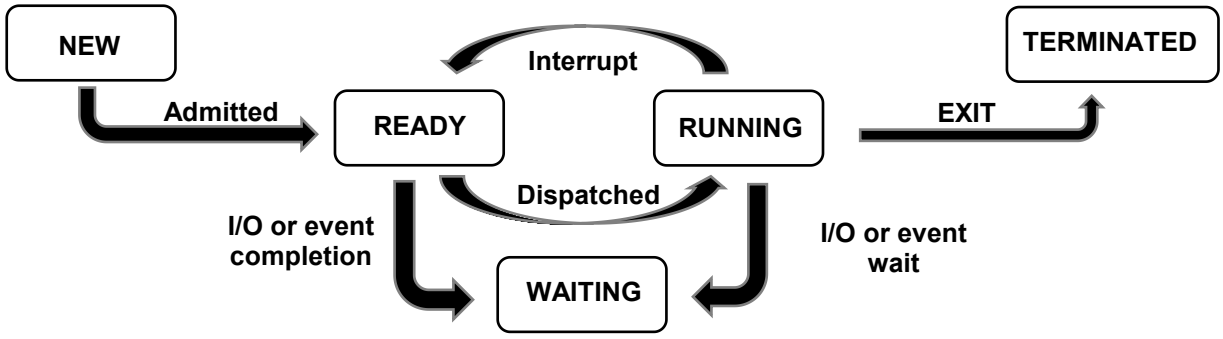
COMPREHENSIVE REPORT

PART A :

- **Create a comparison table: Processes vs Threads.**

Feature	Process	Thread
Definition	An independent program in execution with its own memory space.	A lightweight unit of execution within a process.
Memory	Has its own memory and resources (stack, data, code).	Shares memory and resources with other threads in the same process.
Communication	Requires Inter-Process Communication (IPC) mechanisms.	Easier communication since threads share the same memory.
Creation Overhead	High — creating a new process is expensive.	Low — creating a new thread is faster and requires fewer resources.
Failure Impact	Crash in one process does not affect others.	Crash in one thread may affect the entire process.
Context Switching	More costly (involves changing memory context).	Less costly (same memory space).
Examples	Web browser launching multiple processes (tabs).	Multiple threads handling loading, rendering, and user input in a single tab.

- **Draw and explain the process state diagram (refer to Figure 1).**



- The five-state process model explains how a process moves through different stages as it is managed by the operating system. A process starts in the **New** state, where it is created and given resources like memory. It then moves to the **Ready** state, waiting for its turn to use the CPU. When the CPU starts executing it, the process enters the **Running** state. If the process needs to wait for an input or event, it moves to the **Waiting** state until the event is completed. Once finished, it goes back to **Ready** or, if the task is done, moves to the **Terminated** state, where all its resources are released.
  Transitions occur between these states—for example, from **New to Ready** when admitted, **Ready to Running** when dispatched, **Running to Waiting**

when waiting for I/O, **Waiting to Ready** when I/O is complete, and **Running to Terminated** when the process finishes. This model helps the operating system organize and control how processes are executed efficiently.

- **Explain two IPC methods with real-world examples.**
  1. **Shared Memory**
     - This is a method where it allows multiple processes to access the same region of memory for direct communication.It is the fastest IPC method because of how data can be shared without involving the operating system. A common example of a Shared Memory is a we browser like Google Chrome or Opera where multiple rendering processes share cached data like image or scripts

  2. **Message Passing**
     - Process communicating by sending and receiving messages through the operating system. This method usually used in where distributed systems processes do not share memory.One example is pipes which enables one-way communication between parent and child processes.

PART B:

- **Explain the differences between paging and segmentation.**

  - Paging and Segmentation are both memory management techniques used by an operating system to allocate memory efficiently.But the difference is that paging divides memory into fixed size block called page that are mapped to physical memory frames using a page table. In contrast segmentation divides memory into variable size segments base on program logical dividision like codes,data or stack.

- **Describe the TLB and its impact on performance.**

  - The Translation Lookaside Buffer (TLB) is a small and fast cache located in the CPU that stores recent page table entry.When the CPU needs translate logical address to physical address it first checks the TLB then if the the required page entry is found the address translation quickly happens while in the other hand if the entry is not found The CPU access it in the main memory which takes time.It improves the system performance by reducing the number of memory access needed for the address translation meaning the higher TLB hit rate means faster overall memory access

- **Calculate Effective Memory Access Time (EMAT) for the scenario in Figure 2.**

Memory access time	= 100 ns	,h	= TLB hit ratio
TLB access time	= 10 ns	,Ttlb	= Time to access TLB
TLB hit rate	= 80%	,Tmem	= Time access main memory

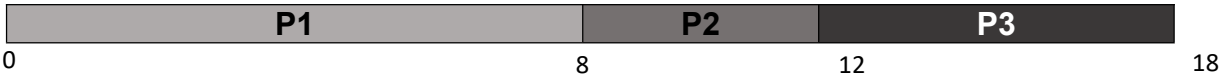
EMAT = (h) x (Ttlb + Tmem) + (1-h)x(Ttlb + 2Tmem)
= (0.8) x (10+100) + (0.2)(10+200)
EMAT = 84 + 42 = 130ns

PART C:

- **Compare FCFS vs. Round Robin scheduling (see Figure 3).**

Aspect	First-Come, First-Served (FCFS)	Round Robin (RR)
Scheduling Order	Processes are executed in the order they arrive (like a queue).	Each process gets a fixed time slice (quantum) and is cycled in order.
Preemption	**Non-preemptive** — once a process starts, it runs until completion.	**Preemptive** — a process is paused after its quantum expires, and the next process runs.
Fairness	May cause long waiting times for short processes (convoy effect).	More fair — every process gets CPU time regularly.
Response Time	Can be high for shorter jobs arriving later.	Lower and more predictable, suitable for time-sharing systems.
Efficiency	Better for batch systems where processes have similar length.	Better for interactive systems (like multitasking OS).
Aspect	First-Come, First-Served (FCFS)	Round Robin (RR)
Scheduling Order	Processes are executed in the order they arrive (like a queue).	Each process gets a fixed time slice (quantum) and is cycled in order.

- **For P1(8ms), P2(4ms), P3(6ms), create Gantt charts and find avg. waiting time for FCFS and Round Robin (quantum=3ms)**



Average Waiting Time (AWT) = (0 + 8 +12) / 3 = 6.67 ms

P1 Waiting Time = 0 + 6 + 4 = 10
P2 Waiting Time = 3 + 9 = 9
P3 Waitign Time = 6 + 4 = 10

Average Waiting Time = (10 + 9 + 10)/3 =9.67ms

PART D:

- **Describe the Dining Philosophers problem (see Figure 4) and its solutions.**

The **Dining Philosophers Problem** describes a situation where five philosophers sit around a table with one fork placed between each of them. To eat, each philosopher needs two forks—one on the left and one on the right. The problem occurs when all philosophers pick up one fork at the same time, causing a **deadlock** because no one can pick up the second fork. To solve this, synchronization techniques such as **semaphores** or **monitors** are used to ensure that a philosopher only picks up both forks if both are available or to limit the number of philosophers eating at once. This problem illustrates how processes must coordinate shared resource access to prevent deadlock.

- **Explain the Producer-Consumer problem (see Figure 5) and the use of semaphores.**

The Producer-Consumer Problem, also known as the Bounded Buffer Problem, is concerned with synchronization between two sorts of processes: producers, which create data or products, and consumers, who use or dispose of them. Both use a common, finite-sized buffer in which the producer stores objects and the consumer retrieves them. If two processes access the buffer at the same time without sufficient synchronization, difficulties such as race situations, buffer overflows, or data inconsistency may arise. To address this issue, semaphores are used to manage access: one semaphore counts the number of empty slots, another counts the number of full slots, and a mutex assures that only one process updates the buffer at a time. For example, in a real-world system such as a print queue, the producer adds print jobs to the queue, and the consumer (the printer) processes them one by one. This assures a conflict-free operation, data integrity, and effective resource consumption.

- **Analyze Reader-Writer problem scenarios.**

The Reader-Writer Problem refers to situations in which numerous programs require access to common data, such as a database or file. Readers are processes that simply read data, whereas writers modify it. The problem emerges when numerous readers and writers attempt to access the same data at the same time. Inconsistencies can develop when a writer alters facts while a reader is reading. As a result, synchronization is required to ensure that numerous readers can access the data simultaneously, while authors must have exclusive access. This is accomplished by employing reader-writer locks or semaphores to regulate access. For example, in a library database, several users (readers) may look up book information simultaneously, but when a librarian (writer) updates the entries, no readers are allowed until the edit is complete