4.) Reader - writer Starvation and Prevention

Ans. Starvation - Writers may starve if readers continuously access the resource.

Prevention :- Use a writer - priority protocol or queue based scheduling to ensure writers eventually get access.

6.) Distributed Deadlock Detection Simulation

Ans. Given: $S_1 : P_1 \rightarrow P_2$, $P_3 \rightarrow P_4$
$S_2 : P_2 \rightarrow P_5$, $P_5 \rightarrow P_6$
$S_3 : P_6 \rightarrow P_1$

a) Global Wait - for Graph:
$P_1 \rightarrow P_2 \rightarrow P_5 \rightarrow P_6 \rightarrow P_1$
$P_3 \rightarrow P_4$

b) Deadlock Detection :

Cycle exists : $P_1 \rightarrow P_2 \rightarrow P_5 \rightarrow P_6 \rightarrow P_1$
Deadlock Processes : P_1 , P_2 , P_5 , P_6

Date
Page No.

c) Distributed Algorithm :
Chandy - Misra - Haas Algorithm for distributed deadlock detection.

7) Distributed File System Performance

Given : local access : 5ms
remote access : 25ms
remote probability : 0.3

a) Expected file access time E[T] :

E[T] = (0.7 x 5) + (0.3 x 25)
= 3.5 + 7.5 = 11ms

b) Caching Strategy :

→ Client - side catching - store frequently accessed remote file locally.

→ Justification - reduces repeated remote access latency and network load.

8) Checkpointing in a Concurrent System

Given : Full : 200ms
Incremental : 50 ms
RPO : 1s

Date
Page No.

a) Optimal Mix :- perform 1 full checkpoint every 18, followed by incremental checkpoints every 250 ms.

b) Reasoning : Incremental checkpoints are faster, reducing overhead. Full checkpoints ensure complete recovery. Combination meets RPO without blocking the system.

9) Case-study - Global E-Commerce Platform

Ans. a) Distributed Scheduling Challenges :
Flash sales create sudden load spikes, uneven across regions.

Suitable Algorithm : Weighted Round Robin or Dynamic load Balancing using least-loaded Server.

b) Fault Tolerance Strategy :-
Geo-Redundant Deployment - replicate services across multiple data centres.

RTO/RPO :- Use synchronous replication for critical services (low RPO) and asynchronous replication for less critical services. (Acceptable RTO)

Result : High availability even if a region fails.

Note - ~~1,2,3,5 ques~~

Ans1.) A race condition occur when two people or processes try to access or update a shared resource at the same time and the result depends on who reaches it first. The outcome becomes unpredictable.

Ex- Two roomates try to write on the same whiteboard shopping list. One erases an item thinking it's bought while the other is adding the same item. The final list become inconsistent.

How mutual exclusion solves it :-

Using mutual exclusion means only one person can access the shared resource at a time.

Eg- Keeping a "key" or token to the whiteboard ensures one person completes their update before the next person start. This removes inconsistency and ensures correct results.

Ans2.

Peterson's Solution	Semaphores
• It is a pure software based solution for achieving mutual exclusion.	• work for multiple processes/ threads and are widely used in operating systems.

2. Does not require any special hardware support.	2 Require atomic hardware instructions
3 Work only for two processes, making it less scalable.	3. Easier for programmer to use and implement.
4. Hard to rely on in modern multi-core system due to compiler optimization and memory reordering issues.	4. More reliable in real operating system.

Overall, Semaphores are more practical and scalable, while Peterson's solution is mostly theoretical.

Ans 3. Advantage of using Monitors in Multi Core system.

Monitors provide automatic locking when a thread enters a monitor procedure and automatically release the lock when it exits. This reduces programmer responsibility for manually locking and unlocking shared data.

In multi-core systems where many threads run in parallel, monitors:

→ Reduce synchronization mistakes
→ Keep shared variables protected automatically.
→ Allow cleaner, object-oriented structure for concurrent code.

This, monitors make concurrent programming safer and easier on multi-core platforms.

Ans 5) Drawback of Eliminating "Hold and wait" condition.

To prevent deadlock, one rule is to force processes to request all resources at once, before starting execution, while this prevents hold and wait, it has a practical drawback:

→ Resources get reserved even when the process doesn't need them immediately.

→ This leads to low resource utilization, where many resources sit idle.

→ It increases waiting time for other processes and reduces overall system efficiency.

DELTA Notebook

Thus, while safe, it makes the OS resource management less efficient.