CSCI-GA.3033-017 Special Topic: Multicore Programming

Homework 3

Due November 27, 2017

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.

In Lecture 8, we looked at implementations of a Concurrent **Ordered** List, one using per-node locks, another using CAS. The code for lock-free insertion and deletion with CAS was provided in the lecture; the code for lookup(), insert() and remove() in the per-node locks case was not given.

1. Write **pseudocode** for:
    a. The structure of a node (eg, define a `struct node` and its fields).
    b. The `lookup()` operation, which must take a key (of a templated type T) and return struct node* pointers to `prev`, `cur`, and `next`. Remember that prev is the node before the point where this key is or would be, cur is the node **at or beyond** the key in question, and if the key is found, next is the next node after the cur node. Remember that you need to grab the per-node locks during the `lookup()` operation.
    c. The `remove()` and `insert()` operations, using the output of `lookup()`. Remember that these must unlock `lookup()`'s locks.
2. An easy start: write the C++ code corresponding to your struct node.
3. Write the C++ code for your `lookup()` operation.
4. Write the C++ code for your `remove()` and `insert()` operations, using the output of `lookup()`.
5. Explain how and why you handled unlocking `lookup()`'s locks, especially for a pure lookup() operation.

Bonus (up to 15% extra points on this homework): Actually compile this, including with a simple test harness to launch threads to insert and remove random elements. You may reuse any test harness **you** have written for labs in this class.