OS IAE-2 QB

Q1) Explain demand paging.

Demand paging is a memory management technique used in modern operating systems where pages of data (from a process) are only loaded into the main memory (RAM) when they are required during program execution. It helps to optimize memory usage by avoiding the loading of unnecessary pages.

How Demand Paging Works

1. **Page Table**:
    - Each process has a page table that keeps track of the mapping between the logical (virtual) addresses and physical addresses.
    - The page table entries include a *Present/Absent* bit, indicating whether the page is in physical memory.

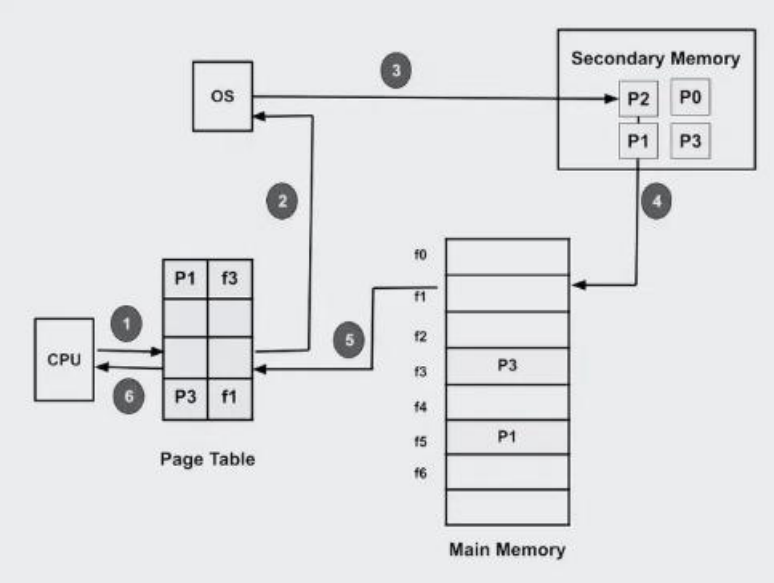
2. **Page Fault**:
    - When a process tries to access a page that is not currently in memory (the Present bit is "absent"), the system raises an interrupt called a **Page Fault**.
    - The operating system then loads the required page from secondary storage (e.g., a hard disk or SSD) into physical memory.

3. **Execution Resumes**:
    - Once the page is loaded into memory, the process resumes execution at the point where it was interrupted.

Advantages of Demand Paging

1. **Efficient Memory Usage**:
    - Only the necessary pages are loaded into memory, which reduces memory wastage.

2. **Supports Large Programs**:
    - Programs larger than the available physical memory can execute because pages are loaded on demand.



Example:
Assume a program has 10 pages, but only pages 1, 3, and 5 are needed at the beginning. With demand paging, only these pages are loaded, saving memory and improving performance.

Q2) Explain in brief: RAID

RAID (Redundant Array of Independent Disks)

Definition: RAID is a **data storage virtualization technology** that combines multiple physical hard drives into a single logical unit to improve **performance**, **fault tolerance**, or both.

Objectives of RAID:

1. **Increased Performance** – by parallel data access.
2. **Redundancy/Fault Tolerance** – by storing data in a way that allows recovery if a disk fails.
3. **High Availability** – systems can continue operating even if one disk fails (depending on RAID level).

Common RAID Levels:

RAID Level	Features	Advantages	Disadvantages
RAID 0	Striping (no redundancy)	High speed	No fault tolerance
RAID 1	Mirroring	High reliability	Doubles storage cost
RAID 5	Striping with parity	Balance of speed and redundancy	Complex parity calculations
RAID 6	Like RAID 5 but with double parity	Can handle 2 disk failures	Slower write performance
RAID 10 (1+0)	Mirroring + Striping	High speed + redundancy	Expensive (needs more disks)

Advantages

- Improves read/write performance.
- Combines multiple drives for large storage capacity.

Q3) Differentiate between paging and segmentation.

Aspect	Paging	Segmentation
Concept	Divides memory into fixed-sized blocks called *pages*.	Divides memory into variable-sized blocks called *segments*.
Size	Pages are of *fixed size*.	Segments are of *variable size*.
Addressing	Uses a **page number** and **offset**.	Uses a **segment number** and **offset**.
Fragmentation	Causes **internal fragmentation**.	Causes **external fragmentation**.
View of Memory	Provides a *physical view* of memory.	Provides a *logical view* of memory.
Mapping Mechanism	Requires a **page table** for mapping pages to frames.	Requires a **segment table** to store base address and size of each segment.
Application	Used in virtual memory systems for efficient memory usage.	Useful in organizing memory based on logical divisions like code, data, stack.
Fragmentation Type	Leads to small unused spaces within allocated pages.	Leads to scattered free memory spaces.

Q4) What is mutual exclusion? Explain hardware approaches for it.

Mutual exclusion is a fundamental concept in operating systems used to **prevent multiple processes from accessing a shared resource (like memory, file, printer, etc.) at the same time**. It ensures that only **one process** can enter its **critical section** at a time, avoiding data inconsistency and race conditions.

Critical Section:
A **critical section** is a part of the program where the process accesses shared resources. Mutual exclusion ensures only one process executes in the critical section at any time.

Hardware Approaches for Mutual Exclusion

Several hardware-based solutions exist to implement mutual exclusion. These mechanisms rely on special instructions or hardware features to manage resource sharing effectively. Below are common hardware approaches:

1. Disabling Interrupts

- **Description**: A process disables all interrupts when entering its critical section. This ensures that no other process can preempt the CPU, giving the current process exclusive access.
- **Advantages**:
  - Simple to implement.
  - Works effectively in single-processor systems.
- **Disadvantages**:
  - Not suitable for multi-processor systems.
  - Can lead to priority inversion or system-wide inefficiency if the process forgets to re-enable interrupts.

2. Test-and-Set Instruction

- **Description**: This is a hardware-supported atomic instruction. It works on a binary variable (lock) and performs two actions:
  1. Tests if the lock is free (0).
  2. Sets the lock to busy (1) in one atomic step.

Algorithm:

Pseudo Copy

```
while (TestAndSet(lock) == 1)
    ; // Busy-wait (loop until lock is free)
// Critical Section
lock = 0; // Release lock
```

- **Advantages**:
  - Atomic operation eliminates race conditions.
  - Works for multi-processor systems.
- **Disadvantages**:
  - May cause **busy-waiting**, which wastes CPU cycles.

3. Compare-and-Swap (CAS) Instruction

- **Description**: Another atomic instruction that compares the value of a variable with an expected value and, if they match, swaps it with a new value.

Algorithm:

Pseudo Copy

```
while (CompareAndSwap(lock, 0, 1) != 0)
    ; // Busy-wait
// Critical Section
lock = 0; // Release lock
```

- **Advantages**:
  - Efficient and widely supported by modern processors.
  - No need for software-level locks.
- **Disadvantages**:
  - Still involves busy-waiting.
  - Complex to implement in some systems.

4. Peterson's Algorithm (Software + Hardware Hybrid)

- **Description**: Though more of a software solution, it works with hardware support for atomic variable updates. It ensures mutual exclusion between two processes using a shared flag variable and a turn variable.
- **Advantages**:
  - Eliminates race conditions without busy-waiting in specific cases.
- **Disadvantages**:
  - Limited to two processes.
  - Not efficient for multi-processor systems.

Q6) What is Semaphore? Explain different types of Semaphore

A **semaphore** is a **synchronization tool** used in operating systems to **manage access to shared resources** by multiple processes in a **concurrent system**, such as a multitasking OS.

Semaphores help in solving problems like:

- **Mutual exclusion**
- **Deadlocks**
- **Process synchronization**

They are **integer variables** that are accessed through two atomic operations:

- **wait()** (also known as **P()** or **down()**)
- **signal()** (also known as **V()** or **up()**)

Operations:

1. **wait(S)**:
   - If S > 0, decrement S and allow the process to continue.
   - If S <= 0, the process is **blocked** (waits in a queue).
2. **signal(S)**:
   - Increments S.
   - If there are blocked processes, one is **unblocked** and allowed to proceed.

Types of Semaphores:

1. Counting Semaphore

- Can have **any non-negative integer value**.
- Used to control access to a resource that has **multiple instances** (e.g., a pool of printers or database connections).

Example: If 3 printers are available:

```c
semaphore = 3;

wait(semaphore);   // process acquires a printer
// use printer
signal(semaphore); // release the printer
```

2. Binary Semaphore (Mutex Semaphore)

- Only takes the values **0 or 1**.
- Used to implement **mutual exclusion**, i.e., ensuring only one process enters the critical section.

Example:

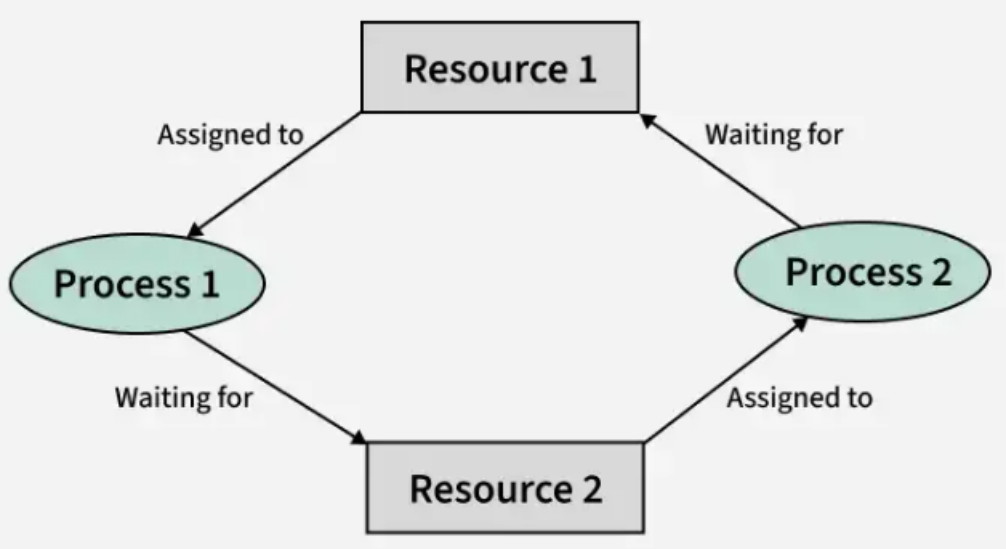
```c
semaphore = 1;

wait(semaphore);   // lock
// critical section
signal(semaphore); // unlock
```

3. Mutex (Mutual Exclusion Object)

- Technically a **special case of binary semaphore**.
- Only the process that **locked (waited)** it can **unlock (signal)** it.
- Often used in **thread synchronization** in multi-threaded programs.

Q7) What is DL & State necessary conditions for DL.

Deadlock is a state in a multi-processing system where a set of processes are unable to proceed because they are waiting for resources that are held by other processes in the same set. In simpler terms, it occurs when two or more processes are stuck waiting for each other to release resources, creating a cycle of dependency that prevents any of them from progressing.



Necessary Conditions for Deadlock
For a deadlock to occur, the following **four necessary conditions** (known as **Coffman Conditions**) must hold simultaneously:

1. **Mutual Exclusion**:
    - At least one resource must be held in a non-shareable mode (i.e., only one process can use the resource at a time).
2. **Hold and Wait**:
    - A process holding at least one resource is waiting to acquire additional resources that are currently held by other processes.
3. **No Preemption**:
    - Resources cannot be forcibly taken away from a process. A process must voluntarily release its resources.
4. **Circular Wait**:
    - There exists a circular chain of processes, where each process in the chain is waiting for a resource held by the next process in the chain.

Example
Imagine two processes, P1 and P2, and two resources, R1 and R2:

- **P1** holds **R1** and waits for **R2** (held by **P2**).
- **P2** holds **R2** and waits for **R1** (held by **P1**). This creates a circular wait, leading to deadlock.

Q8) Explain various ways to prevent DL

Deadlock prevention aims to ensure that at least one of the four necessary conditions for deadlock (mutual exclusion, hold and wait, no preemption, and circular wait) is eliminated. Below are the various strategies:

1. Eliminating Mutual Exclusion

- **Approach**: Make resources shareable whenever possible. If resources can be accessed by multiple processes simultaneously (e.g., read-only files), mutual exclusion is avoided.
- **Limitation**:
    - Not all resources can be shared (e.g., printers), so this method is not universally applicable.

2. Avoiding Hold and Wait

- **Approach**:
    - Ensure that processes request all required resources at the start and hold them until completion. Processes cannot request resources while holding others.
- **Implementation**:
    - Use a protocol where processes release resources before requesting additional ones.
- **Limitation**:
    - Leads to reduced resource utilization and potential process starvation.

3. Allowing Preemption

- **Approach**:
    - Permit the operating system to forcibly take resources away from processes when required by another process.

- **Implementation**:
  - If a process is holding a resource and waiting for another, the resource can be preempted and allocated to other waiting processes.
- **Limitation**:
  - Not all resources can be preempted (e.g., CPU registers).

4. Preventing Circular Wait

- **Approach**:
  - Impose a total ordering on all resources and require processes to request them in ascending order of their ranks.
- **Implementation**:
  - Assign a unique number to each resource. Processes must request resources in increasing order of numbers (e.g., request R1, then R2).
- **Limitation**:
  - This requires careful planning and resource ranking, which may not be feasible in all scenarios.

Q9) Explain DL detection & avoidance techniques.

Deadlock Detection
Deadlock detection focuses on identifying when a deadlock has occurred in the system. It requires monitoring processes and resources.

Techniques for Detection:

1. **Resource Allocation Graph**:
   - A graph is created where:
     - Processes are represented as nodes.
     - Resources are represented as nodes.
     - Edges show resource allocation or requests.
   - If a cycle is detected in the graph:
     - For **single instance resources**: The system is in deadlock.
     - For **multiple instance resources**: Further analysis is needed to confirm deadlock.

2. **Detection Algorithm**:
    - In systems with multiple instances of resources, algorithms (like the Banker's algorithm) are used to check if there are unresolvable requests, indicating deadlock.

Deadlock Avoidance
Deadlock avoidance ensures that the system does not enter a deadlock state. It requires information about resource usage in advance.

Techniques for Avoidance:

1. **Banker's Algorithm**:
    - Named after a bank managing loans, this algorithm dynamically checks whether granting a resource request leads to a safe state.
    - Steps:
        - Evaluate the current resource allocation.
        - Simulate allocation to the requesting process.
        - Check if the system remains in a safe state.
        - If not safe, deny the request.
2. **Resource Reservation**:
    - Processes must declare their maximum resource needs in advance.
    - The system only grants resources if it can guarantee safe execution.

Q10 Draw and explain disk performance parameters.

When accessing data on a **hard disk**, several factors affect how quickly the data can be read or written. These are known as **disk performance parameters**.

1. Seek Time (Ts)

- **Definition:** Time taken by the disk arm to move the read/write head to the track where the data is located.
- **Impact:** Major contributor to delay.
- **Average Seek Time:** Average over many random accesses (usually 2–10 ms for HDDs).

✅ 2. Rotational Latency (Tr)

- **Definition:** Time taken for the desired sector to rotate under the read/write head after the arm is positioned.
- **Average Rotational Latency =**
  $\frac{1}{2} \times \text{Rotation Time}$

> **For example, for a 7200 RPM disk:**
> **Rotation time =** $\frac{60}{7200} = 8.33$ **ms**
> **Avg. rotational latency =** $\frac{8.33}{2} \approx 4.17$ **ms**

3. Transfer Time (Tt)

- **Definition:** Time to actually read or write the data once the head is at the correct position.
- Depends on:
  - Sector size
  - Disk rotation speed
  - Data transfer rate

4. Access Time (Ta)

- **Definition:** Total time to access data from the disk.

$$\boxed{\text{Access Time (Ta)} = \text{Seek Time (Ts)} + \text{Rotational Latency (Tr)} + \text{Transfer Time (Tt)}}$$

5. Disk Throughput

- **Definition:** Amount of data that can be read/written per second.
- Measured in MB/s or GB/s.
- Affected by all the above factors.

Q11) Explain difference between external fragmentation and internal fragmentation.

Aspect	External Fragmentation	Internal Fragmentation
Definition	Occurs when free memory is scattered in small, non-contiguous blocks, preventing allocation of large processes even though there is enough total free memory.	Occurs when allocated memory is larger than the requested memory, leaving unused space within a block.
Cause	Happens due to dynamic memory allocation and the inability to fit processes into available contiguous spaces.	Happens due to the use of fixed-sized memory blocks (or pages) that may not match the process size exactly.
Type of Memory Wastage	Wastage occurs outside the allocated memory blocks.	Wastage occurs inside the allocated memory blocks.
Solution	Compaction (rearranging processes to combine free spaces into a single contiguous block).	Better memory partitioning or allocation techniques like variable-sized partitions.
Example	Having 50 KB free in two non-contiguous blocks (30 KB and 20 KB) but being unable to allocate a 40 KB process.	Allocating 100 KB for a 70 KB process, leaving 30 KB unused within the block.

Q12) How to solve fragmentation problem using paging?

Paging is an effective technique to resolve fragmentation problems, especially **external fragmentation**, by dividing memory into fixed-sized blocks. Here's how it works:

Solving Fragmentation Using Paging

1. **Fixed-Sized Pages**:
   - Logical memory (program address space) is divided into fixed-sized blocks called **pages**.
   - Physical memory (RAM) is divided into fixed-sized blocks called **frames**.
   - Since pages and frames are of the same size, processes can occupy any available frame, eliminating the need for contiguous allocation and **external fragmentation**.

2. **No External Fragmentation**:
    - Paging eliminates **external fragmentation** because processes no longer need contiguous memory.
    - Any free frame can be allocated to a page, irrespective of where it is located in physical memory.

3. **Internal Fragmentation**:
    - Paging may introduce **internal fragmentation**, as the allocated memory frame might not be fully utilized (e.g., if a process requires 6.5 KB and frames are 8 KB, 1.5 KB goes unused).
    - However, this is generally less significant than external fragmentation and can be minimized by using smaller page sizes.

4. **Page Table**:
    - A **page table** is maintained to map logical page numbers to physical frame numbers, ensuring efficient access and management.

Q13) Explain critical section problem. Explain the hardware solution to achieve the same.

The **critical section problem** arises in concurrent programming when multiple processes or threads access and manipulate shared resources (like memory, files, or variables). To prevent inconsistencies and race conditions, it's crucial to ensure that only one process accesses the critical section at a time. This is where the **critical section problem** focuses: designing a solution that ensures correct process synchronization.

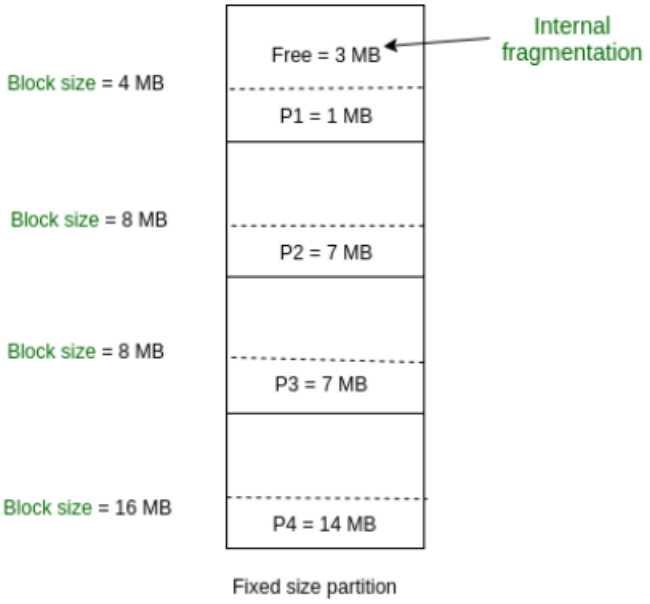
The hardware solutions are in Q4

Q14) Explain memory allocation strategies with suitable examples.

Memory allocation is the process of assigning portions of the system's memory to various programs and processes. The goal is to maximize efficiency, minimize fragmentation, and optimize resource utilization. Below are the common memory allocation strategies with examples:

1. Fixed Partitioning

- **Description**:
  - The physical memory is divided into fixed-size partitions during system initialization. Each process is allocated one partition.
- **Example**:
  - Suppose the memory is divided into partitions of 50 KB each:
    - Process 1 (30 KB) occupies Partition 1, leaving 20 KB unused (**internal fragmentation**).
    - Process 2 (45 KB) occupies Partition 2, leaving 5 KB unused.
- **Advantages**:
  - Simple to implement.
  - No external fragmentation.
- **Disadvantages**:
  - Internal fragmentation due to fixed sizes.

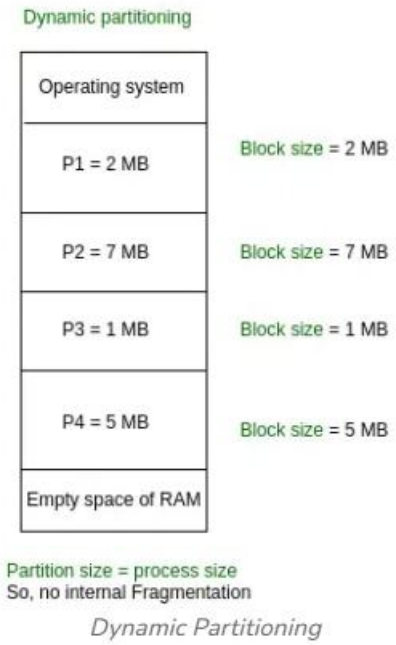


Fixed size partition

2. Variable Partitioning

- **Description**:
  - Partitions are created dynamically to fit the size of the processes. This minimizes internal fragmentation but can lead to **external fragmentation**.

- **Example**:
  - Process A (80 KB) is allocated 80 KB, followed by Process B (100 KB). If Process A finishes, the 80 KB block might remain unusable if no other process fits exactly.
- **Advantages**:
  - Efficient use of memory for varied process sizes.
- **Disadvantages**:
  - External fragmentation.

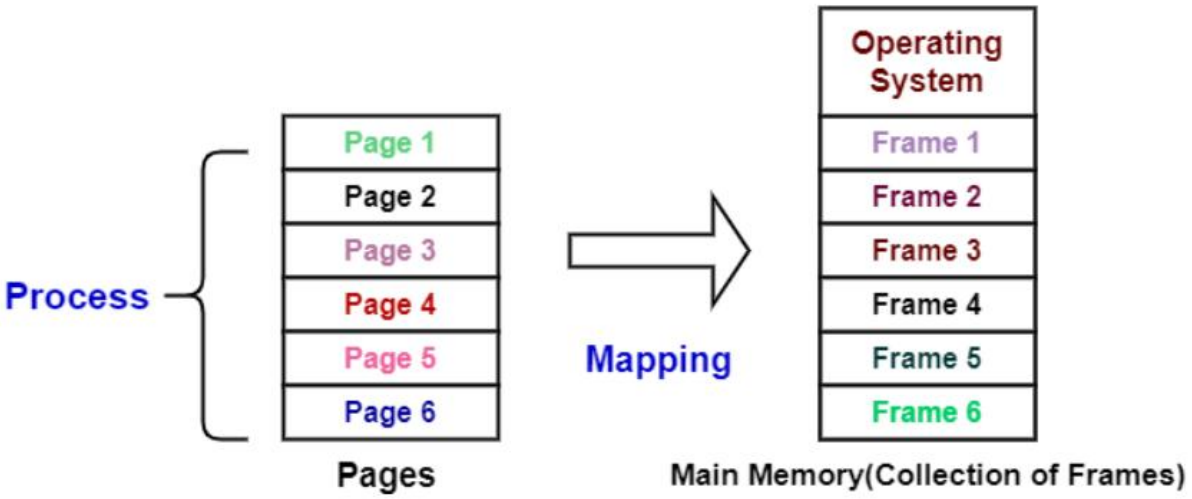


Dynamic Partitioning

3. Paging

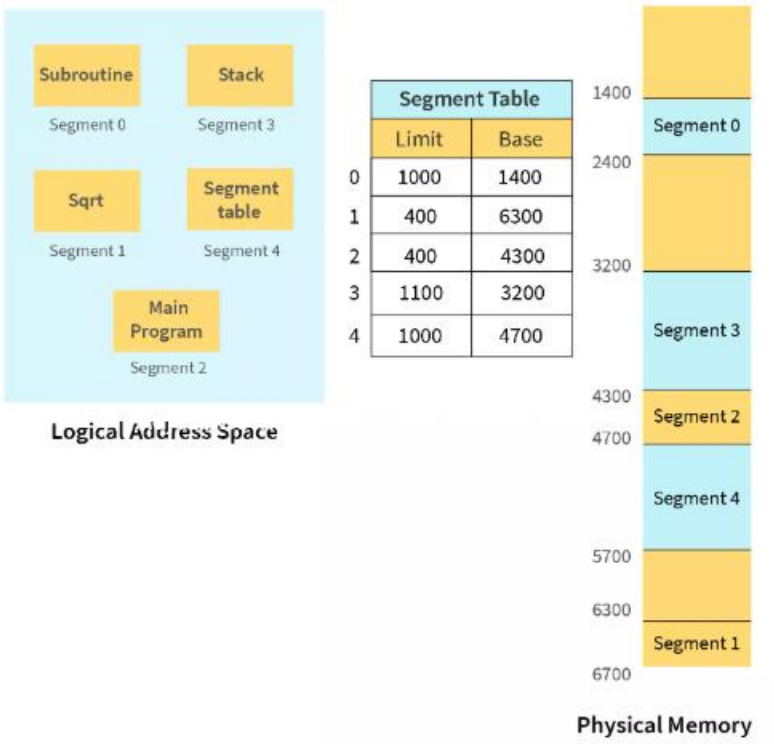
- **Description**:
  - Memory is divided into fixed-sized frames, and processes are divided into pages of the same size. Pages are mapped to frames using a page table.
- **Example**:
  - Suppose the frame size is 4 KB:
    - Process C (10 KB) is divided into 3 pages (4 KB, 4 KB, 2 KB).
    - These pages are allocated to any available frames, even if they are non-contiguous.

- **Advantages**:
  - Eliminates external fragmentation.
  - Flexible allocation.
- **Disadvantages**:
  - Can cause internal fragmentation within frames.



4. Segmentation

- **Description**:
  - Memory is divided into variable-sized segments based on logical divisions of a process, such as code, data, and stack. Each segment is allocated memory independently.
- **Example**:
  - Process D has three segments:
    - Code (30 KB), allocated to one memory block.
    - Data (50 KB), allocated to another block.
    - Stack (20 KB), allocated separately.
- **Advantages**:
  - Matches logical structure of programs.
  - No internal fragmentation.
- **Disadvantages**:
  - External fragmentation due to variable segment sizes.



Q15) Write short notes on principles of concurrency, segmentation and paging

📝 **1. Principles of Concurrency**

✅ **Definition: Concurrency** is the concept where **multiple processes execute simultaneously** or **overlap in time**, potentially interacting with shared resources.

✅ **Key Points:**

- **Processes may run in parallel** (on multicore) or be interleaved (on a single core).
- Shared resources can cause issues like **race conditions**, **deadlocks**, or **inconsistency**.

✅ **Important Concepts:**

- **Mutual Exclusion:** Ensure only one process accesses critical section at a time.
- **Synchronization:** Mechanisms like semaphores or monitors to coordinate processes.

- **Deadlock and Starvation:** Must be handled or prevented in concurrent systems.

📦 **2. Segmentation**

✅ **Definition: Segmentation** is a memory management technique where the **logical address space is divided into variable-sized segments** based on logical divisions in the program (e.g., code, stack, data).

✅ **Key Points:**

- Each segment has a **segment number** and **offset**.
- The OS maintains a **segment table** with base and limit for each segment.

✅ **Example:**

A program may be divided into:

- Segment 0 → Code
- Segment 1 → Data
- Segment 2 → Stack

✅ **Advantages:**

- No internal fragmentation.
- Segments map to program's logical structure.

📄 **3. Paging**

✅ **Definition: Paging** is a memory management scheme that eliminates external fragmentation by dividing **physical and logical memory into fixed-size blocks**:

- Logical memory → **Pages**
- Physical memory → **Frames**

✅ **Key Points:**

- Every logical page maps to a physical frame via a **page table**.
- **No need for contiguous allocation**, improves memory use.

✅ **Advantages:**

- Eliminates external fragmentation.
- Simplifies memory allocation.

✅ **Disadvantage:**

- May cause **internal fragmentation** (in last page).
- Requires overhead for maintaining **page tables**.

Q16) Explain paging in detail. Describe how logical address is converted into physical address

Logical to Physical Address Translation
When a program accesses memory, it generates a **logical address**. This address must be translated into a **physical address** to locate data in RAM.

1. Structure of Logical Address
A logical address consists of two parts:

- **Page Number (p)**: Identifies the specific page in the process's logical memory.
- **Page Offset (d)**: Specifies the exact location within that page.

2. Translation Steps

1. **Extract the Page Number**:
    - Divide the logical address by the page size (using integer division) to get the page number **p**.
2. **Locate the Frame Number**:
    - Use the **page table** to map the page number **p** to the corresponding frame number **f** in physical memory.
3. **Compute the Physical Address**:
    - Combine the frame number **f** (from the page table) with the **offset d** (from the logical address) to calculate the physical address.
    - **Physical Address** = (Frame Number × Frame Size) + Offset.

Example:

- Page Size = 1 KB (1024 bytes).
- Logical Address = 2049.
- Page Table:
    - Page 0 → Frame 5.
    - Page 1 → Frame 7.

1. **Page Number (p)** = ⌊2049 ÷ 1024⌋ = 2.
2. **Offset (d)** = 2049 mod 1024 = 1.
3. **Frame Number (f)** = Frame mapped to Page 2 (from the page table) = **Frame 7**.
4. **Physical Address** = (7 × 1024) + 1 = **7169**.

Thus, the logical address 2049 is translated into the physical address 7169.

Q17) How the classic synchronization problem -Dining philosopher is solved using semaphores?

Numerical Question

Q19) Explain about IPC.

Inter-Process Communication (IPC) refers to a set of mechanisms that allow processes to communicate and share data with each other. In modern operating systems, processes often need to coordinate their activities or exchange information, especially in multi-tasking environments.

Key Features of IPC

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

Common IPC Mechanisms

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

2. **Named Pipes**:
    - Similar to regular pipes but allow bidirectional communication and are identifiable by a name in the file system.
    - Useful for unrelated processes.

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

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

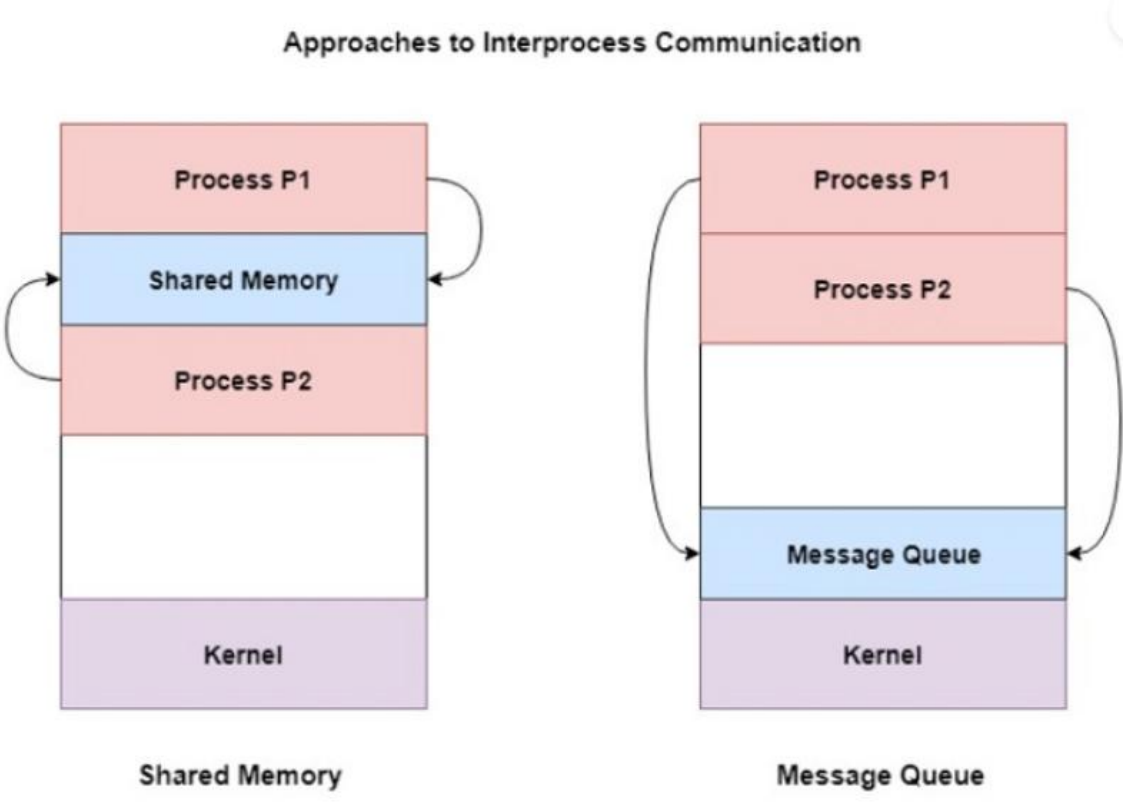
5. **Sockets**:
    - Facilitates communication between processes, even if they're on different machines in a network.
    - Commonly used in client-server models (e.g., web browsers and servers).

Advantages of IPC

- Enables efficient collaboration between processes.
- Improves resource sharing and utilization.

Examples of IPC in Action

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



Q20) What is the content of page table? Explain

A **Page Table** is a data structure used by the **Operating System** to **map logical (virtual) addresses to physical addresses** in **paging-based memory management**. Each **process has its own page table**, maintained by the OS.

📦 Contents of a Page Table Entry (PTE)

Each entry in the page table corresponds to one **logical page** and contains information about where that page resides in physical memory, along with protection and control bits.

🔹 **1. Frame Number**

- Indicates the **physical frame number** where the page is loaded.
- Used to calculate the actual **physical address**.

🔹 **2. Present/Absent Bit**

- **1** → Page is in physical memory.
- **0** → Page is not in memory (i.e., it's in secondary storage → page fault occurs).

🔹 **3. Protection Bits**

- Control **access rights**:
  - Read/Write
  - Read-Only
  - Execute permissions

🔹 **4. Dirty Bit (Modify Bit)**

- **1** → Page has been modified (written to) since it was loaded.
- Helps OS decide whether to write the page back to disk.

🔹 **5. Accessed Bit**

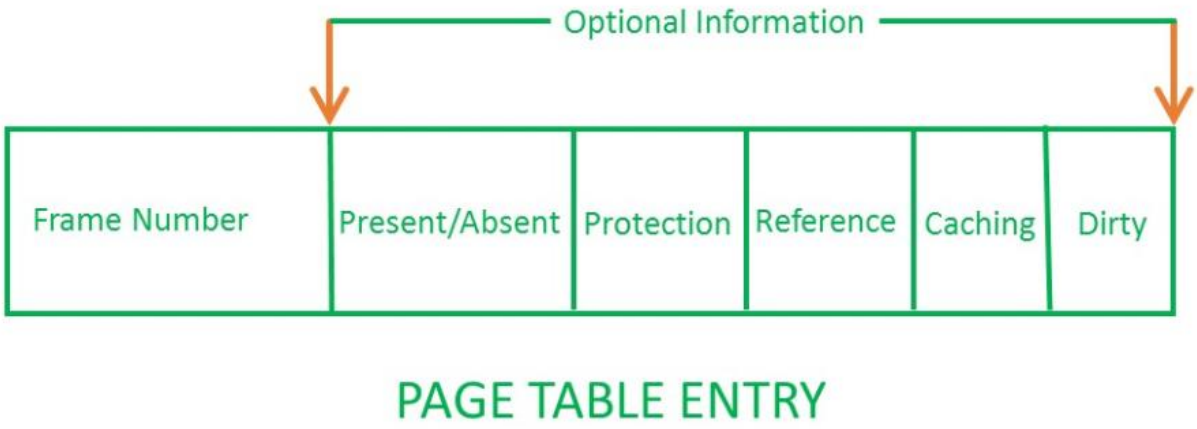
- Indicates whether the page has been accessed (read or written).
- Used for **page replacement algorithms** like LRU.

Explanation of How a Page Table Works

1. When a process accesses memory, it generates a **logical address** containing:
   - **Page Number** (used to index the page table).
   - **Offset** (used within the physical frame).
2. The operating system looks up the **page number** in the page table to find:
   - Whether the page is in memory (valid/invalid bit).

- The frame number corresponding to the page.

3. The physical address is calculated as:
   - **Physical Address** = (Frame Number × Frame Size) + Offset.



PAGE TABLE ENTRY

Q21) Explain with suitable example, how virtual address is converted to physical address?

In a system with virtual memory, every process works with **virtual addresses** (logical addresses) generated by the CPU. These virtual addresses are mapped to **physical addresses** (locations in RAM) using a page table maintained by the operating system.

Steps for Address Translation

1. **Structure of the Virtual Address**:
   - A virtual address consists of:
     - **Page Number (p)**: The high-order bits that identify the page in virtual memory.
     - **Offset (d)**: The low-order bits that identify the specific location within the page.

2. **Role of Page Table**:
   - The page table maps virtual page numbers to physical frame numbers.
   - Each entry in the page table contains:
     - A **frame number** corresponding to a page.
     - Additional information like validity, protection bits, etc.

3. **Translation Process**:
    - The CPU extracts the **page number (p)** from the virtual address and looks it up in the page table.
    - The **frame number (f)** is retrieved from the page table.
    - The physical address is computed by combining the frame number with the **offset (d)** from the virtual address.

Example: Example: dekhlo vo apna apna

Q22) What is virtual memory technique?

Virtual Memory is a **memory management technique** that provides an **illusion of a large, continuous main memory** to the processes, even if the actual physical memory (RAM) is limited. It allows processes to execute even when they are **not completely loaded into physical memory**.

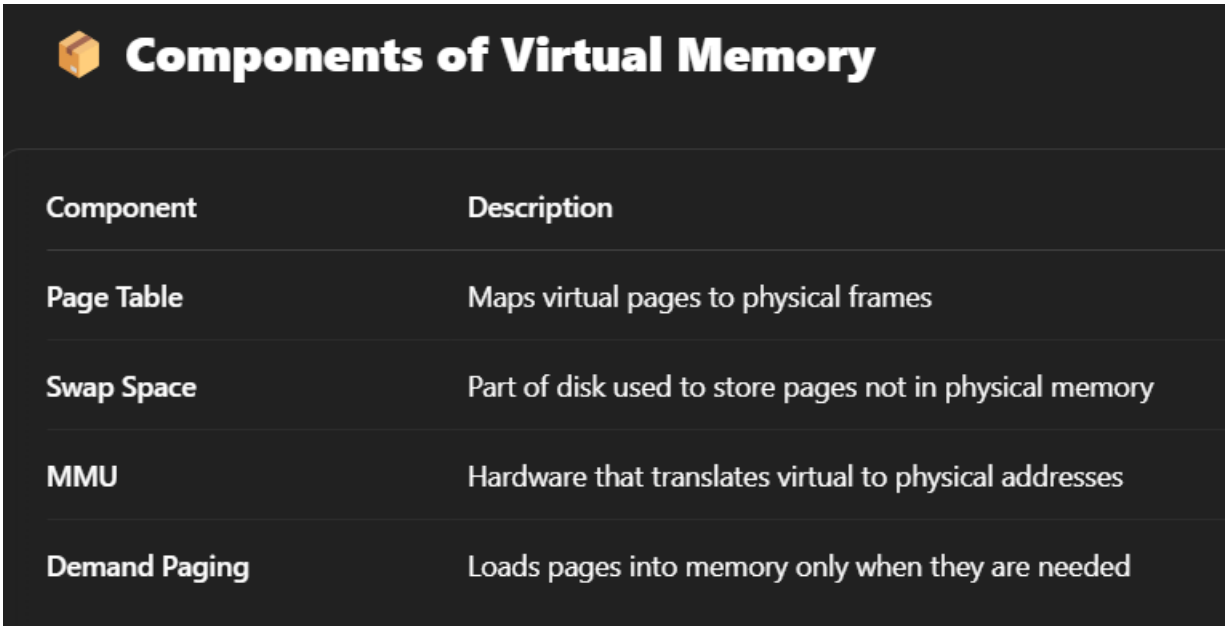
🎯 Key Features of Virtual Memory

- Allows **execution of large programs** that don't fit entirely in RAM.
- Provides **process isolation** and **security**.
- Enables **efficient multitasking** by giving each process its own virtual address space.

⚙️ How It Works

- The process's **logical address space** is stored in **secondary memory** (like a hard disk or SSD).
- Only **required parts (pages)** are loaded into **RAM** using **demand paging**.
- The **Memory Management Unit (MMU)** handles the translation from **virtual address to physical address** using a **page table**.

Example:
If a process needs 8 GB of memory but the system has only 4 GB of RAM, virtual memory loads only the required parts of the process into RAM and swaps other parts in and out as needed.



✅ **Advantages of Virtual Memory**

- **Efficient memory utilization**
- **Larger address space** than physical RAM
- Enables **multitasking**

❌ **Disadvantages**

- Slower than direct physical memory access (due to disk I/O)
- Requires more complex memory management

Q23) List page replacement algorithms? Explain anyone page replacement algorithms with example.

List of Page Replacement Algorithms

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

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

Explanation of FIFO Page Replacement Algorithm

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

Working Steps:

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

Example: dekhlo vo apna apna