

OS MID-TERM — TOP 10 IMPORTANT TOPICS (4 MARKS)

1. CPU Scheduling + FCFS

CPU Scheduling: OS selects a process from the ready queue and allocates the CPU.
Objectives: maximize CPU utilization & throughput; minimize waiting, turnaround and response time.

FCFS: non-preemptive; process arriving first executes first.

Formulas: $TAT = CT - AT$; $WT = TAT - BT$.

Numerical: Gantt chart $\rightarrow CT \rightarrow TAT \rightarrow WT \rightarrow$ averages.

2. SJF + SRTF

SJF: selects the process with the shortest Burst Time; normally non-preemptive.

SRTF: preemptive version of SJF; if a new process has a shorter remaining time, current process is preempted.

Remember: SJF = non-preemptive; SRTF = preemptive.

Numerical: Gantt chart $\rightarrow CT \rightarrow TAT \rightarrow WT$.

3. Round Robin

RR is a preemptive scheduling algorithm using a fixed Time Quantum.

Each process gets the CPU for the quantum; unfinished process goes to the end of the ready queue.

Key: preemptive + time quantum + fair CPU sharing.

Numerical: Gantt chart $\rightarrow CT \rightarrow TAT \rightarrow WT \rightarrow$ averages.

4. Deadlock + 4 Conditions

Deadlock: two or more processes remain blocked because each waits for a resource held by another.

Four necessary conditions:

- 1) Mutual Exclusion — one process uses a resource at a time.
- 2) Hold & Wait — holds one resource while waiting for another.
- 3) No Preemption — resource cannot be forcibly taken.
- 4) Circular Wait — processes form a circular waiting chain.

Shortcut: M-H-N-C.

5. Deadlock Avoidance Algorithms

Deadlock avoidance checks a resource request before allocation so the system stays in a safe state.

Safe $\rightarrow$ allocate; Unsafe $\rightarrow$ postpone/deny.

Main idea: examine each request and test whether granting it keeps the state safe.

Banker's Algorithm is a common deadlock-avoidance algorithm.

6. Resource Graph Model

Resource Allocation Graph represents processes and resources graphically.

Process = circle; Resource = rectangle.

$P \rightarrow R$ = Request edge; $R \rightarrow P$ = Allocation edge.

A cycle is important for deadlock analysis; with single-instance resources, a cycle indicates deadlock.

7. Semaphore + wait()/signal()

Semaphore: synchronization mechanism controlling access to shared resources.

wait(S): acquire resource; decrement S; block if unavailable.

signal(S): release resource; increment S; wake a waiting process.

Binary semaphore: 0/1, mainly mutual exclusion.

Counting semaphore: 0 to n, for multiple resource instances.

8. IPC — Shared Memory vs Message Passing

IPC allows processes to communicate, exchange data and coordinate.

Shared Memory: common memory area; fast; synchronization needed.

Message Passing: send/receive messages; no direct shared memory; simpler/safer but kernel overhead.

Remember: Shared Memory = shared area; Message Passing = messages.

9. Critical Section + Mutual Exclusion

Critical Section: part of a program where shared resources are accessed/modified.

Structure: Entry $\rightarrow$ Critical $\rightarrow$ Exit $\rightarrow$ Remainder.

Requirements:

- 1) Mutual Exclusion — at most one process inside.
- 2) Progress — waiting process should not be postponed unnecessarily.
- 3) Bounded Waiting — no indefinite waiting.

Mutual exclusion prevents simultaneous conflicting access.

10. Threads + Multithreading

Thread: lightweight unit of execution within a process.

Each thread has its own Program Counter, registers and stack; threads of a process share code, data and OS resources.

Multithreading: multiple threads execute within one process.

Advantages: better performance, responsiveness, concurrency, CPU utilization and resource sharing.