Assignment 3 – CMPT-300

Synchronization:

Background

In this problem you will create your own synchronization functionality. You will then test this functionality against two pthread implementations, namely pthread mutex and spinlock. For comparison purposes, all measurements **MUST** be done on machines **PHYSICAL** in the CSIL lab.

We have provided some base files here: http://www.sfu.ca/~rws1/cmpt-300/base-a3.zip
You must implement the missing functionality in these files.

Base Files:

atomic_ops.h: Contains two atomic operations Test-And-Set (TAS), as well as Compare-and-Swap(CAS).

Sync.h/Sync.c: This is where you will implement your version of the spinlock, exponential back-off mutex, and a queue lock(FIFO).

Main.c/main.h: This is where your testing code will go, a skeleton is already provided. Please make sure to implement the command line options exactly as specified.

Actual Assignment:

For this assignment, you will implement three locks: a simple spinlock, an exponential backoff mutex lock and a Queue lock. Put your function implementations in a single file, called sync.c, and your structure definitions in sync.h. Skeleton files are provided, but you may have to add functions and datatypes to these files. Unless otherwise noted, all functions should return 0 on success and -1 on error.

Your implementations should be recursive, i.e., if the owner of a lock tries to get the same lock, it should not deadlock, but just require another call to *_unlock() to fully unlock the section. Note that this is not how all mutex implementations work -- many will simply deadlock if the holder of a lock attempts to get it again. The Linux pthread implementation has both standard and recursive mutexes: see the pthread_mutex_init(1) man page for details.

Your implementation should not allow starvation of any thread waiting for the lock. That is, if two threads are repeatedly locking and unlocking a lock, that should not prevent a third thread from eventually getting the lock. (Of course, if one thread never unlocks the lock, the others will starve, but that's a programming error, i.e., not your problem!)

Detailed Instructions:

1) First, you must complete the processInput(int argc, char *argv[]) command in the source file main.c. Basically, this command must process input from the command line and set the testing variables.

2) The prototypes for the functions you must implement are found in **Sync.h/Sync.c.** You are to

implement versions of both a spinlock and a mutex(exponential backoff). For the spinlock the first version will directly use the TAS atomic operations. The second version will employ a Test-And-Test-And-Set (TTAS) technique. *Note: These two versions will be almost functionally identical except one will check the status of the lock before attempting to acquire it. Also, the unlock command stays the same regardless of the use of TAS or TTAS.* The mutex need only be implemented using the TTAS method. ***Hint: This is a good check point to start your report, the next lock can be tricky, see point 4 for testing instructions.***

3) For your final lock, you will implement a Queue based lock. One example of a queue based lock is a "ticket lock" (https://en.wikipedia.org/wiki/Ticket_lock). You are not limited to the choice of ticket lock; as long as you create a lock that preserves FIFO of the threads. Extra credit can be given for particularly clever or efficient implementations of this lock. *Hint: You will need to create an atomic increment and return old value function for the get ticket function. To implement this lock, you will likely have to use CAS (Compare-and-Swap) instead of TAS.*

4) You will complete the testing code found in main.c/main.h. Then vary the number of iterations of the shared variable as well as the number of threads contending for the lock. Also, vary the amount of work inside and outside the critical section. Try to find something interesting here, as you will write it up in the report.

What to turn in:

Source code:

You are asked to electronically turn in your source files and a Makefile. Attach a README file describing the name of the executable, special compiling instructions, or anything else special you want to let us know. Also include your measurements in this README. The README file should be in plain text format. Please turn in at https://courses.cs.sfu.ca/

Report:

For this assignment you will also turn in a maximum 3 page report on your findings, in PDF format. Your report should be written in the style of a research paper. Please use full sentences and graphs/tables to convey results. Describe interesting and unexpected finding, using evidence to conjecture why the system behaves the way it does. The page limit does not include a cover-page and references, and large figures/graphs. Please describe what you have found, complete with data, interpretations and insights. Also, convey the results as clearly as possible using graphs or tables where appropriate. Further, you must briefly explain your measurement choices and justifications. We will evaluate this assignment based on the quality of the report, quality of the implementation and a proper testing methodology.