.
3. **Paging:**
   - Divides memory into **fixed-size pages** and maps them to **frames** in physical memory.
   - Eliminates external fragmentation.
4. **Segmentation:**
   - Divides memory into **variable-sized segments** based on logical program structure.
   - More flexible than paging.

Q21: What is Dynamic Partitioning? Explain First-Fit, Best-Fit, and Next-Fit placement policies. (4 Marks)

Definition of Dynamic Partitioning:

- In **Dynamic Partitioning**, memory partitions are created **as needed** for processes, reducing internal fragmentation.
- However, **external fragmentation** can occur as memory gets divided unevenly.

Placement Policies:

memorymenagement unit:-3 pg:-21

1. **First-Fit:**
   - The process is allocated to the **first available** memory block large enough to fit it.
   - **Fast but may cause fragmentation.**
2. **Best-Fit:**
   - Allocates the **smallest available block** that fits the process.
   - **Minimizes wasted space** but may leave many **small unusable holes**.
3. **Next-Fit:**
   - Similar to First-Fit, but **searches from the last allocated memory block** instead of starting from the beginning.
   - **Helps distribute memory usage evenly.**

Q22: Explain the two Fetching policies. Explain various replacement algorithms with examples (Optimal, LRU, FIFO, Clock). (4 Marks)

aakash unit1 pg:-24

Fetching Policies:

Fetching policies determine **when** and **how** a page is loaded into memory.

memorymenagment u-3 pg:-26

1. **Demand Paging:**
   - Pages are loaded **only when needed**, reducing memory usage.
   - Example: Virtual memory in Windows/Linux.
2. **Pre-Paging:**

- Loads **multiple pages in advance**, anticipating future memory needs.
- Example: Used in some high-performance computing systems.

Page Replacement Algorithms:

1. **Optimal (OPT) Replacement:**
   - Replaces the page that will **not be used for the longest time** in the future.
   - **Best performance but impractical** (requires future knowledge).
   - Example: Used for theoretical performance evaluation.
2. **Least Recently Used (LRU):**
   - Replaces the **least recently accessed** page.
   - Example: Used in cache memory management.

Aakash unit 1 page:-27

3. **First-In-First-Out (FIFO):**
   - Removes the **oldest loaded page** first.
   - Example: Simple but may lead to **Belady's anomaly** (performance worsens as memory size increases).
4. **Clock (Second-Chance):**
   - Uses a **circular queue** to track page usage.
   - Pages are **given a second chance** before being replaced.
   - Example: Used in UNIX systems.

Q23: Explain the Clock Policy - replacement algorithm in detail. (4 Marks)

Definition:

The **Clock (Second-Chance) Algorithm** is a **page replacement policy** that improves FIFO by giving pages a **"second chance"** before replacement.

Working of Clock Algorithm:

1. Pages are stored in a **circular queue** with a **use bit** (0 or 1).
2. The **OS checks the oldest page** in the queue:
   - If use bit = 1, it is **reset to 0** and the page is given a second chance.
   - If use bit = 0, the page is **replaced**.
3. The clock hand **moves forward** to check the next page.

Example:

Initial Queue:

[P1 (1) | P2 (0) | P3 (1) | P4 (0)] (Clock hand → P1)

- P1 has use bit = 1, so **reset it to 0** and move to P2.
- P2 has use bit = 0, so **replace P2 with a new page**.

Advantages of Clock Policy:

✓ **Efficient:** Avoids frequent replacements.

✓ **Better than FIFO:** Reduces **unnecessary page swaps**.

Q24: What is the Buddy System? Explain in detail. (4 Marks)

Definition:

The **Buddy System** is a **memory allocation technique** that **divides memory into power-of-2-sized blocks** to efficiently manage **dynamic memory allocation** and reduce fragmentation.

Working of the Buddy System:

1. **Memory is divided into large blocks**, and each block is **split into two "buddies"** when a process

requests memory.
2. If a block is **too large**, it is split further into smaller blocks **until the required size is reached**.
3. If a process **terminates**, its memory block **merges back with its buddy** if free, forming a larger block.

Example:

- Suppose we have **1024 KB** of memory and a process requests **200 KB**.
- The system **splits 1024 KB → 512 KB → 256 KB → 128 KB** (too small).
- The **256 KB block** is allocated to the process, leaving **another 256 KB free**.

Advantages:

✓ **Fast allocation and deallocation.**

✓ **Reduces external fragmentation** by merging buddy blocks.

Disadvantages:

✗ May cause **internal fragmentation** due to power-of-2 size constraints.

Q25: What is Virtual Memory? Explain different addressing techniques. (4 Marks)

Definition:

Virtual memory is a **technique that extends physical memory** by using disk space to store inactive parts of processes, allowing **larger programs to run** than available RAM.

Virtual Memory Addressing Techniques:

1. **Logical Addressing:**
   - The **CPU generates logical addresses**, which are mapped to **physical addresses** by the OS.
   - Example: A process may think it has **continuous memory**, but in reality, its data is stored in **scattered locations**.
2. **Paging:**
   - Divides virtual memory into **fixed-sized pages** and maps them to **physical frames** in RAM.
   - Example: A 1GB program may be **split into 4KB pages**, placed in **different RAM locations**.
3. **Segmentation:**
   - Divides memory into **variable-sized segments** based on **program structure** (code, stack, data).
   - Example: A **compiler stores code, variables, and stack separately**, each with a **different base address**.

Q26: What is Paging? Explain Address Translation for Paging. (4 Marks)

Definition of Paging:

Paging is a **memory management scheme** that eliminates **external fragmentation** by dividing both **logical and physical memory into fixed-sized blocks** called **pages** and **frames**.

Working of Paging:

1. The **logical address space** is divided into **pages** (e.g., 4 KB each).
2. The **physical memory (RAM)** is divided into **frames** of the same size.
3. The **OS maintains a Page Table**, mapping **logical pages to physical frames**.
4. When a process accesses a page, the OS **translates the logical address into a physical address**.

Address Translation for Paging:

Logical Address → Page Table → Physical Address

Logical Address Page Table Entry Frame Number Physical Address

Page 0	Frame 5	5	5 * 4KB
Page 1	Frame 2	2	2 * 4KB
Page 2	Frame 8	8	8 * 4KB

Example:
A **logical address (Page 1, Offset 1024)** is translated to a **physical address in Frame 2 (Base Address + 1024)**.

Advantages:

✓ Eliminates **external fragmentation**.
✓ Allows **non-contiguous memory allocation**.

Q27: What is a Translation Lookaside Buffer (TLB)? Explain its working with a clear diagram. (4 Marks)

Definition of TLB:

A **Translation Lookaside Buffer (TLB)** is a **high-speed cache** that stores **recently used page table entries** to speed up **logical-to-physical address translation**.

Working of TLB:

1. When a **CPU requests a memory address**, it first checks the **TLB**.
2. **If the entry is found (TLB Hit),** the address translation is **fast**.
3. **If the entry is not found (TLB Miss),** the OS fetches the page table entry from memory and updates the **TLB**.

Diagram:

Logical Address → TLB → (Hit) → Physical Address
|
→ (Miss) → Page Table → Physical Address

Example:

- Suppose **Page 2 → Frame 5** is stored in the TLB.
- If a process requests **Page 2**, the **TLB provides Frame 5 instantly**, avoiding a **slow memory access**.

Advantages:

✓ **Faster memory access** by avoiding page table lookups.
✓ **Improves CPU performance**.

Q28: What is Segmentation? Explain Address Translation for Segmentation. (4 Marks)

Definition of Segmentation:

Segmentation is a **memory management technique** that divides the process into **logical segments** (e.g., code, data, stack) instead of **fixed-sized pages**.

Address Translation for Segmentation:

1. The **logical address** consists of **Segment Number** and **Offset**.
2. The OS maintains a **Segment Table**, mapping **segments to physical memory locations**.
3. The CPU translates the **logical address into a physical address** using this table.

Example of Segment Table:

Segment Base Address Limit

Code	1000	400
Data	2000	300
Stack	5000	500

If a process accesses **Segment 1, Offset 150**, the physical address is:
Base Address + Offset=2000+150=2150

Advantages of Segmentation:
✓ **Reduces internal fragmentation** (unlike paging).
✓ **Supports dynamic memory allocation**.

Disadvantages:

✗ **Causes external fragmentation** (since segment sizes vary).

Q29: What is Protection and Sharing in Memory Management? (4 Marks)

Protection in Memory Management:

Protection ensures that **one process cannot access another process's memory**, preventing unauthorized access or accidental modification.

Protection Mechanisms:

1. **Base and Limit Registers:** Define the **starting and ending address** a process can access.
2. **Memory Access Control:** The OS restricts read, write, and execute permissions.
3. **Virtual Memory Protection:** Prevents a process from modifying memory not assigned to it.

Sharing in Memory Management:

Sharing allows **multiple processes** to access the **same memory space** for efficiency.

Sharing Mechanisms:

1. **Shared Pages in Paging:** Multiple processes can share the same page.
2. **Shared Code Segments in Segmentation:** Code and libraries can be shared to reduce memory usage.
3. **Interprocess Communication (IPC):** Shared memory is used for processes to exchange data.

Solve Section (Numerical Problems)

Q30: Calculate the Average Waiting Time and Average Turnaround Time for CPU scheduling algorithms (FCFS, HRRN, SPN, SRN, RR with time quantum = 1, RR with time quantum = 2). (4 Marks)

Given Processes:

Process	Arrival Time	Burst Time
P1	0	4
P2	5	5
P3	1	2
P4	4	1
P5	7	3

P6 12 3

Solution:

- **Apply FCFS, HRRN, SPN, SRN, RR (TQ=1), RR (TQ=2).**
- Compute **Completion Time (CT)**, **Turnaround Time (TAT)**, and **Waiting Time (WT)**.
- Find **Average WT and Average TAT** for each scheduling algorithm.

(Detailed step-by-step calculations are needed here.)

Q31: Consider the page reference string and find the number of page faults using Optimal, FIFO, LRU, and Clock Page Replacement Algorithms. (4 Marks)

Given Page Reference String:

7,0,1,2,0,3,0,4,2,3,0,3,2,37, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 37,0,1,2,0,3,0,4,2,3,0,3,2,3

Frames: 4

Solution:

1. **Apply Optimal Page Replacement:**
   - Replace the page that **won't be used for the longest time**.
2. **Apply FIFO Page Replacement:**
   - Replace the **oldest page in memory** first.
3. **Apply LRU Page Replacement:**
   - Replace the **least recently used page**.
4. **Apply Clock Page Replacement:**
   - Use **use-bit tracking** to replace pages efficiently.

(Detailed step-by-step calculations are needed here.)

Q32: Fit a 15K process using First-Fit, Best-Fit, and Next-Fit Algorithms. (4 Marks)

Given Memory Blocks:

yaml
CopyEdit
P10: 10K | P21: 18K | P13: 16K | P3: 50K | P23: 12K | P4: 30K

Process to fit: 15K

Solution:

1. **First-Fit:** Allocate to the **first available block** (P21: 18K).
2. **Best-Fit:** Allocate to the **smallest available block** that fits (P13: 16K).
3. **Next-Fit:** Start from the last allocated block (P13: 16K).

(Step-by-step placement and fragmentation calculations needed.)