Name: Aryan Sharma
Class: IVth sem, 'D'
Roll no: 2301730238

Assignment-3

3/5

Q1. Explain race condition with real world example outside of computing, and show how mutual exclusion addresses it.

Ans. A race condition occurs when the outcome of the system depends on the specific order or timing of execution of multiple threads/processes accessing a shared resource.

Example: There is only 1 ticket left for a concert (in system). Bob and Alice see that. Bob books the ticket and Alice too. Both transaction succeed and system now thinks ~~now only~~ -1 tickets are left. This results in an error. The final state (who gets the ticket first/error state) depends ~~who~~ upon who finishes updating the inventory last.

- Mutual Exclusion Solⁿ: Mutual exclusion addresses this by ensuring that only one processes/thread can access the shared resource (the ticket inventory) at any given time. For example, like putting a digital lock on ticket:

1. Alice locks the ticket system.
2. Alice books the ticket.
3. The ticket count is updated to 0.
4. Alice unlocks the ticket system.
5. Bob attempts to lock the system and sees the count is 0. He ~~locks the~~ ~~system~~ is blocked from ~~the ticket booking system~~ purchasing.

DELTA
Date / /
Pg No.

Q2.

Ans

Feature	Peterson's Solution	Semaphores
Implementation complexity	Low. Uses simple load and store instructions. Requires two simple shared variables (flag and turn)	High. Requires OS support operations (like wait() and signal() and a waiting queue to block processes.
Hardware Dependency	High. Relies on a specific CPU architecture's atomic instruction and memory model to function (ensuring reordering)	Low/Moderate: The primitive implementation relies on low-level atomic hardware instructions ((like Test and Set), but high level Semaphore itself is an OS abstraction.

Q3.

Ans The producer-consumer problem can use either semaphores or Monitors.

• Advantage of Monitor in a ~~core~~ multi-core system: Monitors provide more easier and robust synchronization because they encapsulate the shared data and the synchronization primitives (like condition variables and mutual exclusion locks. into a single language construct. This inherent structure automatically ensures mutual exclusion on the shared data (the buffer), making it less prone to programming errors (like forgetting to call wait() and signal() correctly compared to managing seperate semaphores manually. This simplicity and safety leads to better maintainability in complex multi-core environments.

Q4. Reader-Writer Starvation and Prevention

Ans How starvation occurs: In the reader-writer priority in the reader-writer solution, a constant stream of new readers can continously ~~provide~~ acquire the lock, while the lock is held by other readers. A writer's processes waiting to share the accessed will be indefinetly blocked because there is always an active reader arriving and gaining access before the writer gets its turn.

• Prevention method: Using a Wait/Turnstile Queue: Use a writer-priority or fairness solution. A common fairness method is to implement a turnstile) queue for both readers and writers. When a processes reader/writer wants to enter ~~a~~ the critical section, it must first check a second Semaphore lock. The lock only releases when the current holder ~~releases~~ finishes. By forcing all incoming requests (~~all~~ readers & writers) through a single gate. It ensures that once a writer reaches the front of the line, new readers are blocked until the writer "is" finished preventing indefinite reader preferenc

Q5. Drawbacks of eliminating "Hold and Wait"

Ans: "Hold and wait" condition for deadlock means a processes holds at least one resource while waiting to acquire additional resources currently already holded by other processes. This can be eliminated by:

1. Requesting all resources at the start (before execution begins).
2. Allowing a process to only hold a resources when it's already holding none.

• Practical drawback: Poor resource utilization and potential starvation.

↳ Poor resource utilization: If a process needs (a printer) resource only at the very ~~beginning~~ end of it's execution, requesting it at the start means the printer is held unnecessarily, for the entire execution time, wasting the resource and blocking other processes that could have used it immediately.

↳ Starvation: Requesting all resources at once increases the chance that some resource will be unavailable, preventing the process to ever starting, leading to starvation.

Part-B: Application / Numerical Based

Q6. A system has the following resources:
Total instances ⇒ A=10 B=5 , C=7

Date / /
Pg No.

Allocation & Max tables: A=10 B=5 C=7

Process	Allocation	Max	Need	Available
	A B C	A B C	A B C	A B C
P_0	0, 1, 0	753	✓743	3 3 2
P_1	2 0 0	322	✓122	5 3 2
P_2	3 0 2	902	✓600	7 4 3
P_3	2 1 1	422	✓2 1 1	7 4 5
P_4	0 0 2	533	✓5 3 1	7 5 5
[illegible]	7 2 5			10 5 7

Safe sequence:
$P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_0 \rightarrow P_2$

If P_1 requests (1,0,2) it will be immedeately granted as there is avaibility of more resources.

06. I/O system analysis:
- Interrupt Handling time (T)int : 5μs $= 5\times10^{-6}$ seconds
- Transfer rate (R) : 500 KB/s
- Data block size (B) : 100 bytes/second

(i) Calculate CPU time handling interrupts per second
Total bytes ⇒ 500 KB/s = 500 × 1024 bytes/sec = 512,000 bytes/second

(ii) Calculate no. of interrupts per second (Nint) ⇒ $\frac{\text{Total bytes}}{\text{Data block size}} \Rightarrow \frac{512{,}000 \text{ bytes/sec}}{100 \text{ bytes/sec}}$
(E)s ⇒ 5,120 interrupts per second

(iii) Calculate the time (total) CPU spent handling interrupts ⇒ 5×10^{-6} s × 5,120 int/s
⇒ 0.0256 s/second

⇒ 25.6 millisecond is the time CPU spent handling interrupt per second

(b) Improvement: Use direct memory Access (DMA):
↳ DMA allows the I/O devices to transfer large blocks of data to and from main memory without continous intervention of CPU.

↳ Prevention of CPU overhead: The CPU is interrupted only once for large block transfer (eg 4KB) page instead of once per 100-byte block. This drastically reduces the no. of interrupts per second (N_{int} thereby reducing total CPU time spent in interrupt handling.

Q9. Case Study - Air Traffic Control System:
Ans:

(a) Identify critical section and propose an IPC mechanism.
1. Critical sections (Requires mutual execution): The most critical sections are those accessing and modifying the shared, real-time state of the aircraft and airspace.
↳ Shared airspace/flight database: This is the central repository of all current flight paths, positions, altitudes and headings
Process accessing it: Flight path calculation (writes), Radar Data Acquisation (writes/updates) and Communication (display and transmission).
mutual exclusion helps in maintaining data integrity and consistency
A flight path calculation processes must not read while a radar data acquisation is halfway through an update.

↳ Resource allocation/conflict resolution table: for processes that handle runway assignment or conflict detection.

2. The most suitable IPC mechanism for real-time response: The most suitable ~~response~~ mechanism for high performance, real time data sharing in a tightly coupled system like this is a shared memory approach, synchronized using semaphores and mutexes.

↳ Shared memory because it offers the fastest inter-process communication because process read and write to the same memory region avoiding the overhead of message passing or kernel context switches for every data exchange. Speed is paramount in real-time ATC.

↳ Synchronization: Monitors or Binary Semaphores (~~Mutexes~~ Mutexes) would be used to force mutual exclusion on the shared database access to prevent race conditions.

(b) Deadlock detection and Recovery Strategy (Minimal disruption)

1. Detection strategy: Using a resource allocation Graph (RAG) to ~~avoid~~ check formation of cycles
   - ↳ Implementation: OS monitoring process maintains a graph of processes and resources, tracking which process holds and which process waits for which resource.
   - ↳ Action: When a process blocks (waits for a resource), or ~~paid~~ periodically (eg every 500 ms) the system runs to check for cycles in the RAG. A cycle indicates a deadlock.

2. Recovery strategy (Minimal disruption): Since ATC-systems are safety-critical and must maintain service, the best recovery strategy is process preemption and rollback. Steps:

- Identify victims: select one or more processes in the deadlock cycle to preempt. The victim should be chosen based on a cost factor (e.g, the process that has executed the least, one whose rollback is easiest).
- Preempt resources: Force the chosen victim process to release it's held resources (e.g. locks on data).
- Rollback: Rollback the victim process to a safe checkpoint (a previous, stable state) where it did not hold any deadlocked resources.
- Restart: The process restarts from the checkpoint and tries to reacquire resources