CSC 412 – Operating Systems

Lab Session 09, Spring 2022

Monday, April 11th and Wednesday, April 13th, 2022

What This Lab Is About

In this lab, you are going to add mutex locks to an existing multithreaded program.

The Handout

The handout for this lab is a simple multithreaded version of bubble-sort (for an array of int): pick a random index `i` over the array, compare the values at locations `i` and `i+1`, and swap the values if they are out of order.

This works fine (but not very fast) in a single-threaded program. However, in a multithreaded version, since two threads (or more) could be accessing the same locations, it is very possible to end up with some values duplicated and some other missing. Adding locks will fix this problem. Note that I provide both a C and a C++ version of the multi-threaded randomized bubblesort. Pick the one you prefer to work with (but keep in mind that all remaining assignments are done in C++).

Task 1: Synchronize with a single lock

Define (and initialize) a single mutex lock to control access to the entire array. A thread must acquire the lock before it tests and swaps two elements.

Task 2: Multiple locks

Obviously, the single-lock version doesn't exploit multithreading well: while a thread is testing two elements at one end of the array, it is blocking another thread that want to operate on a different part of the array. So, we can go for a different strategy: Define a lock for each element of the array. Now, before testing and possibly swapping two elements, a thread must acquire the locks for both elements.