CSMC 23010-1 Due May 14, 2019

Homework 3a

Professor: Hank Hoffmann *TA: Will Kong*

Design Review

We will do the design review in class on 5/7/2019.

Theory

There will be no theory in this assignment. Please start the programming early!

Programming

This assignment is all about locks and data contention. We will build a set of different lock types. In the following assignment, we will use them to balance the load of our packet distribution system. In particular, we will be building the following set of lock types, L:

- **Test and Set Lock** (TASLock): This lock (Figure 7.2 in the text) makes use of the gcc atomic builtins, which encapsulate the read-modify-write atomic instructions on Intel hardware.
- **Mutex** (pthread_mutex_t): This is the standard lock in the POSIX threads library. We will compare our locks against the pthread mutex, but you should not use the mutex to implement any of the other locks.
- **Anderson's Array Lock** (ALock): This lock (Figure 7.7 in the text) has a fixed array of locations (equal to the total number of threads who might simultaneously try to grab the lock) furnishing each thread with a private location on which to spin. Take care to implement it with padding, as described in Figure 7.8.
- **CLH Queue Lock** (CLHLock): This lock (Figures 7.9 and 7.10 in the text) creates a linked list of nodes, so that each thread watches a distinct memory location. This is intended to decrease memory contention, at the expense of more work (*e.g.*, more overhead in managing the list etc.).

We will also be extending the locks to include a tryLock method. Essentially, tryLock returns true if the lock is acquired (the caller is then responsible to call unlock eventually) and false if the lock is held by another thread at the time of calling. Notice that there is a race here—you could test whether the lock is held, decide that it is not and then try to lock it, creating an opportunity for another thread to lock it in between the two steps. This behavior is acceptable—once the tryLock decides to grab the lock, it is permitted to wait in the case of this race condition. The purpose is to merely reduce the instances of threads waiting for locks that are already held.

To gauge the relative merits of this set of lock algorithms, we will conduct experiments:

1. **Work-based Counter Test:** We will have all the threads increment a single counter protected by a lock. In this case, however, we will count to a fixed number (BIG) and each thread will be responsible for approximately BIG/n increments (note that only the FCFS locks can guarantee that each thread gets an equal number of increments). You will measure the time taken to perform all of the increments. You are responsible for choosing BIG. Please make an informed choice: it should be big enough that the variance between tests is low, but not so big that other students are blocked from using the machines.

- SerialCounter This is analogous to the above, a single thread that increments the counter. Configurable by:
  - B (B) the number to which we are counting.
- ParallelCounter Again, this application launches n threads, which all try to grab the lock and increment the counter. This code is configurable by:
  - B (B) the number to which we are counting.
  - numThreads (n) the total number of worker threads
  - lockType (L) the lock algorithm

2. **Your Own Lock Test:** The above test is standard for lock implementations, because it evaluates the lock in the most highly contended scenario. For this assignment, consider a second test that you would like to investigate. For example, you could vary contention and try to find the point at which different lock types no longer matter. As another example, instead of counting to a set number, you could consider counting for a set amount of time and measuring the number of increments that each lock achieves. You could also compare the lock algorithms here to one of the locks from earlier in the book (Peterson, Filter, Bakery, MCS, TTAS, Backoff). You are free to choose any aspect of lock performance you would like to investigate.

Experiment

You will perform the following set of experiments across various cross-products of these parameters. Each data point is a measurement taken on a dynamic system (a computer...) and is thus subject to noise. As a result, some care should be taken to extract representative data - we would propose running some reasonable number of experiments for each data point and selecting the median value as the representative.

A reminder on describing performance: Use quantitative language when describing your observations. For example, strategy x provides a 10% improvement compared to strategy y. Avoid qualitative language of the type, "Strategy x is a little better than strategy y."

A reminder on measuring performance: All code should be compiled with optimizations on (`-O3`, for example). You should describe what optimizations you are using. This can be tricky because many lock implementations will work without optimizations and fail with them on. Such behavior usually means that some pointer was not appropriately declared to be volatile.

Counter Tests

1. **Idle Lock Overhead** Run SerialCounter for the Work-based experiment to find the throughput (increments / ms) of a single processor incrementing a counter (a volatile, which admittedly has some overhead). Then, run ParallelCounter with $n = 1$ and all L ($\in \{\text{TASLock}, \text{BackoffLock}, \text{Mutex}, \text{ALock}, \text{CLHLock}, \text{MCSLock}\}$). Provide the speedup (*i.e.* ratio of ParallelCounter throughput to SerialCounter) for each L in a table. Do these results make sense, given the relative complexity of the uncontended path through each lock method? You are responsible for justifying the answer to these questions. Your answers should persuade graders that you understand the internals of your code. If you cannot explain a result with data (taken from experiments or from code analysis) then you should ask for help.

2. **Lock Scaling** Run the work-based approach to ParallelCounter for all L and $n \in \{1, 2, 4, 8, 14\}$. Plot speedup (ratio of ParallelCounter throughput to SerialCounter—it *should* be less than 1) for each L (*i.e.* one curve for each L) on the Y-axis vs. n on the X-axis. Describe your results—what is your hypothesis about the performance, particularly of the more complex locking algorithms? Demonstrate your understanding and think about what we described in class and in the book as justifications for these various lock algorithms.

3. **Your Test:** For this section, it is your responsibility to tell us (the instructor and TAs) what parameters you are planning to vary and make sure to describe the results and analysis so that we understand why the experiment was significant. In general, computer science does not value negative results (for example, almost no papers are published that describe why a seemingly good idea did not work). In this class, however, negative results are okay. If you set out to test a hypothesis and find no evidence to support your hypothesis, that is fine as long as you spend some time explaining the results you did achieve.

Submission

As with previous assignments, a design and test document should be submitted during the design review in class (see date above). Since everyone will implement the same locks, the design document should focus on correctness testing, performance hypotheses for the detailed experiment, and a description of your own experiment and what you hope to learn.

You are responsible for submitting a writeup and all code associated with this project.

You write up should be concise, but it should primarily demonstrate your understanding of the code and experiments and your engagement with the problems. The writeup should include quantitative language that clearly supports your statements. The writeup should stand alone as a document that explains the code you developed, the experiments you ran and the conclusions you drew. The writeup should have a section for each of the experiments described above and an additional section describing your test plan.

The grades will be primarily based on the understanding demonstrated in the writeup, so please spend time on them. This means you need to start early and get help if there is a result you don't understand. For partial credit, higher grades will be assigned to reports that clearly explain results for some smaller number of experiments than those that have a poor explanation for a large number of experiments. (I sincerely hope that everyone completes and explains all experiments, of course).

You should submit your working directory and all the associated code including test code. It should be easy for old people to run your tests and attempt to recreate your results.

You are responsible for understanding what is required of you on this assignment. If you are not clear about what to do, then ask clarifying questions.