DELHI SKILL AND ENTREPRENEURSHIP UNIVERSITY

End-Semester Examination (Set 2)

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. What is a 'System Call'? Give two examples.

Answer: A System Call is a programmatic interface used by a running program to request services from the Operating System kernel (e.g., file access, process creation). Examples include `fork()` for process creation and `read()` for file input.

Q2. Differentiate between Time-Sharing and Real-Time Operating Systems.

Answer: Time-Sharing OSs allow many users to share computer resources simultaneously by rapidly switching the CPU (e.g., Unix). **Real-Time** OSs (RTOS) are designed for applications where strict time constraints on operation/response are mandatory (e.g., Missile guidance, Traffic control).

Q3. Define 'Starvation' in the context of CPU Scheduling.

Answer: Starvation is a situation where a process is perpetually denied necessary resources (like the CPU) to process its work. This often happens in priority scheduling algorithms where low-priority processes wait indefinitely while high-priority processes keep arriving.

Q4. Explain the 'Convoy Effect'.

Answer: The Convoy Effect occurs in FCFS (First-Come, First-Served) scheduling when a CPU-bound process with a long burst time holds the CPU, while several I/O-bound processes wait in the ready queue. This results in poor utilization of both I/O devices and the CPU.

Q5. What is a Process Control Block (PCB)?

Answer: A PCB is a data structure in the Operating System kernel containing information needed to manage a specific process. It includes the Process ID, Program Counter, CPU registers, process state, and memory management information.

Q6. What is the difference between Logical Address and Physical Address?

Answer: A **Logical Address** is generated by the CPU during program execution (virtual address). A **Physical Address** is the actual location in the main memory hardware. The Memory Management Unit (MMU) maps logical addresses to physical addresses.

Q7. What is the purpose of the 'Dirty Bit' in page replacement?

Answer: The Dirty Bit (or Modify Bit) is a flag in a page table entry that indicates whether a page has been modified (written to) while in memory. If set, the page must be written back to the disk before being replaced; otherwise, it can simply be discarded.

Q8. Define 'Demand Paging'.

Answer: Demand paging is a virtual memory technique where pages are loaded into memory only when they are actually required (demanded) by the executing program, rather than loading the entire program at once (pure swapping).

Q9. What is Direct Memory Access (DMA)?

Answer: DMA is a hardware feature that allows certain hardware subsystems (like disk controllers or sound cards) to access main system memory independently of the CPU. This reduces CPU overhead during large data transfers.

Q10. List four standard File Attributes.

Answer:

(a) Name (human-readable identifier)
(b) Identifier (unique tag/number in file system)
(c) Type (needed for systems supporting different types)
(d) Size (current file size in bytes/blocks)

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 Round Robin (RR) scheduling algorithm. How does the size of the Time Quantum affect performance?

Answer: Round Robin is a preemptive scheduling algorithm designed for time-sharing systems.

- Each process gets a small unit of CPU time (Time Quantum, q).
- If the process burst is $> q$, it is preempted and moved to the back of the ready queue.
- If the process burst is $< q$, it releases the CPU voluntarily.

Impact of Time Quantum (q):

- **Very Large q:** RR behaves like FCFS (First-Come, First-Served). Response time suffers.
- **Very Small q:** High context switching overhead. The CPU spends more time switching than executing processes (Processor Sharing).

Q12. Describe the Banker's Algorithm for Deadlock Avoidance.

Answer: Banker's Algorithm is used to decide whether allocating resources to a process will leave the system in a **Safe State**.

(a) **Data Structures:**
- `Available`: Vector of available resources.
- `Max`: Matrix of max demand of each process.
- `Allocation`: Matrix of current allocation.
- `Need`: Matrix where $Need[i][j] = Max[i][j] - Allocation[i][j]$.

(b) **Request Algorithm:** When a request arrives, check if $Request \leq Available$. If yes, pretend to allocate and check if the resulting state is Safe.

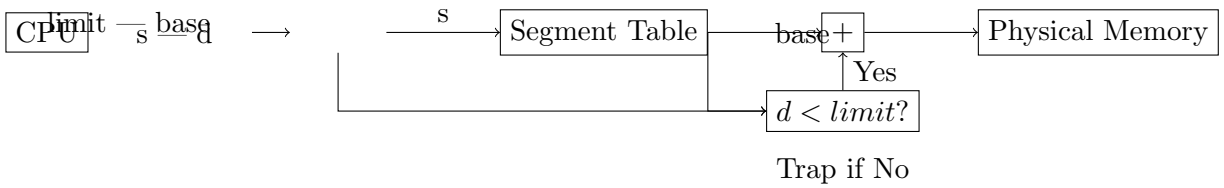
(c) **Safety Algorithm:** Try to find a sequence of processes that can finish using current available resources plus resources released by finishing processes. If such a sequence exists, the state is safe.

Q13. Explain Segmentation memory management technique with a diagram.

Answer: Segmentation supports the user view of memory. A program is a collection of segments (e.g., main, stack, symbol table). A logical address is a pair: $\langle$Segment-Number (s), Offset (d)$\rangle$.

Hardware Implementation:

- **Segment Table:** Maps segment number (s) to physical base address ($base$) and segment length ($limit$).
- The offset d must be between 0 and $limit$. If $d > limit$, a trap (error) occurs.
- Physical Address = $base + d$.



Q14. What is Belady's Anomaly? Explain with an example.

Answer: Belady's Anomaly is a phenomenon in some page replacement algorithms (specifically FIFO) where increasing the number of page frames results in **more** page faults.

- **Example:** Reference String: $1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5$.
- **3 Frames (FIFO):** 9 Page Faults.
- **4 Frames (FIFO):** 10 Page Faults.

This contradicts the intuition that more memory should lead to fewer faults. Algorithms like LRU and Optimal do not suffer from this anomaly (they are "Stack Algorithms").

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

Q15. Priority Scheduling (Preemptive) Problem:
Consider the following set of processes. Lower priority number implies higher priority.

Process	Arrival Time	Burst Time	Priority
P1	0	10	3
P2	1	1	1
P3	2	2	4
P4	3	1	5
P5	4	5	2

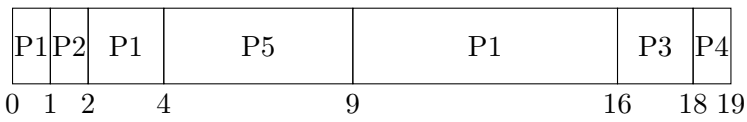
Draw the Gantt Chart and calculate **Average Turnaround Time** and **Average Waiting Time** using **Preemptive Priority Scheduling**.

Answer: Logic: At any time t, execute the process with the highest priority (lowest number) among available processes. If a new process arrives with higher priority than the current one, preempt.

Trace:

- $t = 0$: P1 (Pri 3) arrives. Starts.
- $t = 1$: P2 (Pri 1) arrives. $1 < 3$. Preempt P1 (Rem: 9). **P2 runs**.
- $t = 2$: P2 finishes (Burst 1). P3 (Pri 4) arrives. Ready: P1(Pri 3), P3(Pri 4). P1 is higher. **P1 runs**.
- $t = 3$: P4 (Pri 5) arrives. Ready: P1(Pri 3), P3(4), P4(5). P1 continues.
- $t = 4$: P5 (Pri 2) arrives. Ready: P1(Pri 3), P3(4), P4(5), P5(Pri 2). $2 < 3$. Preempt P1 (Rem: 7). **P5 runs**.
- $t = 9$: P5 finishes. Ready: P1(3), P3(4), P4(5). P1 is highest. **P1 runs**.
- $t = 16$: P1 finishes. Ready: P3(4), P4(5). P3 is highest. **P3 runs**.
- $t = 18$: P3 finishes. Only P4 remains. **P4 runs**.
- $t = 19$: P4 finishes.

Gantt Chart:



Calculations:

Proc	Arr	Burst	CT	TAT (CT-Arr)	WT (TAT-Burst)
P1	0	10	16	16	6
P2	1	1	2	1	0
P3	2	2	18	16	14
P4	3	1	19	16	15
P5	4	5	9	5	0
Total				**54**	**35**

Avg TAT $= 54/5 = 10.8$. **Avg WT** $= 35/5 = 7$.

Q16. Readers-Writers Problem:
Describe the Classical Readers-Writers problem. Provide a solution using Semaphores that gives priority to Readers (i.e., multiple readers can read simultaneously, but writers need exclusive access).

Answer: Problem: A data set is shared among concurrent processes.

- **Readers:** Only read the data; do not update.
- **Writers:** Can read and write.

Constraint: Multiple readers can read at the same time. Only one writer can access the data at a time. If a writer is writing, no other process (reader or writer) can access it.

Semaphore Solution: Variables:

- `mutex` (Semaphore, init 1): Protects `read_count`.
- `wrt` (Semaphore, init 1): Ensures exclusive write access.
- `read_count` (Integer, init 0): Counts active readers.

Pseudocode:

Writer Process

```
while(true) {
    wait(wrt);

    // Writing occurs here

    signal(wrt);
}
```

Reader Process

```
while(true) {
    wait(mutex);
    read_count++;
    if (read_count == 1)
        wait(wrt); // First reader locks writer
    signal(mutex);

    // Reading occurs here

    wait(mutex);
    read_count--;
    if (read_count == 0)
        signal(wrt); // Last reader releases
    signal(mutex);
}
```

Q17. Optimal Page Replacement:
Consider the following reference string with **3 Frames**:

$$2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2$$

Calculate the number of Page Faults using the **Optimal Page Replacement** algorithm. Explain why Optimal is not implementable in general practice.

Answer: Algorithm: Replace the page that will not be used for the longest period of time in the future.

Ref	2	3	2	1	5	2	4	5	3	2	5	2
F1	2	2	2	2	2	2	2	2	2	2	2	2
F2		3	3	3	5	5	5	5	3	3	3	3
F3				1	1	1	4	4	4	4	5	5
Fault?	Y	Y	N	Y	Y	N	Y	N	Y	N	Y	N

Step-by-step logic:

- 2, 3, 1 arrive: Faults (Frames: 2, 3, 1).
- 5 arrives: Look ahead. 2 is used soon. 5 is used soon. 3 is used later. 1 is never used again (or furthest). Replace 1 with 5. (Frames: 2, 3, 5).
- 4 arrives: Look ahead at 5, 3, 2. 2 is next, 5 is after. 3 is furthest. Replace 3 with 4. (Frames: 2, 4, 5).
- 3 arrives: Look ahead. 2 is next. 5 is next. 4 is not used. Replace 4 with 3.
- 5 arrives: In memory. Hit.

Total Page Faults: 7 (Note: Initial loading counts as faults).

Feasibility: Optimal is not implementable because it requires future knowledge of the reference string, which the OS cannot predict accurately in a real-time execution environment. It is used mainly as a benchmark.

Q18. Disk Scheduling (C-SCAN):
The disk head is at cylinder **53**. The queue of pending requests is:

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

The disk has cylinders 0 to 199. Calculate total head movement using **C-SCAN** (Circular SCAN) algorithm moving towards **higher** cylinders initially.

Answer: Logic: Head moves 53 $\rightarrow$ 199 (End), servicing requests. Then jumps immediately to 0 (Start) without servicing, and continues moving upwards.

Sorted Queue: 14, 37, 65, 67, 98, 122, 124, 183. Start: 53. Direction: Up.

Sequence:

(a) **Upward Pass:** 53 $\rightarrow$ 65 $\rightarrow$ 67 $\rightarrow$ 98 $\rightarrow$ 122 $\rightarrow$ 124 $\rightarrow$ 183 $\rightarrow$ 199 (Boundary).

(b) **Circular Jump:** 199 $\rightarrow$ 0 (No service).

(c) **Second Upward Pass:** 0 $\rightarrow$ 14 $\rightarrow$ 37.

Calculation:

- $199 - 53 = 146$
- $199 - 0 = 199$ (Jump distance)

- $37 - 0 = 37$

Total Head Movement $= 146 + 199 + 37 =$ **382** cylinders.

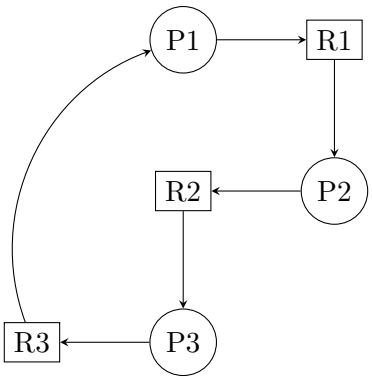
Q19. Resource Allocation Graph (RAG) & Deadlock Detection:
(a) What is a Resource Allocation Graph? Define Request Edges and Assignment Edges.
(b) Given the following state, draw the RAG and determine if there is a deadlock.

- Processes: P1, P2, P3
- Resources: R1 (1 instance), R2 (1 instance), R3 (1 instance)
- Edges: $P1 \to R1$, $R1 \to P2$, $P2 \to R2$, $R2 \to P3$, $P3 \to R3$, $R3 \to P1$.

Answer: (a) RAG: A directed graph used to model resource management.

- **Request Edge ($P_i \to R_j$):** Process P_i wants resource R_j.
- **Assignment Edge ($R_j \to P_i$):** Resource R_j is allocated to P_i.

(b) Analysis: The described edges form a cycle: $P1 \to R1 \to P2 \to R2 \to P3 \to R3 \to P1$.



Conclusion: Since there is a cycle and all resources have only single instances, a **Deadlock exists**. All three processes are waiting for each other in a circular chain.

Q20. File Allocation Methods:
Explain the **Linked Allocation** method for file systems. What are its advantages over Contiguous Allocation? How does the **File Allocation Table (FAT)** improve upon basic Linked Allocation?

Answer: Linked Allocation:

- Each file is a linked list of disk blocks.
- The directory contains a pointer to the first and last blocks.
- Each block contains a pointer to the next block.

Advantages vs Contiguous:

(a) **No External Fragmentation:** Any free block on the disk can be used to satisfy a request.
(b) **File Growth:** File size does not need to be declared in advance; it can grow dynamically as long as free blocks exist.

FAT (File Allocation Table):

- Basic linked allocation requires disk I/O to read the current block just to find the pointer to the next block (slow for random access).
- **FAT Improvement:** The pointers are stored in a separate table (FAT) at the beginning of the disk (cached in RAM). This allows the OS to traverse the chain of pointers in memory, enabling much faster random access and distinct block seeking without spinning the disk for every link.