CSCI-GA.3033-017 Special Topic: Multicore Programming
Homework 3
Due November 19, 2018

Please solve the following and upload your solutions to your private GitHub repository for the class as homework3.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 (6:20pm) on the day of the deadline. **Unlike labs, late homeworks will be assigned a grade of 0.**

As is university policy, instances of cheating will be taken very seriously. If you use any source for reference, including speaking with other students and/or consulting internet resources, you MUST cite those sources and/or people in your assignment. Any instances of cheating will earn you a zero on the homework, a visit to the administration, and potentially other punishments including and up to expulsion from the class and/or school.

---

1. Why does adding a sentinel simplify a concurrent queue? How does it make it possible to construct a lock-free queue?

2. Consider the performance of a program responsible for taking a set of 10,000 large numbers (e.g., between 1e20 and 1e40), and determining the prime factors of each number. A user has a four-core machine, and wants to run this program in the background while they play a fast-paced game (that spawns two threads, one for the game logic and one for rendering) at the same time.

    a. What kind of bounds (minimum and maximum) can you give on the number of threads the prime factor program should spawn? Why not fewer or more?

    b. What performance metric(s) would you as a programmer use? What would you optimize for in this situation, not just in your own program, but system-wide?

3. Consider the following two threads accessing shared global variable `global_thread`. (a) What kind(s) of concurrency bug(s) are present in this code? (b) **Describe** (ie, don't just list) two possible ways to fix this. If it helps, use code to demonstrate why these are good fixes.

Thread 1	Thread 2

`if (global_thread->proc_info) {`
`  fputs(global_thread->proc_info);`
`}`
	`global_thread->proc_info = nullptr;`

4.Regarding Compare And Swap (CAS):

    a. Why do we want to try to make lock-free algorithms with CAS instead of mutices or similar synchronization primitives?

b. Review our CAS-based queue `enqueue()` method, and to the best of your ability, explain why we need to allow the `tail` pointer to point to either the last or second-to-last entry in the queue instead of only the last one.

Think about the properties of CAS, what we talked about in class, and the slides. Show how you think: if you still don't quite understand it from class, you may want to explain your thought process as you try to work through examples of how two threads might interleave their `enqueue()` operations (not just the one explained in the slides). You should come up with some fundamental deductions about this and other CAS-based algorithms and the global states visible to other threads in a CAS-based algorithm that wouldn't be visible with mutex-based critical sections. Again: when in doubt, show your work!

5. What are the advantages and disadvantages of binary instrumentation versus dynamic binary translation?