DELHI SKILL AND ENTREPRENEURSHIP UNIVERSITY

End-Semester Examination (Set 3)

Subject Name:	Operating Systems
Subject Code:	BT-CS-ES502
Duration:	3 Hours
Max. Marks:	100

Instructions:

1. This paper contains **Two Sections**: Section A and Section B.
2. **Section A** is compulsory and contains Short Answer Questions.
3. **Section B** contains Descriptive and Application-based Questions.
4. Assume necessary data if not given.

SECTION A – Short Answer Questions

Attempt all questions. Each question carries 2 marks. **[10 × 2 = 20 Marks]**

Q1. Differentiate between a 'Zombie Process' and an 'Orphan Process'.

Answer: A **Zombie Process** has completed execution but its entry remains in the process table because the parent hasn't read its exit status (via 'wait()'). An **Orphan Process** is a running process whose parent has finished or terminated; it is typically adopted by the 'init' process.

Q2. Define 'Spooling'. How is it different from Buffering?

Answer: Spooling (Simultaneous Peripheral Operations On-Line) uses a disk as a large buffer to hold jobs (like print tasks) until the device is ready. While buffering overlaps I/O of one job with its computation, spooling overlaps the I/O of one job with the computation of *other* jobs.

Q3. What is 'Dual Mode' operation in an Operating System?

Answer: To protect the system, hardware provides at least two modes: **User Mode** (for executing user applications) and **Kernel/Monitor Mode** (for executing OS instructions). Privileged instructions can only be executed in Kernel Mode.

Q4. Explain the concept of 'Compaction' in Memory Management.

Answer: Compaction is a technique used to resolve **External Fragmentation**. The OS shuffles memory contents to place all free memory blocks together in one large contiguous block, making it usable for processes that require large memory chunks.

Q5. What is a 'Trap' (or Software Interrupt)?

Answer: A trap is a software-generated interrupt caused either by an error (e.g., division by zero, invalid memory access) or by a specific request from a user program that an operating system service be performed (a System Call).

Q6. Define 'Hard Real-Time' vs 'Soft Real-Time' systems.

Answer: Hard Real-Time systems guarantee that critical tasks complete within a defined deadline; missing a deadline causes total system failure (e.g., pacemaker). **Soft Real-Time** systems prioritize critical tasks but do not guarantee strict deadlines; missing one degrades performance but is not fatal (e.g., video streaming).

Q7. What is the function of the 'Medium-Term Scheduler'?

Answer: The Medium-Term Scheduler manages the degree of multiprogramming by **swapping** processes in and out of main memory. It removes partially executed processes from memory (swap out) to reduce contention and reintroduces them later (swap in).

Q8. What is 'Busy Waiting'?

Answer: Busy waiting occurs when a process repeatedly checks a condition (like a loop checking a lock variable) to see if it can proceed. This wastes CPU cycles that could be used by other processes.

Q9. Describe the 'Bit Vector' method for Free Space Management.

Answer: In the Bit Vector method, free space on the disk is represented by a bitmap. Each block is represented by 1 bit. If the bit is '1', the block is free; if '0', the block is allocated. It is efficient for finding the first free block or contiguous blocks.

Q10. What is a Page Fault?

Answer: A Page Fault occurs when a program tries to access a page that is mapped in its logical address space but is not currently present in the physical main memory (RAM). This triggers the OS to fetch the page from the backing store (disk).

SECTION B – Descriptive Questions

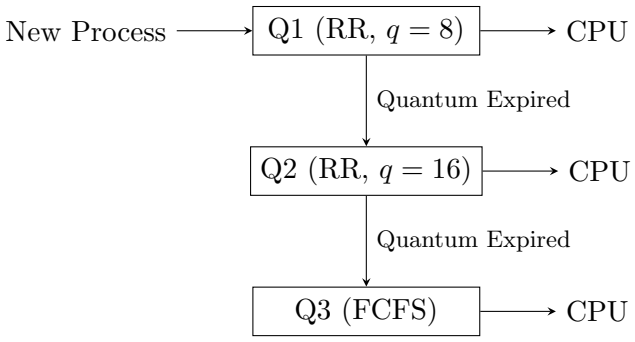
This section contains analytical and descriptive questions.

Part I: Answer the following (5 Marks each) **[4 × 5 = 20 Marks]**

Q11. Explain the Multilevel Feedback Queue scheduling algorithm with a diagram.

Answer: Multilevel Feedback Queue (MLFQ) scheduling allows processes to move between queues. It separates processes according to their CPU burst characteristics.

- If a process uses too much CPU time, it is moved to a lower-priority queue.
- If a process is I/O-bound (gives up CPU before quantum expires), it may stay in high priority.
- This prevents starvation using **Aging** (moving old processes up).



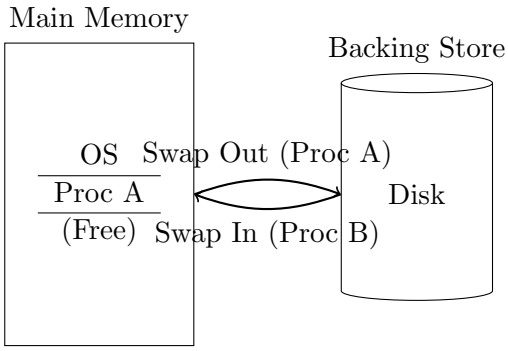
Q12. Distinguish between Deadlock Prevention and Deadlock Avoidance.

Answer:

Deadlock Prevention	**Deadlock Avoidance**
Ensures at least one of the four necessary conditions (Mutual Exclusion, Hold & Wait, No Preemption, Circular Wait) cannot hold.	Allows the conditions to exist but analyzes the system state dynamically to ensure a circular wait never occurs.
Static approach: Rules are enforced regardless of current state.	**Dynamic approach:** Checks current resource availability before granting requests.
Example: Spooling everything (No Hold & Wait), or Numbering resources (No Circular Wait).	Example: Banker's Algorithm.

Q13. Explain the concept of 'Swapping' with a schematic diagram.

Answer: Swapping is a memory management technique where a process can be swapped temporarily out of main memory to a backing store (disk) and then brought back into memory for continued execution. This allows the total physical address space of all processes to exceed the real physical memory.



Q14. Compare Tree-Structured Directories and Acyclic-Graph Directories.

Answer:

- **Tree-Structured:** A hierarchical structure where a root directory has sub-directories. Every file has a unique path. *Limitation:* Files cannot be shared; the same file cannot appear in two different directories naturally.
- **Acyclic-Graph:** An extension of the tree structure that allows directories to share sub-directories and files (using links or aliases). The graph contains no cycles. *Benefit:* Allows file sharing. *Challenge:* Deletion is complex (dangling pointers) and backup is harder (files traversed multiple times).

Part II: Answer the following (10 Marks each) [6 × 10 = 60 Marks]

Q15. CPU Scheduling (SJF Non-Preemptive):
Consider the following processes. The CPU burst times are given in milliseconds.

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

Using **Non-Preemptive Shortest Job First (SJF)**:

(a) Draw the Gantt chart.

(b) Calculate Average Turnaround Time.

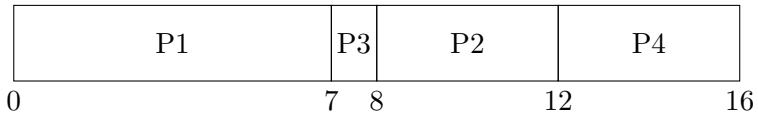
(c) Calculate Average Waiting Time.

Answer: Logic: Non-Preemptive means once a process gets the CPU, it runs to completion. Decision is made only when the current process finishes.

Execution Trace:

- $t = 0$: Only P1 available. **P1 runs** (0 to 7).
- $t = 7$: P1 finishes. Available: P2 (Arr 2, Burst 4), P3 (Arr 4, Burst 1), P4 (Arr 5, Burst 4).
- Comparison: P3 has shortest burst (1). **P3 runs** (7 to 8).
- $t = 8$: P3 finishes. Available: P2 (Burst 4), P4 (Burst 4).
- Tie-breaking: FCFS is usually used. P2 arrived before P4. **P2 runs** (8 to 12).
- $t = 12$: P2 finishes. Only P4 remains. **P4 runs** (12 to 16).

1. Gantt Chart:



2 3. Calculations:

Proc	Arr	Burst	Finish	TAT (Fin-Arr)	WT (TAT-Burst)
P1	0	7	7	7	0
P2	2	4	12	10	6
P3	4	1	8	4	3
P4	5	4	16	11	7
Sum				**32**	**16**

Avg Turnaround Time = $32/4 = \mathbf{8}$ ms.
Avg Waiting Time = $16/4 = \mathbf{4}$ ms.

Q16. Classical Synchronization Problem:
Explain the **Dining Philosophers Problem**. Why does a deadlock occur in the naive solution? Provide a solution strategy (using Semaphores or Monitors) that prevents deadlock.

Answer: Problem: 5 philosophers sit at a circular table. There is a bowl of rice and 5 chopsticks (one between each pair). To eat, a philosopher needs 2 chopsticks (left and right). They alternate between thinking and eating.

Deadlock in Naive Solution: If every philosopher picks up their *left* chopstick simultaneously, all chopsticks are taken. Everyone waits for the *right* chopstick, which is held by the neighbor. This creates a Circular Wait $\rightarrow$ Deadlock.

Solution Strategy (Resource Hierarchy): Force an ordering on the resources (chopsticks).

- Chopsticks are numbered 1 to 5.
- Philosophers P_1 to P_4 pick up the lower-numbered chopstick first, then the higher one.
- Philosopher P_5 picks up the **higher-numbered** chopstick first (5), then the lower one (1).

This breaks the Circular Wait condition.

Q17. Contiguous Memory Allocation:
Given memory partitions of **100 KB, 500 KB, 200 KB, 300 KB, and 600 KB** (in order), how would each of the **First-Fit, Best-Fit, and Worst-Fit** algorithms place processes of **212 KB, 417 KB, 112 KB, and 426 KB** (in order)? Which algorithm makes the most efficient use of memory?

Answer: Partitions: $H_1(100), H_2(500), H_3(200), H_4(300), H_5(600)$.

1. First Fit: Allocate the first hole big enough.

- 212K $\rightarrow$ Fits in $H_2(500)$. Rem: 288K.
- 417K $\rightarrow$ Fits in $H_5(600)$. Rem: 183K.

- 112K → Fits in H_2 (remaining 288K). Rem: 176K.
- 426K → No hole large enough (Rem: 100, 176, 200, 300, 183). **WAIT**.

2. Best Fit: Allocate the smallest hole that is big enough.

- 212K → Fits in $H_4(300)$. Rem: 88K.
- 417K → Fits in $H_2(500)$. Rem: 83K.
- 112K → Fits in $H_3(200)$. Rem: 88K.
- 426K → Fits in $H_5(600)$. Rem: 174K. **ALL PLACED**.

3. Worst Fit: Allocate the largest hole available.

- 212K → Fits in $H_5(600)$. Rem: 388K.
- 417K → Fits in $H_2(500)$. Rem: 83K.
- 112K → Fits in H_5 (remaining 388K). Rem: 276K.
- 426K → No hole large enough. **WAIT**.

Conclusion: Best Fit is the most efficient here as it places all processes.

Q18. Disk Scheduling (LOOK Algorithm):
The disk head is at cylinder **53** moving towards larger cylinders. The queue contains:

$$98, 183, 37, 122, 14, 124, 65, 67$$

Calculate the total head movement using the **LOOK** algorithm. Draw the path.

Answer: Logic: LOOK is like SCAN but does not go to the disk boundary (0 or 199); it reverses direction immediately after the last request in the current direction.

Current Head: 53. Direction: Up. Requests in order: 14, 37, 65, 67, 98, 122, 124, 183.

Path: 1. Move Up: 53 → 65 → 67 → 98 → 122 → 124 → 183. 2. Reverse (Last request was 183, don't go to 199). 3. Move Down: 183 → 37 → 14.

Total Movement:

- Upward: $183 - 53 = 130$.
- Downward: $183 - 14 = 169$.
- Total: $130 + 169 = \mathbf{299}$ cylinders.

Q19. RAID Levels:
What is RAID? Explain the structure, redundancy, and performance characteristics of **RAID 0**, **RAID 1**, and **RAID 5**.

Answer: RAID (Redundant Array of Independent Disks) combines multiple physical disk drives into one logical unit for data redundancy or performance improvement.

- **RAID 0 (Striping):**
  - Data is split into blocks and written across all drives.
  - **Pros:** High performance (parallel R/W).
  - **Cons:** No redundancy. If one disk fails, all data is lost.
- **RAID 1 (Mirroring):**
  - Data is written identically to two (or more) drives.
  - **Pros:** 100% redundancy. Good read speed (can read from either).

  - **Cons:** High cost (50% storage efficiency).
- **RAID 5 (Striping with Distributed Parity):**
  - Data and parity bits are striped across 3+ disks. Parity allows reconstruction of data if one disk fails.
  - **Pros:** Good balance of performance and redundancy. Storage loss is only $1/N$ disks.
  - **Cons:** Write operations are slower due to parity calculation overhead.

Q20. File System Implementation:
(a) Discuss the **Access Matrix** model of protection.
(b) Compare **Sequential Access** and **Direct Access** methods for files.

Answer: (a) Access Matrix: A model to manage protection where:

- **Rows** represent Domains (users/processes).
- **Columns** represent Objects (files/printers).
- Each entry $Access(i, j)$ defines the set of operations (Read, Write, Execute) that Domain i can perform on Object j.

(b) Access Methods:

Sequential Access	**Direct (Random) Access**
Information is processed in order, one record after another.	Programs can jump to any record in file immediately without reading pr ous ones.
Modeled on tape drives. Good for compilers or media players.	Modeled on disk drives. Good databases.
Operations: Read Next, Write Next, Rewind.	Operations: Read block n, Write b n.