Lab 4: Thread Synchronization 1

Total points: 100

For each of the following problems,

1. Identify the correctness constraints of the problem
2. Specify the conditions that each method must wait for
3. Write down the shared state that you will use to check these conditions

Assignment 1 (20 points)

Develop a program which has a shared data and two threads. The shared data contains an integer variable which is set to 0 initially. The first thread will increase the variable a number of times (1000000 times) and the second thread will decrease the variable the same number of times (1000000 times). You should protect critical sections and thus prevent race conditions.

a. Using `synchronized` methods
b. Using `ReentrantLock` class
c. Using `Semaphore` class

The program creates, starts two threads and waits for the two threads to terminate and display the variable's value. Explain the result.

Assignment 2 (20 points)

Develop a **thread-safe** queue of integer (TSQueue). The thread-safe queue has two methods `addLast` and `removeFirst`. The `addLast` method inserts an integer value into the end of the queue and the `removeFirst` removes the first element of the queue. Test the class with several threads to insert and remove integer values into the queue.

Assignment 3 (20 points) – The Bounded-Buffer Problem

A bounded integer buffer is a fixed size FIFO queue and is implemented by an array of integer values. The queue has methods `add` and `remove`. The `add` method adds a new integer value to the end of the queue and the thread that call the `add` method must wait if the queue is full. The `remove` method removes the first element of the queue and the thread that call the `remove` method must wait if the queue is empty. Test the class with several threads to insert and remove integer values into the queue.

Assignment 4 (20 points) – The `Barrier` Class

A barrier allows a set of threads to all wait for each other to reach a common barrier point. The situation where the required number of threads have called `await()`, is called tripping the barrier.

- Constructor: `public Barrier(int parties)`

    - Creates a new `Barrier` that will trip when the given number of parties (threads) are waiting upon it.
    - `parties`: The number of parties (threads)
- `public void await()`:
    - Waits until all parties have invoked await on this barrier.

Write a `main` method to test the `Barrier` class.

Assignment 5 – Old Brigde (20 points)

An old bridge has only one lane and can only hold at most 3 cars at a time without risking collapse. Create a monitor with methods `arriveBridge (int direction)` and `exitBridge ()` that controls traffic so that at any given time, there are at most 3 cars on the bridge, and all of them are going the same direction. A car calls `arriveBridge` when it arrives at the bridge and wants to go in the specified direction (0 or 1); `arriveBridge` should not return until the car is allowed to get on the bridge. A car calls `exitBridge` when it gets off the bridge, potentially allowing other cars to get on. Don't worry about starving cars trying to go in one direction; just make sure cars are always on the bridge when they can be.