Name: Monika Raghav
Roll No.: 23014 20010
Course: B.Tech (CSE) D.S. Date.....

Assignment - 4 (Theory)

Short Answer type : Part A

Ans1) Race condition (Real-world Example)
A race condition happens when two or more people try to do the same task at the same time, and the final result becomes incorrect because their actions.

Eg: • A piggy bank has ₹ 500.
• Person A and Person B both check it at the same time and see ₹ 500.
• Each decided to take out ₹ 300.
• Both withdraw, and the piggy bank ends up giving ₹ 600, which is incorrect.

This error happens because they acted simultaneously.

Ans2) How mutual Exclusion fixes it.
Mutual exclusion (mutex) means only one person is allowed to access the shared resource at a time using the same piggy bank example:
• Person A takes a key (lock) to the piggy bank
• Person B must wait until A finishes.
• A checks ₹ 500 → withdraw ₹ 300 → update the amount → returns the key.
• Then B checks the new amount (₹ 200) and realizes they cannot withdraw ₹ 300.

Thus, the final result is correct.

Date.....

Ans 2) Comparison : Peterson's Solution Vs Semaphores

Aspect	Peterson's Solution	Semaphores
Implementation Complexity	Simple in concept but hard to implement correctly; works only for two processes; requires careful ordering of statements	Easier to use in practice; supports multiple processes; but misuse (like forgetting signal()) can cause deadlocks.
Hardware Dependency	Purely Software-based and requires no special hardware instructions.	Often relies on atomic hardware instructions (like test-and-set, swap) to ensure atomicity.
Scalability	Does not scale well beyond two processes.	Highly scalable for many processes.
Practical use	Mostly theoretical, rarely used in real systems.	Widely used in operating systems and real applications.

Peterson's Solution is simple but limited, purely software-based, and mainly theoretical. Semaphores are more practical, scalable, and commonly used, but depend on hardware atomic operations.

Ans 3) Advantages of Using Monitors in a Multi-Core system

Monitors provide built-in mutual exclusion, so only one thread can access shared data at a time

Spiral

Date.....

without manually managing locks. This makes synchronization safer and less error-prone than semaphores. In a multi-core system with many threads running simultaneously, this automatic control greatly simplifies coordination and prevents common mistakes like forgetting to release a semaphore.

Ans4) Starvation in the Reader-Writer Problem
Starvation occurs when one group (readers or writers) keeps getting delayed forever. For ex: in a readers-priority system, continuous arrival of readers may keep allowing readers to enter, while a waiting writer never gets a chance to write. This leads to writer starvation.

Method to prevent Starvation
Fair Scheduling (FIFO ordering):
Use a queue to serve readers and writers in the exact order they arrive.
This ensures that once a writer is waiting no new readers can skip ahead, and every process eventually gets its turn.

Ans5) Drawback of Eliminating "Hold and Wait".
If a process must request all resources at once before running, it may end up holding resources it won't use until much later. This leads to Resource underutilization and longer waiting for other processes.

Spiral

Date.....

Example: A process needs a printer only at the end of its execution, but because of this rule, it must request the printer at the beginning. While it runs, the printer stays idle and locked, preventing other processes from printing - even though this process won't use the printer until much later.

Part - B : Application / Numerical Based

Ans 6) Banker's Algorithm Simulation:

Given: Total resources: A=10, B=5, C=7

Allocation and Max:

Proc	Allocation (A, B, C)	Max (A, B, C)
P_0	(0, 1, 0)	(7, 5, 3)
P_1	(2, 0, 0)	(3, 2, 2)
P_2	(3, 0, 2)	(9, 0, 2)
P_3	(2, 1, 1)	(4, 2, 2)
P_4	(0, 0, 2)	(5, 3, 3)

(a) Need matrix = Max - Allocation

Proc	Need (A, B, C)
P_0	(7, 4, 3)
P_1	(1, 2, 2)
P_2	(6, 0, 0)
P_3	(2, 1, 1)
P_4	(5, 3, 1)

(b) Is the system in a safe state?

Sol. Compute Available = Total - Σ(Allocation)

Σ Allocation = (7, 2, 5)

Available = (10, 5, 7) - (7, 2, 5) = (3, 3, 2)

Find a safe sequence using Banker's algorithm:

Date .....

1) Available = (3,3,2) → P_1 (need (1,2,2)) can run.
After P_1 finishes, Available ← (3,3,2) + Alloc(P_1) = (5,3,2)
2) Available = (5,3,2) → P_3 (need (2,1,1)) can run.
After P_3, Available = (7,4,3)
3) Available = (7,4,3) → P_0 (need (7,4,3)) can run.
After P_0, Available = (7,5,3)
4) Available = (7,5,3) → P_2 (need (6,0,0)) can run.
After P_2, Available = (10,5,5)
5) Available = (10,5,5) → P_4 (need (5,3,1)) can run.
A safe sequence exists: $P_1 \rightarrow P_3 \rightarrow P_0 \rightarrow P_2 \rightarrow P_4$
So the System is in a safe state.

(c) Request by P_1: (1,0,2) - can it be granted immediately?
Sol: Check conditions: (1) Request ≤ Need (P_1)? Need(P_1) = (1,2,2)
Request (1,0,2) ≤ (1,2,2) → Yes
(2) Request ≤ Available? Available = (3,3,2)
(1,0,2) ≤ (3,3,2) → Yes
Tentively grant and test safety (temporary update):
- New Available = (3,3,2) - (1,0,2) = (2,3,0)
- New Alloc(P_1) = (2,0,0) + (1,0,2) = (3,0,2)
- New need(P_1) = (1,2,2) - (1,0,2) = (0,2,0)

Run safety check with updated values:
- Available (2,3,0) → P_1 (need (0,2,0)) can finish → Available becomes (5,3,2)
- Then as in (b), $P_3 \rightarrow P_0 \rightarrow P_2 \rightarrow P_4$ can finish in seqn.

Since a safe sequence exists after granting the request, the request can be granted immediately.

Ans) Dining Philsophers Problem - 5 Philosophers
- Philosophers: P_0, P_1, P_2, P_3, P_4

Shinal

Date.....

- chopsticks (shared resources) b/w each philosopher: C_0, C_1, C_2, C_3, C_4
- Each philosopher needs two chopsticks to eat (left & right)
- Semaphores for chopsticks: sem [C_0 ... C_4] = 1 (binary semaphore).

(1) Semaphore solution (naive) :-

```
# Pseudo code for each philosopher
while (true) {
    wait (left-chopstick);        // acquire left
    wait (right-chopstick);       // acquire right
    eat ();
    signal (left-chopstick);      // release left
    signal (right-chopstick);     // release right
```

(2) Step-by-step Execution leading to Deadlock

Assume all philosophers pick left chopstick first:

Step	Philosopher action	Status of Chopsticks
1	P_0 picks C_0	$C_0 = 0$, others = 1
2	P_1 picks C_1	$C_1 = 0$, others = 1
3	P_2 picks C_2	$C_2 = 0$, others = 1
4	P_3 picks C_3	$C_3 = 0$, others = 1
5	P_4 picks C_4	$C_4 = 0$, others = 1
6	P_0 tries C_1	blocked ($C_1 = 0$)
7	P_1 tries C_2	blocked ($C_2 = 0$)
8	P_2 tries C_3	blocked ($C_3 = 0$)
9	P_3 tries C_4	blocked ($C_4 = 0$)
10	P_4 tries C_0	blocked ($C_0 = 0$)

Deadlock occurs: All philosophers hold one chopstick and wait forever for the others.

(3) Avoiding Deadlock: Resource Hierarchy Solution

Spiral

Date
Page No.

- Assign a numbering to chipsticks: $C_0 = 0$, $C_1 = 1$, ... $C_4 = 4$
- Philosophers always pick the lower-numbered chopstick first, then higher-numbered.

Pseudo code modified:

```
left = min(my_chopstick);
right = max(my_chopstick);
wait(left);
wait(right);
eat();
signal(left);
signal(right);
```

Step-by-step safe Execution Example

- P_0 wants C_0 and C_1 → picks C_0 first, then C_4
- P_4 wants C_4 and C_0 → picks C_0 first, then C_4

Because everyone picks in order of chopstick no.:

- No circular waiting can occur → Deadlock prevented.
- Philosophers eat one by one or in parallel safely.

Ans 8) I/O System Analysis:

Given:
- Interrupt-driven I/O
- Interrupt handling time = $5\mu s = 5 \times 10^{-6}$ s
- Device transfer rate = 500 KB/s = 500×1024 B/s $\approx 512,000$ B/s
- Data block per interrupt = 100 B

(a) CPU time spent handling interrupts per second

Step 1: Calculate number of interrupts per second

$$\text{Interrupts/sec} = \frac{\text{Data rate}}{\text{Data per interrupt}} = \frac{512000}{100}$$

$\Rightarrow 5120$ interrupts/sec.

Step 2: CPU time per second spent handling interrupts

CPU time = Interrupts/sec × Interrupt handling time.

Date
Page No.

CPU time = 5120 × 5 × 10^{-6} s
= 0.0256 s ⇒ 25.6 ms/sec.

(b) Improvement to reduce CPU overhead
Method: Use larger data blocks per interrupt or DMA (Direct Memory Access).
- Increasing block size per interrupt reduces the no. of interrupts, hence CPU spends less time handling them.
- Using DMA allows the device to transfer data directly to memory without CPU intervention, further reducing CPU overhead while maintaining the same transfer rate.

Ans 9) Air Traffic control System case study
(a) Critical sections in the ATC system.
- Radar data updates: Shared radar database must be updated by only one process at a time.
- Fight path calculations: Updates to flight paths must be mutually exclusive to prevent conflicting routes.
- Pilot communication buffer: Outgoing messages must be written safely without overlap.

IPC Mechanism:
Use Shared memory with semaphores or mutex locks, as they offer very fast access and are suitable for real-time systems. For process-to process messaging, priority-based message queues can also be used to deliver urgent updates first.

(b) Deadlock Detection and Recovery

Detection:

Use a resource allocation graph and periodically check for cycles. A cycle indicates a deadlock b/w radar acquisition and flight-path processes.

Recovery:

Apply preemption or rollback:

- Temporarily force the lower-priority process to release resources, or.
- Roll back one process to its previous safe state and restart it.

This ensures minimal disruption while high-priority operations (like radar data updates) running smoothly.