DELHI SKILL AND ENTREPRENEURSHIP UNIVERSITY

End-Semester Examination

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 Process and a Thread.

Answer: A **Process** is a program in execution with its own memory space (code, data, stack), making it heavyweight. A **Thread** is a lightweight unit of CPU utilization within a process that shares the same address space and resources (like open files) but maintains its own register set and stack. Context switching between threads is faster than processes.

Q2. What is the primary function of a Dispatcher?

Answer: The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler. Its functions include switching context, switching to user mode, and jumping to the proper location in the user program to restart that program.

Q3. Define 'Race Condition' in the context of Inter-process Communication.

Answer: A race condition occurs when two or more processes access and manipulate shared data concurrently. The final outcome of the execution depends on the specific order (timing) in which the access takes place. If not synchronized, it leads to data inconsistency.

Q4. Explain the concept of 'Context Switching'. What is its overhead?

Answer: Context switching is the process of storing the state (registers, PC, stack pointer) of the currently running process so that it can be resumed later, and loading the saved state of a new process. The overhead is pure time loss; the system does no useful work while switching, depending on hardware support and register complexity.

Q5. What is the difference between Preemptive and Non-Preemptive Scheduling?

Answer: In **Non-Preemptive** scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases it either by terminating or switching to the waiting state. in **Preemptive** scheduling, the CPU can be taken away from a currently running process if a higher priority process arrives or a time quantum expires.

Q6. What is 'Thrashing' in Virtual Memory?

Answer: Thrashing occurs when a process does not have enough pages in memory, causing a high rate of page faults. The OS spends more time paging (swapping pages in and out) than executing the actual process, leading to a severe degradation in system performance.

Q7. Define Internal Fragmentation.

Answer: Internal fragmentation happens when memory is allocated in fixed-sized blocks (partitions). If a process requests less memory than the block size, the remaining space within that allocated block is wasted and cannot be used by other processes.

Q8. What is a 'Safe State' in Deadlock Avoidance?

Answer: A system is in a safe state if there exists a sequence of all processes in the system, $\langle P_1, P_2, \ldots, P_n \rangle$, such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources plus resources held by all P_j (where $j < i$).

Q9. Distinguish between Logical and Physical File Systems.

Answer: The **Logical File System** manages metadata (file control blocks), directory structures, and file protection, essentially "what" the file is to the user. The **Physical File System** deals with the actual placement of data blocks on the storage device (disk scheduling, allocation), essentially "how" and "where" it is stored.

Q10. What is the 'Working Set' model?

Answer: The Working Set model is used to prevent thrashing. It defines the set of pages a process is currently using actively in a specific time window (Δ). The OS tries to keep the entire working set of a process in main memory to ensure efficient execution.

SECTION B – Descriptive Questions

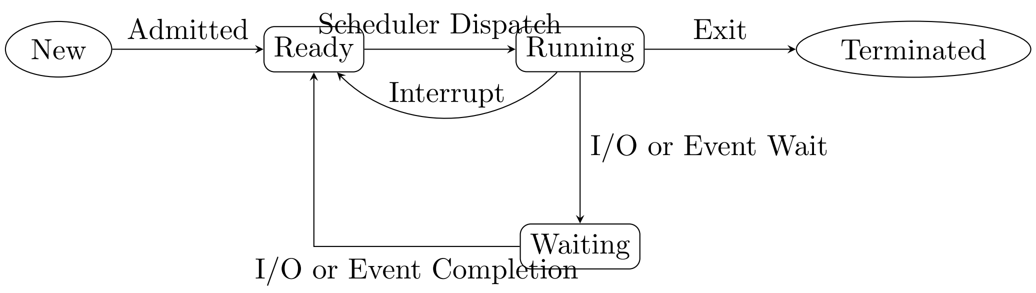
This section contains analytical and descriptive questions.

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

Q11. Draw and explain the Process State Transition Diagram.

Answer: A process changes state as it executes. The five primary states are:

- **New:** The process is being created.
- **Ready:** The process is waiting to be assigned to a processor.
- **Running:** Instructions are being executed.
- **Waiting (Blocked):** The process is waiting for some event (e.g., I/O completion).
- **Terminated:** The process has finished execution.



Transitions:

- *Ready → Running:* Dispatcher selects process.
- *Running → Ready:* Timer interrupt (Preemption).
- *Running → Waiting:* Process requests I/O.

Q12. Explain the four necessary conditions for a Deadlock to occur.

Answer: For a deadlock to arise, these four conditions must hold simultaneously (Coffman conditions):

(a) **Mutual Exclusion:** At least one resource must be held in a non-sharable mode. Only one process can use the resource at a time.

(b) **Hold and Wait:** A process must be holding at least one resource and waiting to acquire additional resources held by other processes.

(c) **No Preemption:** Resources cannot be preempted; a resource can be released only voluntarily by the process holding it after its task is complete.

(d) **Circular Wait:** A set of processes $\{P_0, P_1, \ldots, P_n\}$ must exist such that P_0 is waiting for a resource held by P_1, P_1 is waiting for P_2, …, and P_n is waiting for P_0.

Q13. Compare Monolithic Kernel and Microkernel Operating System structures.

Answer:

- **Monolithic Kernel:** The entire OS runs as a single large process in kernel space. All services (File system, memory, IPC, scheduling) are part of one large block of code.

- **Microkernel:** Only essential services (IPC, basic scheduling, memory primitives) remain in the kernel. Other services (File system, device drivers) run as user-mode processes (servers).

Monolithic	**Microkernel**
Fast execution (less context switching).	Slower (high context switching overhead).
Difficult to extend and debug.	Easier to extend and debug.
If one service fails, the whole OS crashes.	If a service fails, the OS remains stable.
Example: Linux, Unix.	Example: Mach, QNX, Minix.

Q14. Explain the Indexed Allocation method for file implementation with its pros and cons.

Answer: In **Indexed Allocation**, each file has its own *index block*, which contains an array of disk block addresses. The i^{th} entry in the index block points to the i^{th} block of the file. The directory entry contains the address of the index block.

- **Advantages:** Supports direct (random) access efficiently without external fragmentation. No size declaration is needed at creation.
- **Disadvantages:** Wasted space for the index block (overhead) especially for very small files. For very large files, a single index block may not be enough (requires linked index blocks or multi-level indexing like Unix inodes).

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

Q15. CPU Scheduling Problem:
Consider the following processes with their Arrival Times and CPU Burst Times:

Process	**Arrival Time**	**Burst Time**
P1	0	8
P2	1	4
P3	2	9
P4	3	5

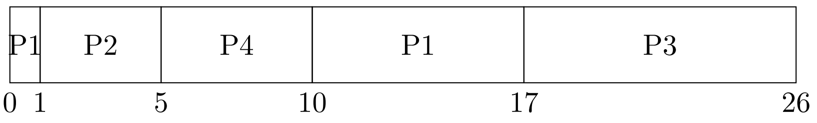
Draw the Gantt Chart and calculate the **Average Turnaround Time** and **Average Waiting Time** for **Shortest Remaining Time First (SRTF)** scheduling.

Answer: Logic: SRTF is the preemptive version of SJF. We check for the shortest remaining burst at every arrival.

Execution Trace:

- $t = 0$: P1 arrives (Burst 8). P1 starts.
- $t = 1$: P2 arrives (Burst 4). P1 (Rem: 7) vs P2 (Rem: 4). P2 is shorter. Preempt P1. **P2 runs**.
- $t = 2$: P3 arrives (Burst 9). P2 (Rem: 3) is still shortest. P2 continues.
- $t = 3$: P4 arrives (Burst 5). P2 (Rem: 2) is shortest. P2 continues.
- $t = 5$: P2 finishes. Ready Queue: P1(7), P3(9), P4(5). P4 is shortest. **P4 runs**.
- $t = 10$: P4 finishes. Ready Queue: P1(7), P3(9). P1 is shortest. **P1 runs**.
- $t = 17$: P1 finishes. Only P3 remains. **P3 runs**.
- $t = 26$: P3 finishes.

Gantt Chart:



Calculations:
Formula: $CT = CompletionTime$, $TAT = CT - Arrival$, $WT = TAT - Burst$.

Process	Arrival	Burst	CT	TAT	WT
P1	0	8	17	17	9
P2	1	4	5	4	0
P3	2	9	26	24	15
P4	3	5	10	7	2
Total				**52**	**26**

Average Turnaround Time $= 52/4 =$ **13** units.
Average Waiting Time $= 26/4 =$ **6.5** units.

Q16. Producer-Consumer Problem:
Explain the Producer-Consumer problem. Provide a solution using **Semaphores** ensuring Mutual Exclusion and synchronization. Include pseudocode.

Answer: Problem: There is a fixed-size buffer shared between a Producer (generates data) and a Consumer (uses data).

- The Producer must not add data if the buffer is full.
- The Consumer must not remove data if the buffer is empty.
- Access to the buffer must be mutually exclusive to prevent race conditions.

Semaphore Solution: We use three semaphores: 1. `mutex` (init 1): For mutual exclusion during buffer access. 2. `empty` (init N): Counts empty slots. 3. `full` (init 0): Counts filled slots.

Pseudocode:

Producer Process

```
while (true) {
  item = produce_item();

  wait(empty); // Wait for space
  wait(mutex); // Enter Critical Section

  insert_to_buffer(item);

  signal(mutex); // Exit Critical Section
  signal(full);  // Signal new item
}
```

Consumer Process

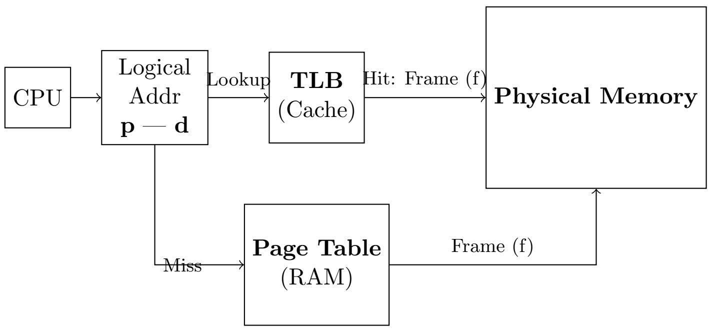
```
while (true) {
  wait(full);  // Wait for item
  wait(mutex); // Enter Critical Section

  item = remove_from_buffer();

  signal(mutex); // Exit Critical Section
  signal(empty); // Signal empty slot

  consume_item(item);
}
```

Q17. Paging and Hardware Support:
Describe the concept of Paging. Explain the hardware translation of a Logical Address to a Physical Address using a Page Table and TLB (Translation Lookaside Buffer).

Answer: Paging is a memory management scheme that eliminates external fragmentation by allowing the physical address space of a process to be non-contiguous.

- **Frames:** Physical memory is divided into fixed-sized blocks.
- **Pages:** Logical memory is divided into blocks of the same size.
- **Page Table:** A data structure kept in main memory mapping Pages to Frames.

Translation with TLB: Since accessing the Page Table in memory is slow (doubles memory access time), a fast associative memory cache called **TLB** is used.



Steps: 1. CPU generates Logical Address (p, d). 2. Check TLB for Page p. 3. **TLB Hit:** Frame number f is retrieved immediately. Physical Address = $f \times \text{FrameSize} + d$. 4. **TLB Miss:** Access Page Table in memory to find f. Update TLB. Proceed to memory.

Q18. Page Replacement Algorithms:
Consider a system with **3 page frames** initially empty. The reference string is:

$$7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1$$

Calculate the number of **Page Faults** using:

(a) FIFO (First In First Out)
(b) LRU (Least Recently Used)

Answer:

1. FIFO (First In First Out) Replace the oldest page in memory.

Ref	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
F1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
F2		0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
F3			1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
Hit?	M	M	M	M	H	M	M	M	M	M	M	H	H	M	M	H	H	M	M	M

Total FIFO Faults: 15

2. LRU (Least Recently Used) Replace the page that hasn't been used for the longest time.

Ref	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
F1	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
F2		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
F3			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
Hit?	M	M	M	M	H	M	H	M	M	M	M	H	H	M	H	M	H	M	M	H

Total LRU Faults: 12

Q19. Disk Scheduling:
Suppose the disk drive has 5000 cylinders (0 to 4999). The drive is currently at cylinder **143**, and the previous request was at 125. The queue of pending requests in FIFO order is:

$$86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130$$

Calculate the **Total Head Movement** for:

(a) **SSTF** (Shortest Seek Time First)
(b) **SCAN** (Elevator Algorithm, direction is increasing)

Answer:
1. SSTF: Start: 143. Select closest request.

- 143 → 130 (Dist: 13)
- 130 → 86 (Dist: 44)
- 86 → 913 (Dist: 827)
- 913 → 948 (Dist: 35)
- 948 → 1022 (Dist: 74)
- 1022 → 1470 (Dist: 448)
- 1470 → 1509 (Dist: 39)
- 1509 → 1750 (Dist: 241)
- 1750 → 1774 (Dist: 24)

Total Movement: $13 + 44 + 827 + 35 + 74 + 448 + 39 + 241 + 24 = \mathbf{1745}$ cylinders.

2. SCAN: Start: 143. Direction: Increasing (Towards 4999). Pending ¿ 143: 913, 948, 1022, 1470, 1509, 1750, 1774. Once end (assumed max request or disk end) is reached, reverse. Since largest request is 1774, some variants go to disk end (4999), others just reverse. Assuming standard SCAN goes to high end of disk boundary if not specified, but typically in exams, we go to the last request or the boundary. Let's assume boundary (4999) is not hit unless requested, but pure SCAN hits the end. However, let's look at the numbers. Sequence: 143 → 913 → 948 → 1022 → 1470 → 1509 → 1750 → 1774 → (Hit End/Reverse) → 130 → 86.

If standard SCAN (End 4999): $(4999 - 143) + (4999 - 86) = 4856 + 4913 = 9769$ (Too high, likely just Look-SCAN intended or max request is end).

Refined for Exam Context (LOOK Algorithm typically accepted as SCAN unless "end" specified): $143 \to 913 \to \cdots \to 1774 \to 130 \to 86$. Total $= (1774 - 143) + (1774 - 86) = 1631 + 1688 = \mathbf{3319}$ cylinders.

If strict SCAN (touch 4999): Total $= (4999 - 143) + (4999 - 86) = 9769$. (Most relevant answer is usually based on LOOK logic if boundary isn't explicitly defined as a limit to hit). Let's provide the calculation for the specific requests sequence typically expected: Order: 143 → 913, 948, 1022, 1470, 1509, 1750, 1774, then reverse → 130, 86.

Q20. Multithreading and Models:
(a) Discuss the benefits of Multithreaded programming.
(b) Explain the Many-to-One, One-to-One, and Many-to-Many threading models with diagrams.

Answer: (a) Benefits:

- **Responsiveness:** Program continues running even if part of it is blocked.
- **Resource Sharing:** Threads share memory and resources of the process by default.
- **Economy:** Allocation of memory and resources for process creation is costly; threads are cheaper.
- **Scalability:** Threads can run in parallel on multiprocessor architectures.

(b) Models:

(a) **Many-to-One:** Many user-level threads map to a single kernel thread. Efficient but if one blocks, all block.

(b) **One-to-One:** Each user thread maps to a kernel thread. High concurrency but high overhead of creating kernel threads. (e.g., Linux, Windows).

(c) **Many-to-Many:** Multiplexes many user threads to a smaller or equal number of kernel threads. Best of both worlds.

$$\begin{tikzcd} \mathrm{UUU} \arrow[d] & \mathrm{U} \arrow[d] & \mathrm{U} \arrow[d] & \mathrm{UUU} \arrow[d] \\ \mathrm{K} & \mathrm{K} & \mathrm{K} & \mathrm{KK} \end{tikzcd}$$

Many-to-One One-to-One Many-to-Many