CSCI-UA.0480-001 Special Topic: Multicore Programming
Homework 2
Due June 26, 2019

Please solve the following and upload your solutions to your private GitHub repository for the class as homework2.pdf by 11:59pm on the due date above. If for some reason this poses a technical problem, or you wish to include diagrams that you don’t wish to spend time drawing in a drawing application, you may hand in a printed copy (not hand-written) at the beginning of class (5:45pm) on the day of the deadline. **Unlike labs, late homeworks will be assigned a grade of 0.**

This homework will give you some extra practice thinking about synchronization and thread safety. It is intended to help you hone the skills you need for Lab 1. It will also help you prepare further for the midterm exam.

1. **Warm-up:** As we discussed in class, explain at least two important differences between semaphores and condition variables.

2. What criterion or criteria must a correct mutex fulfill? Without looking it up, reason out which of these criteria (and/or possibly others) must be true for other thread synchronization primitive (namely condition variables and semaphores), and explain why.

3. Consider the following code:

   ```c
   1 static double sum_stat_a = 0;
   2 static double sum_stat_b = 0;
   3 static double sum_stat_c = 1000;
   4 int aggregateStats(double stat_a, double stat_b, double stat_c) {
   5     sum_stat_a += stat_a;
   6     sum_stat_b -= stat_b;
   7     sum_stat_c -= stat_c;
   8     return sum_stat_a + sum_stat_b + sum_stat_c;
   9 } 
   10 void init(void) { }
   ```

   Use a single pthread mutex or std::mutex to make this function thread-safe. Add global variables, and content to the init() function, iff necessary.

4. Let’s make this more parallelizable. We always want to reduce critical sections as much as possible to minimize the time threads need to wait for a resource protected by a lock. Modify the original code from question 3 to make it thread-safe, but use three mutexes this time, one each for sum_stat_a, sum_stat_b, and sum_stat_c. Hint: explain what guarantee(s) you provide about the return value of aggregateStats().
5. **In your own words**, describe one of the possible causes of the Lost Wakeup Problem, including a scenario that triggers this cause, and how to fix it. Use C++ or pseudocode in your explanation iff you find it necessary. If you do research and find other causes beyond what we discussed in class, please note your source(s).

6. Using any pseudocode style you prefer, outline the functionality of a thread-safe queue **twice**, with at least the methods:

   - **push(item)**: Add item to the end of the queue
   - **pop()**: Remove item from the beginning of the queue and return it
   - **listen()**: wait for and then return an item when available

   these two ways:

   (a) Built using mutexes, semaphores, and/or condition variables, and then
   (b) Built only using semaphores.