Assignment - 3

PART-A

Ans1. A race condition occurs when two or more processes access a shared resource simultaneously and the outcome depends on the order of access, leading to unpredictable results. For a real world example, consider two people trying to write on the same whiteboard at the same time. If both write simultaneously without coordination, the writings may overlap and become unreadable. Mutual exclusion addresses race conditions by ensuring that only one person can write on the white board at a time, preventing simultaneous access and keeping the content consistent.

Ans 2. Peterson's solution is a software-based mutual exclusion algorithm that requires only two processes and uses shared variables and busy waiting; it is simple but limited in scope and sensitive to hardware memory label. Semaphores are more complex synchronization primitives that can be used for multiple process, involve atomic hardware operations for wait and signal, and support blocking, making them more versatile and efficient but also hardware-dependent.

Ans3 An advantage of using monitors in a multi-core system is that monitors provide higher level synchronization abstraction built into the programming language, which can automatically handle lock management and condition variables efficiently, reducing programming complexity and potential errors compared to lower level semaphore primitives.

Ans4 Starvation in the reader-writer problem can occur when either readers or writers are perpetually delayed because the other group is continuously given preference. For example, if readers keep coming, a writer might starve waiting for access. One method to prevent starvation is to use a fair scheduling policy like writer-preference or a queue that ensures that writers get access in a timely manner, preventing indefinite postponement.

Ans5. Eliminating the Hold and Wait condition usually requires a process to request all resources at once before execution. A practical drawback is that it can lead to low resource utilization and process starvation, as processes may hold resources longer than needed or be delayed until all requested resources become unavailable, reducing system throughput.

Part - B

(a) ~~Calculate~~ Banker's Algorithm Simulation:

Calculate the need matrix (Need = Max - Allocation):

Process	Need (A, B, C)
P0	7-0=7, 5-1=4, 3-0=3
P1	3-2=1, 2-0=2, 2-0=2
P2	9-3=6, 0-0=0, 2-2=0
P3	4-2=2, 2-1=1, 2-1=1
P4	5-0=5, 3-0=3, 3-2=1

(b) Available resource calculation
Total Resource $A = 10; B = 5, C = 7$

Allocated resources sum:

$A = 0+2+3+2+0 = 7$

$B = 1+0+0+1+0 = 2$

$C = 0+0+2+1+2 = 5$

Available: $A = 10-7 = 3$

$B = 5-2 = 3$

$C = 7-5 = 2$

(c) Safety check
Start with available = (3,3,2)
P1 need (1,2,2) ≤ (3,3,2) → Yes
Allocate resources to P1, then release P1's allocated resources to Available.

Available = Available + Allocation P1 = (3,3,2) + (2,0,0)
= (5,3,2)

Next, find next process with Need ≤ available
P3 Need (2,1,1) ≤ (5,3,2) → Yes
Available = (5,3,2) + (2,1,1) = (7,4,3)

- Next

P0 need (7,4,3) ≤ (7,4,3) → Yes
Available = (7,4,3) + (0,1,0) = (7,5,3)

- Next

P4 Need (5,3,1) ≤ (7,5,3) → Yes
Available = (7,5,3) + (0,0,2) = (7,5,5)

- Next.

P2 Need (6,0,0) ≤ (7,5,5) → Yes
All processes can finish; the system is in a safe state.

_/_/_

d. Check request of P1 for (1,0,2):
- Check if request ≤ Available.

Request (1,0,2) ≤ Available (3,3,2) → Yes.
(available C is 2 matches request C 2)
- Pretend to allocate and check safety.

Available after allocation = (3,3,2) − (1,0,2) = (2,3,0)

P1 allocation after request = (2,0,0) + (1,0,2) = (3,0,2)

update need for P1 = (1,2,2) − (1,0,2) = (0,2,0)
- Run safety algorithm with new Available (2,3,0).

7. Dining philosophers problem.

Deadlock scenario: Each of the five philosophers picks up forks simultaneously to 4(N−1), ensuring always atleast one philosophers can eat.

Another is to enforce an ordering rule for picking forks (resource hierarchy)

8. I/O system analysis
CPU time spent handling interrupts per second

interrupt rate = Data transfer rate / Data block size per interrupts
CPU time per second = interrupt/sec × interrupt handling time

= ~~5000 × 5~~ 500 KB/s / 0.1 KB (100 Bytes) = 5000 interrupts/sec

5000 × 5 μs = 25,000 μs = 25 ms

9. Air Traffic control system Case study.

(a) Critical sections needing mutual exclusion:- Shared radar data structures accessed by multiple processes.

Proposed IPC mechanism
Use message queues or shared memory with priority-based locking for real time low latency communication.

b. Deadlock detection and recovery strategy

- Detect deadlock using resource allocation graph cycle detection.
- Recover by preempting resources or terminating the lowest-priority process to minimize disruption.

x ——— x ——— x ——— x ——— x