2201010218
Yadav Amit

Operating System Assignment - 4

Q6. Distributed deadlock Detection Simulation :
Ans Fragments :

- $S_1 = P_1 \longrightarrow P_2 \ , \ P_3 \longrightarrow P_4$
- $S_2 = P_2 \longrightarrow P_5 \ , \ P_5 \longrightarrow P_6$
- $S_3 = P_8 \longrightarrow P_1$

a. Global wait-for graph (combined)
$P_1 \rightarrow P_2 \longrightarrow P_5 \longrightarrow P_6 \longrightarrow P_1$ (cycle). Also $P_3 \longrightarrow P_4$ (separate)

b. Deadlock ?
→ Yes. Processes involved in deadlock cycle : P_1, P_2, P_5, P_6.

c. Suggested distributed algorithm :
→ Use the chandy-Misra-Has edge-chasing (Probe) algorithm for distributed deadlock detection - each side sends probes along wait-for edges to detect cycles without centralized graph assembly.

Q7. Distributed file system Performance :
Given : local = 5ms , remote = 25ms , prob(remote) = 0.3

a Expected access time :
$E = 0.7 \times 5 \text{ ms} + 0.3 \times 25 \text{ ms} = 3.5 + 7.5 = 11 \text{ ms}$

b. Caching Strategy :
Client-side read cache with LRU + TTL - based validation
→ Justification : Frequently-read remote files will be served locally reducing remote access (0.3 fraction), LRU evicts less used items. TTL keeps staleness bounded. Improves average latency while keeping consistency manageable.

classmate
Date
Page

Q1. Race Conditi (real-world example) and mutual exclusion.

Ans Two people (A and B) share a single car key in an office. Both check the key box at the same time, see the key present, and both take it - resulting in no key for either when they try to leave.
This is a race condition: the outcome depends on timing of accesses.
Mutual Exclusion Solution: make the key box accessible to one person at a time (eg. a lock). Only one person can take the key after acquiring exclusive access; others wait. This prevents conflicting.

Q2. Peterson's solution vs semaphores (implementation complexity & hardware dependency).

Ans Peterson's Solution: simple algorithm for two processes using shared flags and a turn variable.

Semaphores: higher-level OS primitive with blocking, wakeup and queue management. More complex to implement but less hardware-dependent.

Q3. Producer-Consumer: advantage of monitors on multi-core systems.

Ans Monitors provide automatic mutual exclusion and condition variables with blocking (no busy-waiting). On multi-core systems this reduces CPU waste (no spinning) and improves scalability by letting the OS scheduler park threads, reducing contention and cache coherence traffic compared with spin-based semaphore solutions.

Qu. Reader - Writer starvation and one prevention method

Ans. Starvation Scenario: If writers are rare but continuous streams of new readers keep arriving, writer can be perpetually delayed (reader preference implementations starve writers).

Prevention: Use writer preference or a fair queue (eg. queue incoming requests and access in FIFO order) or use a ticket/sequence number system so writers wait only a bounded time. This ensures fairness and prevents indefinite postponement of writers.

Q5. Eliminating "Hold and Wait" - practical drawback.

Ans. If OS forces processes to acquire all needed resources at start (to eliminate hold-and-wait), processes must request all resources up-front or release held resources before requesting more. Practical drawback: very low resource utilization and poor concurrency - processes hold many unused resources while waiting for others, increasing blocking and reducing throughput. User programs often cannot predict all future resource needs in advance.

8. Checkpointing Mix to meet RPO = 1s

→ Given, full = 200 ms, incremental = 50 ms, RPO = 1s

a. Proposed Mix (over 10s):

- Take one full checkpoint every 10s (at t=0 in the period)
- Take incremental checkpoint every 1s (at t = 1, 2 ... 9)

Total overhead (per 10s): 1 x 200 ms + 9 x 50 ms = 200+450 = 650

b. Reasing:

- With incremental every 1s, the maximum work lost on failure ≤ 1s → meets RPO.
- Full once per 10s bound recovery time (so restoring an older full + a small number of incrementals is feasible)
- This mix minimises full checkpoint cost while keeping incremental is feasible.
- This mix minimizes full-checkpoint cost while keeping incremental frequency high enough to meet the RPO.

9. Case Study - Global E-commerce Platform:

a. Distributed scheduling challenges for flash sales:

- [illegible], sudden split in request
- Geographic distribution & latency - data locality matters for latency and inventory correctness.
- Heterogeneous nodes
- Strateful server.

→ Suggest algorithm for lead balancing ⇒ Hybrid Approach

b. Fault - Tolerance Strategy (RTO & RPO):

- Active - active multi-region deployment: service run concurrently in multiple regions to failures is seamless (RTO ≈ near-zero at service level)

- Data Strategy:

→ Critical transactional data: Use synchreanous replication within the region to guarantee consistency and low RPO; cross region replication can be asychronous but with frequent replication to keep RPO small.