CSCI-GA.3033-017 Special Topic: Multicore Programming
Homework 3
Due November 10, 2016

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 (7:10pm) on the day of the deadline. **Unlike labs, late homeworks will be assigned a grade of 0.**.

---

1. Consider the following two threads accessing shared global variable `thr_glob`. (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 (thr_glob->proc_info) {`
`  fputs(thr_glob->proc_info);`
`}`
	`thr_glob->proc_info = nullptr;`

2. What causes the ABA problem? How can you solve it, and why does that solve it? What extra hardware and software primitives, if any, are required to implement a fix to the ABA problem?

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

4. What is linearizability? If an algorithm or technique is linearizable, what does that mean in practice? For example, you could motivate this with a CAS example, or another example of your choosing.

5. Consider the following code implementing a reader-writer lock (from Lecture 6). Describe some ways you could shift the priority between favoring readers, favoring writers, and trying to balance both equally. Consider especially who you signal, and when.

```
int a_readers, a_writers, p_readers, p_writers   // Active & pending
mutex mut, cond_var read_cond, write_cond
```

```
reader_lock():
    lock(mut)
    while a_writers + p_writers:
        p_readers += 1
        read_cond.wait(mut)
        p_readers -= 1
    a_readers += 1
    unlock(mut)

reader_unlock():
    lock(mut)
    a_readers -= 1
    if !a_readers && pwriters:
        write_cond.signal()
    unlock(mut)
```

```
writer_lock():
    lock(mut)
    while a_writers + a_readers:
        p_writers += 1
        write_cond.wait(mut)
        p_writers -= 1
    a_writers += 1
    unlock(mut)

writer_unlock():
    lock(mut)
    a_writers -= 1
    if p_writers:
        write_cond.signal()
    else if p_readers > 0:
        read_cond.broadcast()
    unlock(mut)
```