Internal Assessment Test 1 – DEC 2025

Operating System							Sub Code:	MMC104
29/12/2025	Duration:	90 min's	Max Marks:	50	Sem:	I	Branch:	MCA

PART I

1. Explain in brief about functions and services of an OS.

An **Operating System (OS)** is system software that acts as an interface between the user and computer hardware. It manages system resources efficiently and provides a convenient environment for program execution. The major **functions** and **services** of an operating system are explained below.

Functions of an Operating System

1. Process Management

The OS is responsible for managing processes in the system. It creates processes, schedules them for execution, and terminates them when they are completed. The OS ensures efficient CPU utilization through techniques like multitasking, multithreading, and context switching.

2. Memory Management

Memory management involves allocating and deallocating main memory to programs as required. The OS keeps track of which part of memory is in use and by which process. It also supports virtual memory, paging, and segmentation to optimize memory utilization.

3. File System Management

The OS manages files stored on secondary storage devices. It provides mechanisms for creating, deleting, reading, writing, and organizing files into directories. It also maintains file permissions to ensure data security.

4. Device Management

The OS controls and coordinates the operation of input and output devices such as keyboards, printers, and disks. Device drivers are used to communicate with hardware devices. The OS also handles buffering, caching, and spooling.

5. Secondary Storage Management

The OS manages disk space by keeping track of free space, allocating storage to files, and scheduling disk access to improve performance.

6. Security and Protection

The OS protects system resources and user data from unauthorized access. It provides authentication mechanisms such as passwords and access control lists to ensure data integrity and privacy.

Services of an Operating System

1. Program Execution

The OS loads programs into memory, executes them, and handles their termination. It provides the environment needed for program execution.

2. Input/Output Operations

The OS offers standardized interfaces for I/O operations so that users and applications do not need to worry about hardware-specific details.

3. File Manipulation Services

The OS allows users and programs to create, modify, read, write, and delete files in a secure and organized manner.

4. Communication Services

The OS supports inter-process communication (IPC) through mechanisms like pipes, shared memory, and message passing. It also facilitates network communication.

5. Error Detection and Handling

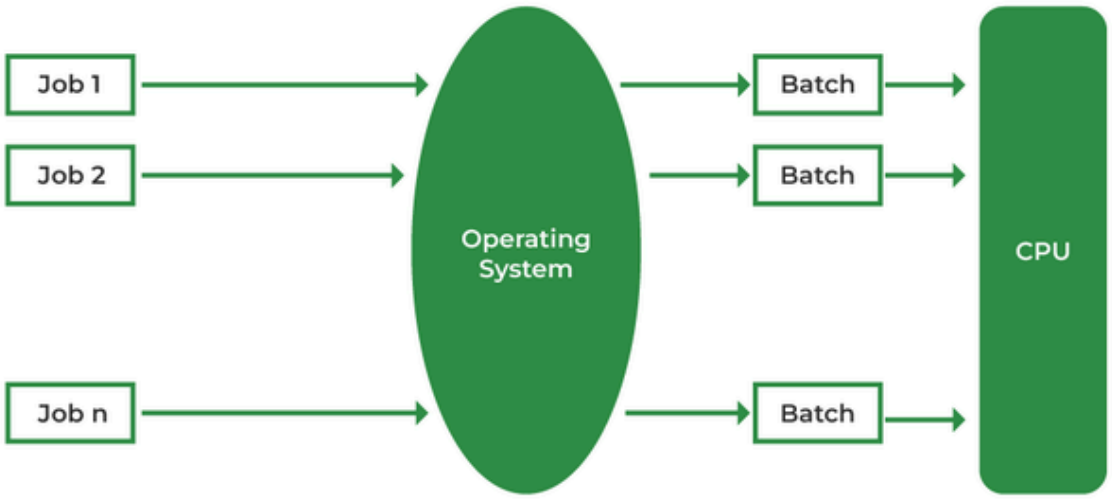
The OS continuously monitors the system for errors such as memory faults, I/O failures, and hardware malfunctions, and takes corrective actions.

6. User Interface

The OS provides a user-friendly interface in the form of a Command Line Interface (CLI) or Graphical User Interface (GUI) for user interaction.

2. **Explain batch processing and real time OS in detail**

The batch-processing operating system was very popular in the **1970s**. In batch operating system the jobs were performed in batches. This means Jobs having similar requirements are grouped and executed as a group to speed up processing. Users using batch operating systems do not interact with the computer directly. Each user prepares their job using an offline device for example a punch card and submits it to the computer operator. Once the programmers have left their programs with the operator, they sort the programs with similar needs into batches.



Working of Batch Processing Operating Systems

The Batch operating system is a new, open-source operating system that is being developed by the Berkeley Open Infrastructure for Network Computing (BOINC) project. A batch is a segmental operating system that can be collected from smaller pieces, allowing it to be modified to specific needs.

The Batch project is led by Berkeley computer scientist Pieter Abbeel, who is also the project's primary code contributor. The batch is intended to be lightweight and efficient and is intended to be used primarily in grid computing environments.

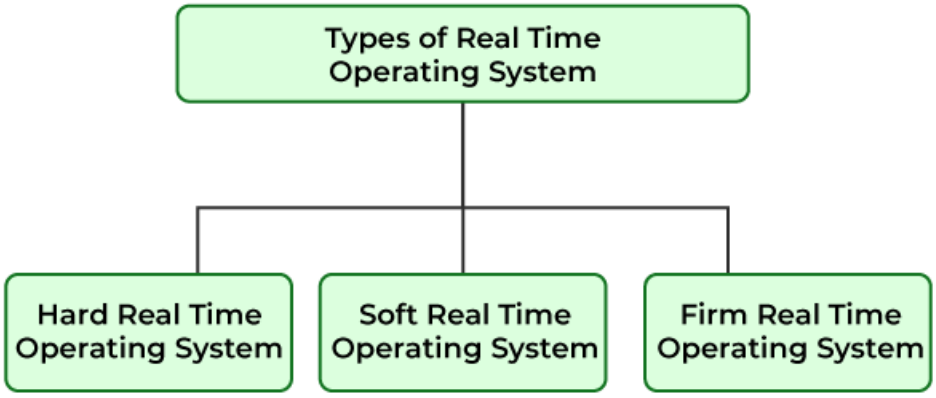
The Batch project is presently in the progress stage, and there is still a lot of work to be done before the operating system is ready for use. However, growth has been made in recent months, and the project is probable to be completed within the next year.

There are many types of batch operating systems. One popular type is the scheduled batch system. This type of system is used to control the execution of a series of tasks or jobs. Other types of batch systems include the interactive batch system, the real-time batch system, and the concurrent batch system.

Real-Time Operating System (RTOS)

A real-time operating system (RTOS) is a special kind of operating system designed to handle tasks that need to be completed quickly and on time. Unlike general-purpose operating systems (GPOS), which are good at multitasking and user interaction, RTOS focuses on doing things in real time.

The idea of real-time computing has been around for many years. The first RTOS was created by Cambridge University in the 1960s. This early system allowed multiple processes to run at the same time, each within strict time limits.



- **Hard Real-Time Operating System :** These operating systems guarantee that critical tasks are completed within a range of time. For example, a robot is hired to weld a car body. If the robot welds too early or too late, the car cannot be sold, so it is a hard real-time system that requires complete car welding by the robot hardly on time., scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.
- **Soft Real-Time Operating System :** This operating system provides some relaxation in the time limit. For example - Multimedia systems, digital audio systems, etc. Explicit, programmer-defined, and controlled processes are encountered in real-time systems. A separate process is changed by handling a single external event. The process is activated upon the occurrence of the related event signaled by an interrupt.

PART II

3. Write a short note on implementation of system calls.

User programs cannot directly access hardware or critical OS resources because it would make the system unstable and insecure. To maintain safety, the operating system provides system calls — controlled interfaces that allow user programs to request services from the kernel. These calls act as a gateway between user mode and kernel mode. System Calls are,

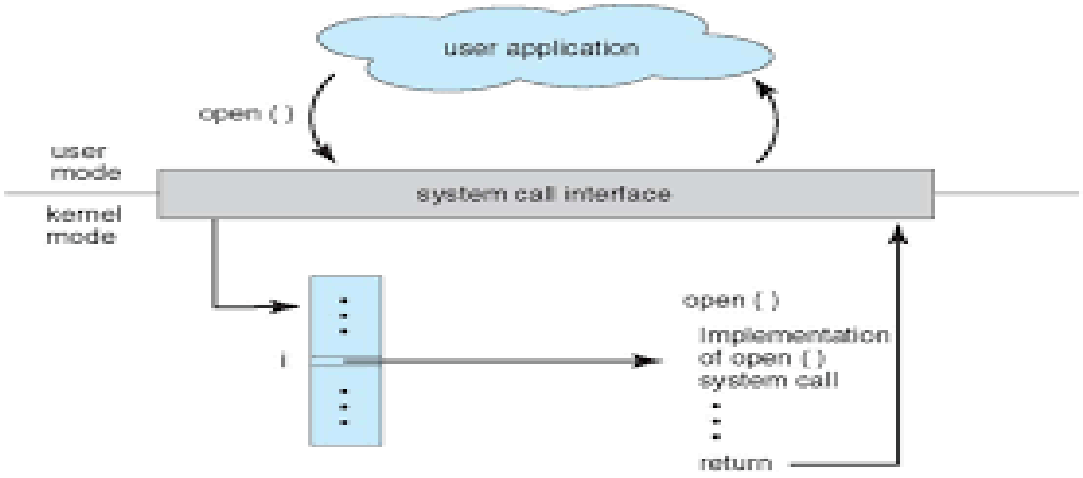
- A way for programs to interact with the operating system.
- Provide the services of the operating system to the user programs.
- Only entry points into the kernel are executed in kernel mode.

A system call is a controlled entry point that allows a user program to request a service from the operating system. Here's how it works:

- The user program executes a system call instruction (e.g., using syscall or int 0x80).

- The CPU switches from user mode → kernel mode for safe execution.
- The kernel identifies the system call number and performs the requested operation (file access, process creation, memory allocation, etc.).
- After completing the task, the kernel switches back to user mode.
- The result (success/failure/data) is returned to the program.
- Without system calls, every program would need its own way to access hardware, leading to inconsistent and insecure systems.

System calls do not always cause context switching. They primarily involve a **mode switch** from user mode to kernel mode. A **context switch happens only when the calling process is blocked**, not during every system call.



Types of System Calls
Services provided by an OS are typically related to any kind of operation that a user program can perform like creation, termination, forking, moving, communication, etc. Similar types of operations are grouped into one single system call category. System calls are classified into the following File System: Used to create, open, read, write, and manage files and directories.

- Process Control: Used to create, execute, synchronize, and terminate processes.
- Memory Management: Used to allocate, deallocate, and manage memory for processes.
- Interprocess Communication (IPC): Used for data exchange and communication between different processes.

- Device Management: Used to request and release devices, and to perform read/write operations on them.

4. Differentiate between preemptive and non- preemptive scheduling.

CPU scheduling in operating systems is the method of selecting which process in the ready queue will execute on the CPU next. It aims to utilise the processor efficiently while minimising waiting and response times. By determining an optimal execution order, CPU scheduling enhances overall system performance, supports smooth multitasking, and improves the user experience.

Scheduling can be broadly classified into two types: Preemptive and Non-Preemptive.

Preemptive Scheduling	Non-Preemptive Scheduling
In this resources(CPU Cycle) are allocated to a process for a limited time.	Once resources(CPU Cycle) are allocated to a process, the process holds it till it completes its burst time or switches to waiting state
Process can be interrupted in between.	Process can not be interrupted until it terminates itself or its time is up
If a process having high priority frequently arrives in the ready queue, a low priority process may starve	If a process with a long burst time is running CPU, then later coming process with less CPU burst time may starve

It has overheads of scheduling the processes	It does not have overheads
Average process response time is less	Average process response time is high
Decisions are made by the scheduler and are based on priority and time slice allocation	Decisions are made by the process itself and the OS just follows the process's instructions
More as a process might be preempted when it was accessing a shared resource.	Less as a process is never preempted.
Examples of preemptive scheduling are Round Robin and Shortest Remaining Time First	Examples of non-preemptive scheduling are First Come First Serve and Shortest Job First

PART III

5.Illustrate with a neat sketch, the process states and process control block.

When a process runs, it goes through many states. Distinct operating systems have different stages, and the names of these states are not standardised. In general, a process can be in one of the five states listed below at any given time.

Start
When a process is started/created first, it is in this state.

Ready

Here, the process is waiting for a processor to be assigned to it. Ready processes are waiting for the operating system to assign them a processor so that they can run. The process may enter this state after starting or while running, but the scheduler may interrupt it to assign the CPU to another process.

Running

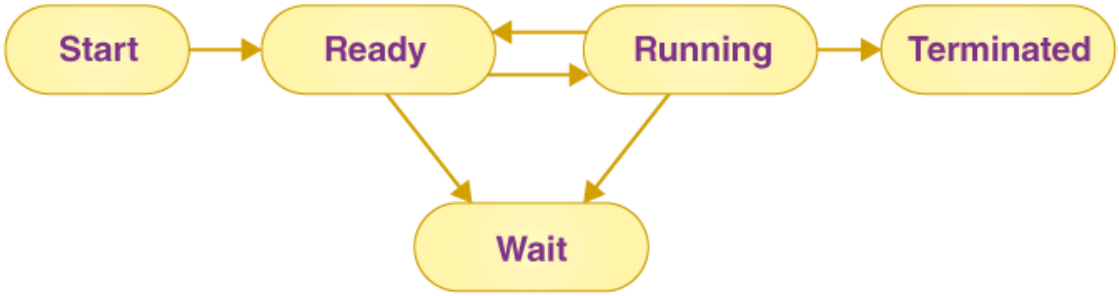
When the OS scheduler assigns a processor to a process, the process state gets set to running, and the processor executes the process instructions.

Waiting

If a process needs to wait for any resource, such as for user input or for a file to become available, it enters the waiting state.

Terminated or Exit

The process is relocated to the terminated state, where it waits for removal from the main memory once it has completed its execution or been terminated by the operating system.



Process Control Block (PCB)

Every process has a process control block, which is a data structure managed by the operating system. An integer process ID (or PID) is used to identify the PCB. As shown below, PCB stores all of the information required to maintain track of a process.

Process state

The process's present state, such as whether it's ready, waiting, running, or whatever.

Process privileges

This is required in order to grant or deny access to system resources.

Process ID

Each process in the OS has its own unique identifier.

Pointer

It refers to a pointer that points to the parent process.

Program counter

The program counter refers to a pointer that points to the address of the process's next instruction.

CPU registers

Processes must be stored in various CPU registers for execution in the running state.

CPU scheduling information

Process priority and additional scheduling information are required for the process to be scheduled.

Memory management information

This includes information from the page table, memory limitations, and segment table, all of which are dependent on the amount of memory used by the OS.

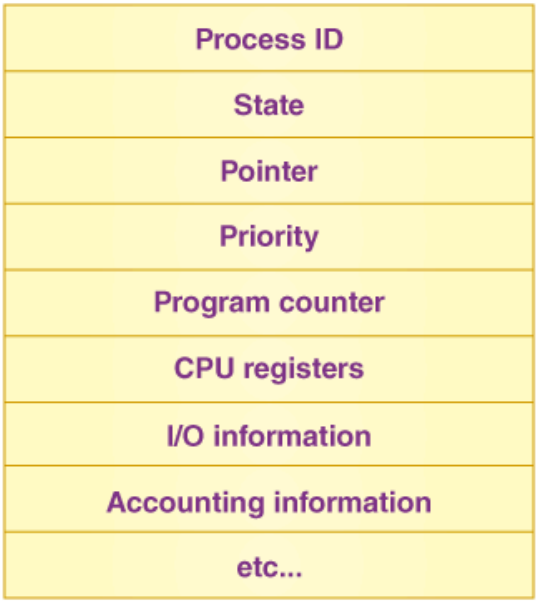
Accounting information

This comprises CPU use for process execution, time constraints, and execution ID, among other things.

IO status information

This section includes a list of the process's I/O devices.

The PCB architecture is fully dependent on the operating system, and different operating systems may include different information. A simplified diagram of a PCB is shown below.



6. Explain about the basic concepts of process scheduling and also common criteria used to evaluate scheduling algorithms?

Process scheduling in OS is an essential function of an operating system (OS) is an essential function that manages how different programs (processes) share the CPU. In a multitasking environment where several processes are generally executed simultaneously, the operating system has to choose which process will be granted CPU time and for how long. Such a procedure keeps the system running smoothly and makes sure that resources are utilized efficiently.

Process scheduling in an operating system is practically dependent on the basic operations that can be done on processes during their lifecycle. Such operations are:

1. Process Creation

- A new process can be created by an existing process, typically using system calls like fork (in Unix-based systems) or spawn.
- The creating process is known as the parent process, while the new one is called the child process.
- The child process may inherit resources from its parent or have its own separate resources, depending on the system design.

- Creation is often triggered when a new program needs to run or a task requires a separate execution flow.

2. Process Termination

- A process finishes execution and is terminated using system calls such as exit.
- Termination can occur voluntarily (process completes its task) or involuntarily (killed by the OS or parent).
- When a process terminates, its resources are released and its entry is removed from the process table.
- If a parent process ends before its child, the child becomes an orphan process, which is then typically re-parented to a special system process.

3. Process Suspension and Reintroduction

- Sometimes, a process may be suspended (swapped out of memory) to free up resources or manage memory pressure—this is often handled by the medium-term scheduler.
- Suspended processes can later be reintroduced (swapped back in) to resume execution from where they left off.
- This operation is crucial for balancing CPU and memory usage, especially when managing many I/O-bound tasks.

4. Parent and Child Relationships

- Parent and child processes may communicate and synchronize their operations.
- The parent may be allowed to wait for the child to finish, or they can both operate separately.
- System calls such as wait give the opportunity for the parent to stop it will be a moment only until the child disappears, thus providing an orderly way to manage resources.

5. Special Process States

- Orphan processes: Created when a parent terminates before its child; the OS typically reassigns them to a system process.

- Zombie processes: Processes that have run to completion but still have a record in the process table (waiting for the parent to get their termination status).

Different **CPU Scheduling algorithms** have different structures and the choice of a particular algorithm depends on a variety of factors.

- **CPU Utilization:** The main purpose of any CPU algorithm is to keep the CPU as busy as possible. Theoretically, CPU usage can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the system load.
- **Throughput:** The average CPU performance is the number of processes performed and completed during each unit. This is called throughput. The output may vary depending on the length or duration of the processes.
- **Turn Round Time:** For a particular process, the important conditions are how long it takes to perform that process. The time elapsed from the time of process delivery to the time of completion is known as the conversion time. Conversion time is the amount of time spent waiting for memory access, waiting in line, using CPU and waiting for I/O.
- **Waiting Time:** The Scheduling algorithm does not affect the time required to complete the process once it has started performing. It only affects the waiting time of the process i.e. the time spent in the waiting process in the ready queue.
- **Response Time:** In a collaborative system, turn around time is not the best option. The process may produce something early and continue to computing the new results while the previous results are released to the user. Therefore another method is the time taken in the submission of the application process until the first response is issued. This measure is called response time.

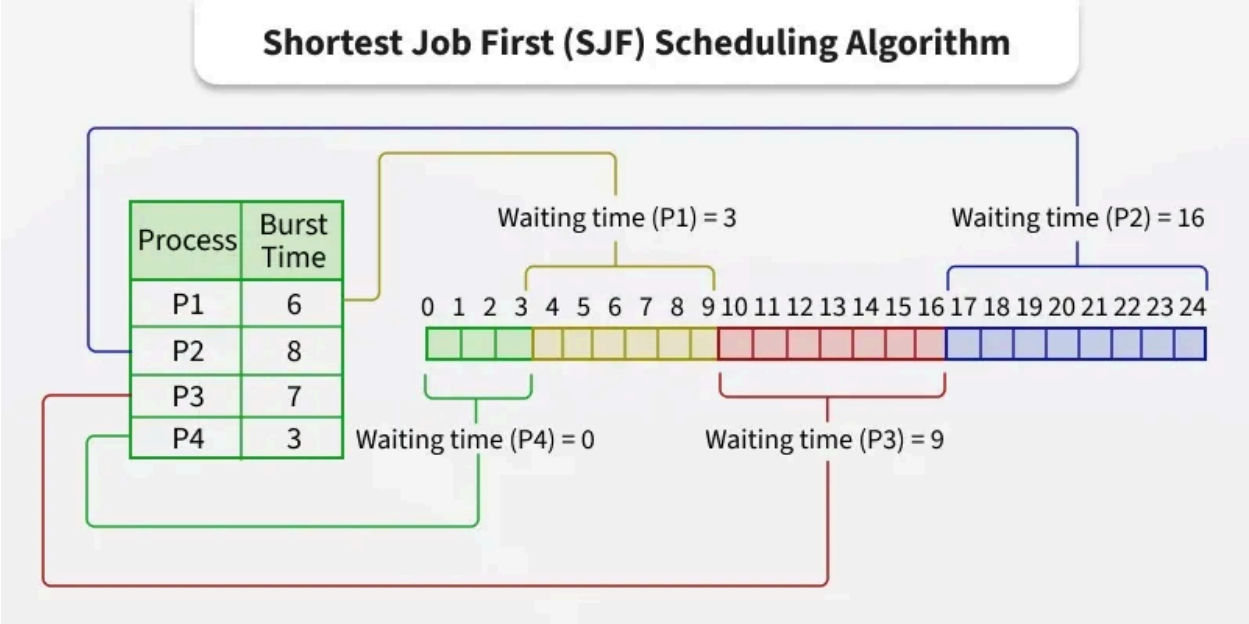
PART IV

7.Explain in detail about the SJF scheduling algorithm with examples

Shortest Job First (SJF) or Shortest Job Next (SJN) is a scheduling process that selects the waiting process with the smallest execution time to execute next. This scheduling method may or may not be preemptive. Significantly reduces the average waiting time for other processes waiting to be executed.

Estimation Formula Concept in SJF Scheduling

The Shortest Job First (SJF) Scheduling algorithm selects the process with the smallest burst time for execution. But in some cases, the exact burst time of a process may not be known in advance. In such scenarios, an estimation formula is used to predict the next burst time based on the previous burst times.



Characteristics of SJF Scheduling

- Shortest Job first has the advantage of having a minimum average waiting time among all operating system scheduling algorithms.
- It is associated with each task as a unit of time to complete.
- It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of aging.

Example: Consider the following table of arrival time and burst time for three processes **P1, P2 and P3**.

Process	Burst Time	Arrival Time

P1	6 ms	0 ms
P2	8 ms	2 ms
P3	3 ms	4 ms

8.What is the critical section problem? explain about the criteria in detail.
A critical section is a part of a program where shared resources (like memory, files, or variables) are accessed by multiple processes or threads. To avoid problems such as race conditions and data inconsistency, only one process/thread should execute the critical section at a time using synchronization techniques. This ensures that operations on shared resources are performed safely and predictably.

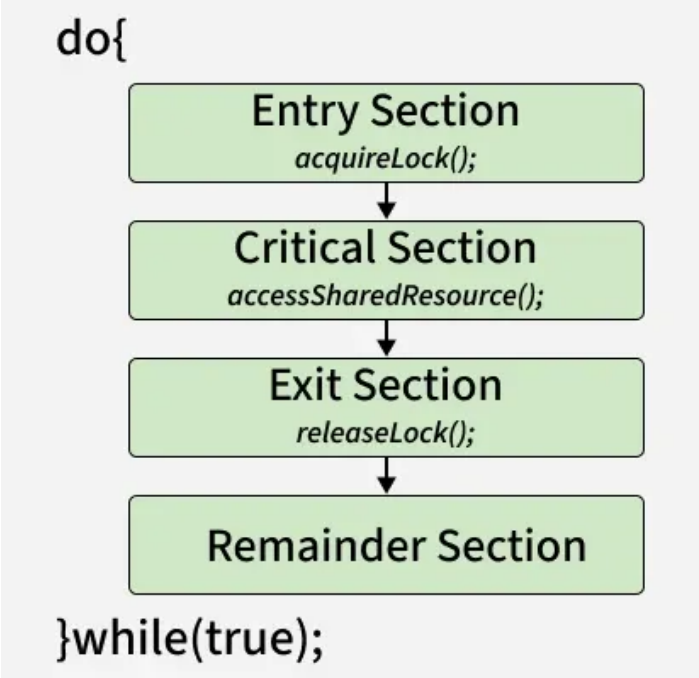
Structure of a Critical Section
1. Entry Section

- The process requests permission to enter the critical section.
- Synchronization tools (e.g., mutex, semaphore) are used to control access.

2. Critical Section: The actual code where shared resources are accessed or modified.

3. Exit Section: The process releases the lock or semaphore, allowing other processes to enter the critical section.

4. **Remainder Section:** The rest of the program that does not involve shared resource access.



Requirements of Critical Section Solutions

1. Mutual Exclusion

- At most one process can be inside the critical section at a time.
- Prevents conflicts by ensuring no two processes update the shared resource simultaneously.

2. Progress

- If no process is in the critical section, and some processes want to enter, the choice of who enters next should not be postponed indefinitely.
- Ensures that the system continues to make progress rather than getting stuck.

3. Bounded Waiting

- There must be a limit on how long a process waits before it gets a chance to enter the critical section.
- Prevents **starvation**, where one process is repeatedly bypassed while others get to execute.

PART V

9.Calculate the average waiting time, turnaround time for

i) priority ii)Round Robin(tq=2ms) with the following set of processes.

Process	**P1**	**P2**	**P3**	**P4**	**P5**
Burst Time	**10**	**1**	**2**	**1**	**5**
Priority	**3**	**1**	**3**	**4**	**5**

Priority Scheduling

Process	Burst Time (ms)	Priority	Start Time (ms)	Completion Time (ms)	Waiting Time (ms)	Turnaround Time (ms)
P2	1	1	0	1	0	1
P1	10	3	1	11	1	11
P3	2	3	11	13	11	13
P4	1	4	13	14	13	14
P5	5	5	14	19	14	19

Average Waiting Time	**7.8 ms**
Average Turnaround Time	**11.6 ms**

Round Robin

Process	Burst Time (ms)	Completion Time (ms)	Turnaround Time (ms)	Waiting Time (ms)
P1	10	19	19	9
P2	1	3	3	2
P3	2	5	5	3
P4	1	6	6	5
P5	5	15	15	10

Metric	**Value**
Average Waiting Time	**5.8 ms**
Average Turnaround Time	**9.6 ms**

10.Explain about the producer-consumer problem of synchronization

The Producer-Consumer problem is a classic example of a synchronization problem in operating systems. It demonstrates how processes or threads can safely share resources without conflicts. This problem belongs to the process synchronization domain, specifically dealing with coordination between multiple processes sharing a common buffer.

- **Producers**: Generate data items and place them in a shared buffer.
- **Consumers**: Remove and process data items from the buffer.

The main challenge is to ensure:

1. A producer does not add data to a full buffer.
2. A consumer does not remove data from an empty buffer.
3. Multiple producers and consumers do not access the buffer simultaneously, preventing race conditions.

Problem Statement
Consider a fixed-size buffer shared between a producer and a consumer.

- The producer generates an item and places it in the buffer.
- The consumer removes an item from the buffer.

The buffer is the critical section. At any moment:

- A producer cannot place an item if the buffer is full.
- A consumer cannot remove an item if the buffer is empty.

To manage this, we use three semaphores:

- mutex – ensures mutual exclusion when accessing the buffer.
- full – counts the number of filled slots in the buffer.
- empty – counts the number of empty slots in the buffer.

Producer

```
do {

  // Produce an item
   wait(empty);    // Check for empty slot
   wait(mutex);    // Enter critical section

  // Place item in buffer

  signal(mutex);  // Exit critical section
   signal(full);   // Increase number of full slots
} while (true);
```

Consumer

```
do {
   wait(full);     // Check for filled slot
   wait(mutex);    // Enter critical section

  // Remove item from buffer
```

```
    signal(mutex);  // Exit critical section
    signal(empty);  // Increase number of empty slots
} while (true);
```