Part A

Explain race conditions with a real-world example outside computing and how mutual exclusion fixes it.

A race condition happens when multiple entities try to use a shared resource at the same time, causing inconsistent results.

For example, if two people try to withdraw money from the same bank account simultaneously without coordination, both may check the balance before it updates, leading to incorrect withdrawals.

Mutual exclusion prevents this by ensuring only one person accesses the shared resource at a time.

2. Compare Peterson's solution and semaphores in terms of complexity and hardware dependency.

Peterson's solution is logically complex and depends on strict memory ordering, which may fail on modern processors. Semaphores are simpler to implement and rely on hardware-supported atomic operations, making them practical and reliable in real systems.

3. Advantage of using monitors for the producer-consumer problem in a multi-core

system.
Monitors automatically handle synchronization by combining shared data and mutual exclusion. This reduces programming errors and ensures safe access to shared resources in multi-core environments.

How starvation occurs in the Reader-Writer problem and how to prevent it.
Starvation can occur when continuous readers prevent waiting writers from getting access, meaning a writer may wait indefinitely. It can be prevented using fair scheduling or writer-priority schemes where waiting writers eventually get access.

5. Practical drawback of eliminating the "Hold and Wait" condition in deadlock prevention.**
If processes must request all resources at once, many allocated resources may stay unused for long periods. This leads to poor resource utilization and inefficiency in the system.

Part B

6. Distributed deadlock detection simulation

*Combined global wait-for graph:
P1 ⟶ P2 ⟶ P5 ⟶ P6 ⟶ P1 and separately P3 ⟶ P4
A deadlock exists because the cycle P1 ⟶ P2 ⟶ P5 ⟶ P6 ⟶ P1 forms a closed loop.
*A suitable distributed detection algorithm is the Chandy–Misra–Haas algorithm.

7. Distributed file system performance
Expected access time:
((0.7 \times 5ms) + (0.3 \times 25ms) = 11ms)

*To improve performance, client-side caching with an LRU policy can be used, as it keeps frequently accessed files local and reduces remote access delays.

8. Optimal checkpoint approach

To maintain a 1-second RPO while minimizing cost, one full checkpoint can be taken initially, followed by incremental checkpoints every second for the rest of the 10-second period. This provides fast recovery while reducing system overhead.

9. Case study: global e-commerce platform

*Managing flash sales is challenging because of sudden spikes in load. Dynamic scheduling with a load-balancing algorithm like Least-Loaded or Round Robin with real-time monitoring helps distribute workload efficiently.

For fault tolerance, geo-redundant replication with automatic failover ensures services remain available during regional outages. Continuous replication supports low RPO, and automated failover allows a low RTO.