Operating Systems(CS)

Assignment# 03

Deadline: Sunday, 7th May, 2023 (11:59 PM PST)

- This is an individual assignment.
- Zero marks will be awarded to the students involved in plagiarism.
- All the submissions will be done on Google classroom.
- You have to submit .cpp files in Zip Folder named after your roll no (20I-XXXX.zip). Naming convention has to be followed strictly.
- Be prepared for viva or anything else after the submission of assignment.

Problem Statement# 01

Implement your own **lock** and **unlock** mechanism (**methods**) using Bakery algorithm as a baseline to prevent race conditions and ensure mutual exclusion. The solution must be **efficient**, **fault-tolerant**, and **scalable** to handle multiple concurrent requests while minimizing the risk of **starvation**. This involves creating a multi-threaded program with shared resources, implementing a data structure for the lock, and defining operations for acquiring and releasing it.

Detailed Description:

1. Create a class to represent the lock, with methods to acquire and release the lock.
2. Define a data structure to represent the ticket, with fields for the ticket number and a flag indicating whether the process or thread is currently in the critical section.
3. Implement a method for acquiring the lock that assigns a ticket number to the process or thread and waits until its turn arrives. This can be done using a busy-wait loop that repeatedly checks the ticket number of all other processes or threads and waits until its own turn arrives.
4. Implement a method for releasing the lock that resets the ticket flag and signals the next process or thread in the queue. This can be done by incrementing the ticket number and notifying the next process or thread in the queue that it is now their turn.
5. Use atomic operations such as compare-and-swap or test-and-set to ensure atomicity and prevent race conditions while accessing the ticket data structure.

Instructions

You can use only C/C++ but cannot use any OS based locks (e.g. mutexes / semaphores etc.). The solution should be implemented in Linux (preferably Ubuntu) as demo will be on machines where Ubuntu is installed. Even if you use any other Linux/Mac distro or any other OS, make sure your program runs as expected in Ubuntu environment.

Problem Statement# 02

A small island has a historic bridge with only one lane that can only hold a limited number of vehicles at a time without risking collapse. At most, four cars or a combination of one bus and two cars are allowed on the bridge simultaneously. However, two buses cannot be on the bridge at the same time. If two buses arrive at the same time, only one can cross while the other waits. All vehicles can travel in one direction only, and there may be cars or buses waiting on the other side of the bridge.

To manage traffic flow, you need to implement a mutex synchronization mechanism in a multi-threaded program. A master thread will spawn vehicle threads at regular intervals, and each vehicle thread will call the functions "**ArriveAtBridge(int direction, int vehicleType)**" and "**ExitTheBridge()**" to control traffic flow.

The "**ArriveAtBridge()**" function takes two parameters: the direction the vehicle wants to go (0 or 1) and the type of vehicle (car or bus). When a vehicle thread arrives at the bridge, it calls "**ArriveAtBridge()**" and waits until it is allowed to cross the bridge. The function ensures that no more than four cars or one bus and two cars are on the bridge at any given time. If a bus arrives while there are already two cars on the bridge, it waits until the bridge is clear. If two buses arrive at the same time, only one can cross while the other waits.

The "**ExitTheBridge()**" function is called when a vehicle thread gets off the bridge, potentially allowing other vehicles to get on. The program should run for a user-specified number of minutes before exiting.

Instruction:
You can use only C++ and pthread library to implement the above-mentioned scenario. The solution should be implemented in Linux (preferably Ubuntu) as demo will be on machines where Ubuntu is installed. Even if you use any other Linux/Mac distro or any other OS, make sure your program runs as expected in Ubuntu environment.