Name- Atishay Jain
E.No - 21103285

Date....................

Operating Systems
Tutorial-5

Q1. A race condition is a programming issue where the behaviour of a program becomes unpredictable due to the unpredictable timing of multiple threads or processes accessing shared resources concurrently.

eg → Imagine two threads A + B, both trying to increment 'counter' by 1. The initial value is 0.

1. Thread A reads 'counter' into a temp variable.
2. Thread B also reads 'counter' into a temp. variable.
3. Thread A increments variable by 1.
4. Thread B also ~~a~~ increments variable by 1.
5. Thread A writes value of var. into 'counter' (1)
6. Thread B also write value into 'counter' (1).

This way final value of counter is '1' instead of 2.

Q2. The progress condition ~~in~~ the critical section problem ensures that processes attempting to enter a critical section will eventually succeed, preventing indefinite blocking.
Starvation, where a process is continually delayed, violates the progress condition as it leads to processes being unable to make progress. Solutions to the critical section problem should aim to prevent starvation.

Date....................

Q3. The tradeoff between fairness and throughput in the readers-writers problem is about balancing the need for equitable access to a shared resource (fairness) with the desire to maximise the no. of concurrent operations (tradeoff).

To solve the readers-writers problem without causing starvation, implement a solⁿ such as readers-preference or writers-preference.

Q4 Busy waiting is a programming technique where a process or thread continuously checks a condition in a loop while waiting for an event, consuming CPU resources and potentially leading to insufficient resource utilization.

Other kinds of waiting in OS →

i) Blocking (Passive Waiting)
ii) Event-driven (Asynchronous) Waiting.
iii) Semaphore and condition variable waiting.
iv) Sleep or Delayed Waiting.

Busy waiting can be avoided using ~~efficiencient~~ efficient waiting techniques like blocking, etc.

Spiral

Date......................

Q5. Disabling interrupts for synchronisation in user-level programs in a single-processor system is not appropriate because it leads to loss of system responsiveness, disrupts multitasking, poses security risks, reduces portability and complicates debugging. User level programs should ~~be~~ use higher-level Synchronisation primitives provided by OS.

Q6. Interrupts are not appropriate for implementing synchronization primitives in multiprocessor systems because they lack predictability, mutual exclusion support, precise control over execution flow, scalability, and can lead to issues like deadlocks & priority inversions. Specialized sync. mechanisms like locks, semaphores and atomic operations are preferred for effective sync. in multiprocessor systems.

Q7. Monitors and semaphores are equivalent in their ability to solve the same types of sync problems. They can both be used to coordinate access to shared resources, among multiple threads or processes and ensure that critical sections of code are executed safely, although they have different implementations and usage patterns. The choice b/w monitors & semaphores often depends on the prog. language and specific requirements of the problem.

Q8.

```
~~# include <iostream>~~
# include
Monitor_alarm
{
        Condition C;
        int current =0;
        void delay (int ticks)
        {
              int alarms;
              alarms =  current + ticks;
              while ( current > alarms)
              C. wait (alarms);
            c.signal ;
        }
        void tick ( )
        {
            current = current +1;
            delay. signal;
        }
}
```

Q9. The possible sequences are B, C, A ($x=6$) or C, A, B ($x=36$) or C, B, A ($x=18$)

Q10 A The different possible values of x are 4.

B. There is only 1 possible value of x.

[illegible]