Instructors: **Ms. Ayesha Liaqat -- Mr. Tahir Farooq -- Mr. Haseeb Arshad -- Ms. Mahzaib Younas – Ms. Rukhsana Zafar -- Ms. Juhinah Batool.**

Due Date: ---------------- **CS-2006 Operating System** **Assignment # 5**

Ch # 6. Process Synchronization **Marks: 100**

INSTRUCTIONS

- ***Plagiarism***: *Strictly not allowed.* ***Binary checking*** *will be done to ensure no similarity with any other student's submission or content from the internet. A single line of plagiarism will result in rejection.*
- ***Code***: *Implemented in* ***C/C++*** *only on* ***Ubuntu/Linux****. Use synchronization tools like semaphores, mutexes, etc., where applicable.*
- ***Theory***: *Handwritten answers must be scanned into a* ***PDF****.*
- ***Submission***: *All code files (.cpp files) along with their* ***output*** *must be presented in a* ***Word*** *file with scanned pdf of theory questions on Google Classroom.*
- ***Hard Copy***: *Submit the hard copy of the theory questions in class.*
- ***Late Submission: No work will be accepted after the deadline.***

Question # 1: (15 Marks)

Consider a system where there are **n** processes sharing a critical resource. Each process can enter its critical section for reading or writing shared data, and processes must alternate between these operations (i.e., no two processes can perform a write operation concurrently). Design a solution using **monitors** to handle this critical section problem while meeting the following requirements:

- ✓ *Mutual exclusion is guaranteed.*
- ✓ *Bounded waiting is ensured.*
- ✓ *The system supports readers-writers synchronization.*

Question # 2: (15 Marks)

Consider the classical **Producer-Consumer problem** where there are two processes (a producer and a consumer) and a shared buffer with 10 slots. The producer places items into the buffer, and the consumer removes them. Modify the problem such that:

- ✓ *There are n producers and m consumers.*
- ✓ *The buffer size is k, where k is much smaller than the number of producers and consumers.*
- ✓ *Each consumer consumes more than one item in a single transaction.*

Develop a detailed **theoretical solution** using **counting semaphores** to synchronize access to the buffer and ensure no deadlocks occur. Explain the role of each semaphore in the solution.

Question # 3: (10 Marks)

Compare and contrast three hardware-based synchronization mechanisms*: Test-and-Set, Compare-and-Swap, and Fetch-and-Add*. Discuss the **advantages**, **disadvantages**, and practical application of each in modern operating systems.

Provide a comparative table and in-depth theoretical analysis of these synchronization techniques, with real-world examples of their usage.

Question # 4: (10 Marks)

Discuss the difference between **starvation** and **deadlock** in *process synchronization*. Provide a real-world analogy for each and describe a scenario in a distributed system where both can occur simultaneously. *Propose solutions to both issues using synchronization techniques discussed in the chapter.*

Question # 5: (10 Marks)

Implement a solution to the **Critical Section Problem** for n processes, ensuring the following conditions:

- ***Mutual Exclusion****: Only one process can be in its critical section at any time.*
- ***Bounded Waiting****: Each process must get a chance to enter its critical section within a limited number of turns.*
- ***Progress****: The system cannot remain idle if a process wants to enter its critical section.*

Write a ***C++ program*** using **Test-and-Set locks** to manage the critical sections of n processes. Ensure that no starvation or deadlock occurs.

Question # 6: (15 Marks)

Write a **C++ program** to solve the **Dining Philosophers problem** using **monitors**. Each philosopher should alternate between thinking and eating. Use the following requirements:

- *No philosopher should starve.*
- *The system should avoid deadlock.*
- *Allow a maximum of 3 philosophers to eat at the same time.*

Your program should include:

- ✓ *The implementation of monitors.*
- ✓ *Proper synchronization using condition variables.*

Question # 7: (15 Marks)

Imagine a system where three types of **processes (A, B, and C)** need to access a shared memory location. The following constraints apply:

- *Process **A** reads from memory.*
- *Process **B** writes to memory.*
- *Process **C** can only proceed when both **A** and **B** have completed their tasks.*

Write a ***C++ program*** using semaphores to synchronize these processes, ensuring no race conditions occur. Use complex synchronization logic to handle interactions between A, B, and C.

Question # 8: (10 Marks)

You are tasked with creating a **C++ program** that simulates a **ticket reservation system** using multiple threads. The system allows multiple users to simultaneously reserve tickets from a limited pool of available tickets. The requirements are:

- ***Multiple threads*** *represent users trying to reserve tickets at the same time.*
- *Each user can only reserve one ticket at a time, and they can try again if tickets are still available.*
- *The total number of tickets is limited, and once all tickets are sold out, no further reservations can be made.*
- *Synchronization must be used to ensure that no two threads can reserve the same ticket at the same time, avoiding race conditions.*

Use **mutex locks** to handle critical sections and ensure the consistency of the shared resource (the available tickets).

GOOD LUCK 😊.