Assigment - 03

Ques (1) A race condition happens when two or more people (or processes) try to happen perform action at same time & the final outcome depends on who gets there first.

Because the actions overlap, the result became incorrect, uncons'istent or unexpected.

ex :- Two people (A & B) try to withdraw money from the same ATM account at the same time.
if both check the balance (₹5000) before either withdraws, both think enough money exists & both may withdraw ₹5000 (inconsistent result)

how exclusion fixes it :-

only one person can accesses the account ~~data~~ at a time (ATM session lock)
when A is withdrawing, B must wait.
This ensure orderly acess → correct balance update.

Ques 2 Peterson's solution vs Semaphores

Aspect	peterson	semaphores
Implement Complexity	pure software, harder to understand & error-prone	Simple to use wait() / signal() with clear primitives

Hardware dependency	Requires strict ordering & atomicity of load/stores → works on uniprocessor only	Requires hardware atomic instruction (Test-and-set) but work across multi-core system

Ques 3 Advantage of using monitors in multi-core systems :-

⇒ Monitors automatically provide mutual exclusion using an internal lock, so programmers don't manage semaphores manually

Advantages :- Better safety & correctness - no issues like forgetting to release semaphore avoiding race conditions in multi-core concurrency.

Ques 4 Reader - writer starvation + prevention

How starvation occurs :

In reader-priority RW, continuous arrival of readers may prevent writer from ever getting back the lock → writer starvation

In writer priority RW, queue of writers may block readers indefinitely → reader starvation

__/__/____

Prevention method -:

use fair queuing (FIFO): maintain arrival order; next request (reader or writer) is served in first-come first serve manner. Guarantees bounded waiting

Ques Drawback of removing "Hold & wait"

To prevent hold & wait, a process must request all resources at once before starting.

Drawback: large resource waste → processes hold unused resources for long periods, reducing overall system utilization & causing increased waiting

Ques Air Traffic Control System — Case Study.

a) Critical sections requiring mutual exclusion + suitable IPC

In an ATC system, the following shared data structures require strict mutual exclusion

1. Shared Radar Data Buffer
updated continuously by radar acquisition, read by flight-path calculators.

2. Global Aircraft position Table -:
Contains real-time coordinates / altitude; accessed

by multiple processes simultaneously

3. Communication Queue.

⟹ Messages sent to pilots must be consistent & non-conflicting

These sections must never be corrupted as they directly impact aircraft safety.

Suitable IPC Mechanism :-

use real-time shared memory protected by priority-inheritance mutexes or real-time message queues

- shared memory provide fast, deterministic access (important for real-time)
- priority-inheritance mutex avoids priority & ensure time-critical task (radar update) are never delayed
- message queues ensure safe, asynchronous communication for pilot messaging

__/__/__

b) Deadlock detection & recovery with minima disruption

If a deadlock occurs b/w Radar Acquisition & flight path calculation, the system cannot stop radar updates - this is safety-critica

Detection Strategy :-

- Use a resource allocation graph (RAG) or periodic deadlock detection algorithm to detect circular wait b/w process
- ATC system typically run watch-dog timers : if a process does not update within a specific interval the system suspect a deadlock.

Recovery Strategy (minimal Disruption)

1. Pre empt or rollback the non-critical process (flight path calculation)
2. Release its held locks / resources
3. Restart its held work or resume the flight -path calculation using the latest radar data snapshot.
4. Log the event for analysis.

__/__/____

This ensure

- Safety-critical radar data collection is never interrupted.
- flight-path calculation is re-run with minimal delay.
- System recovers without shutting down and losing air tracking continuity

Ques Given

A = 10, B = 5, C = 7

Process	Allocate	Max
P_0	010	753
P_1	200	322
P_2	302	902
P_3	311	422
P_4	002	533

a) Need Matrix = Max - Allocation

Process	Need
P_0	743
P_1	122
P_2	600
P_3	211
P_4	631

Total Allocated

A : 0 + 2 + 3 + 2 + 0 = 7

B : 1 + 0 + 0 + 1 + 0 = 2

C : 0 + 0 + 2 + 1 + 2 = 5

Available = Total - Allocation

A = 10 - 7 = 3

B = 5 - 2 = 3

C = 7 - 5 = 2

b) Safe sequence ⇒ $P_1 \rightarrow P_3 \rightarrow P_0 \rightarrow P_2 \rightarrow P_4$

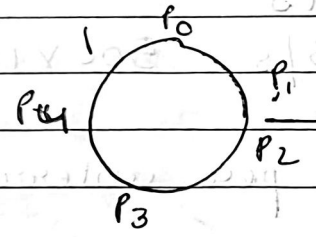
__/__/____

c) Need of P_1 - (1, 2, 2)
Request - (1, 0, 2) ≤ Need
P_1's request cannot be granted immediately

Ques 7 :- Dining Philosphere :-

5 Philosphers
P_0, P_1, P_2, P_3, P_4

Philosphere task
Eating
Thinking



Fork :- one b/w each pair.

Naive Semaphore

It leads to deadlock

```
while (true) {
    think();
    wait (fork[i]);
    wait (fork[(i+1)%5]);
    eat ();
    signal (fork[(i+1)%5]);
    signal (fork[i]);
}
```

constraints

- No eating while thinking
- No thinking while eating
- Each philosphere has to pick fork on left & right, if available.

every one picks fork on the left meaning all will hold the fork on their left & wait for fork on right causing a deadlock

deadlock free.

Asymmetric if (i%2 == 0) { wait (left); wait (right); }

if (i%

else { wait (right); wait (left); }

eat();

signal (right(); signal (left)).

Ques 8) a) Interrupt time = 5 MS

Transfer rate = 500 KB/s = 500 X 1024 = 512000 b/s

Block size = 100 bytes per interrupt

Interrupt/sect = $\frac{512000}{100} = 5120$

CPU time spent

5120 X 5MS = 25600MS = 0.0256 S

b) use DMA (Direct memory access as it trans fers large chunks of data directly to memory with for fewer interrupt reducing the CPU involvement while keeping the same transfer rate