Assignment 2: Read-Write Lock

In this assignment you will be creating a read-write lock using primitives in C/C++. You will not be allowed to use any locks/mutexes already created in C/C++. Atomics and condition variables are allowed.

Read-Write Lock:

A read-write lock is a lock that has two modes of acquisition, reading and writing. If a lock is acquired by a reader, the lock can also be acquired by any other readers. However, if the lock is acquired by a writer, then the writer has exclusive access to the resource (no readers, no other writers).

In short, there are 3 types of valid states:

No one has the lock (readers = 0, writers = 0)
A reader has the lock (readers = x > 0, writers = 0)
A writer has the lock (readers = 0, writers = 1)

Read-write locks are used for filesystems (a file can be read by any number of processes concurrently, but cannot be written to while others are reading, nor can more than 1 process write to a file) and for situations where a resource cannot be updated atomically and is invalid to use while being updated.

Your Assignment:

You will be implementing your own read-write lock. You should make use of the code skeleton given to you. The only methods you need to implement are readLock, writeLock, readUnlock, writeUnlock, and the constructor. You are free to add other methods if you wish, but those are the only methods that the main program interfaces with.

You are additionally given a testing file for running benchmarks. You should not need to change anything in the test.cpp. If you have implemented the lock correctly, you should not see any errors printed when running the test cases.

Once your lock is working correctly, report your average run time on each of the 6 test cases.

458 Additional Work:

Make two versions of your lock - one that prioritizes writes (i.e. once a writer wants to acquire the lock, a reader cannot acquire it), and one that prioritizes readers (a reader can always acquire the lock, so long as a writer does not currently have it). Measure the performance of these two locks on each of the 6 inputs.

Try the program with a standard mutex instead of a read-write lock (implement your own standard lock, don't use the built in C/C++ mutex). Measure the performance of your lock on each of the 6 inputs. It will likely be worse on all inputs - are there any inputs where it is less worse?

Summary of Submission:

Your submission should include:
Your version of rwl.cpp
A readme explaining your code
A text file called summary.txt that gives the average execution time of your lock used in test.cpp for each of the 6 input files.

(458)
Both versions of your read-write lock
Your implementation of a standard mutex
Additionally in summary.txt report the performance of your mutex on all 6 inputs. Make observations about which inputs it's better/worse on proportionally to your read-write lock.