CS6378 Project 1 – Ricart Agrawala Mutual Exclusion with Optimizations

Implementation Details

This implementation of distributed system running the Ricart Agrawala Mutual Exclusion algorithm with the Coucairol-Carvalho optimization is in C and C++. The code is in the program1 folder. It utilizes pthreads, sockets, and parts of the standard Unix library. Therefore it will not run on a Windows platform.

The instructions specify the nodes must wait some number of "time units" between mutual exclusion invocations. The time units I specified are *milliseconds*, implemented using the usleep() Unix time function.

A general program flow is given below

1. Node starts up and is given its ID
2. Node reads addresses.txt for locations of all nodes in the network
3. Node connects to all other nodes in the network
4. Node begins computation
5. Node requests mutual exclusion repeatedly, waiting between invocations
6. Node sends end of computation messages
    a. Node 0 waits for all computation end messages
    b. Node 1-N sends computation end message to node 0
7. Node 0 sends network end of computation messages
8. Node records its data in results<ID>.txt
9. Node shuts down

Pitfalls

Initially I used a completely object oriented approach, using encapsulation, inheritance, and other object oriented design paradigms. Later I realized that pthreads are C-style library, and therefore are not object oriented in nature. They don't work well with starting on class member functions due to an implicit this pointer. So I turned to object oriented threading: I tried to use C++0x on my local Ubuntu virtual machine, but g++ 4.6 doesn't support them yet. I switched to using Boost threads. It didn't work. The campus cluster doesn't have Boost installed, and my account didn't have permission to install it. So I scrapped all my previous work and went for a C-style approach with no real classes and wrote the whole thing in the last 48 hours.

Observations

During the testing phase of this implementation, the campus cluster was very busy. There were multiple times when the calculation thread on a node (the thread which simulates calculations and invokes mutual exclusion calls) would be delayed for minutes on startup, thus removing that node from critical section contention and reducing message traffic globally.

The instructions specify that even numbered nodes increase their wait duration between mutual exclusion invocations after 20 invocations. It can be seen the results spreadsheet in the results folder that the increase in wait duration between mutual exclusion invocations does not drastically affect the waiting time.

Results

The results are recorded in the results folder. Results were obtained running with 3 nodes and 5 nodes. The cluster was too busy to finish any run with more nodes.

Acknowledgements

Parts of the code are modified from other's ideas or code

File	Creditor
hr_timer.h	http://allmybrain.com/2008/06/10/timing-cc-code-on-linux/
hr_timer.cpp	http://allmybrain.com/2008/06/10/timing-cc-code-on-linux/
Mutex.h	Blake O'Hare
ConditionVariable.h	Blake O'Hare
NaiveConcurrentQueue.h	Blake O'Hare

Additional thanks to Deckmaster and BlackDragonHunt for listening to me gripe about non-object oriented thread libraries.

Various other internet resources such as tutorials and reference pages helped including but not limited to

1. https://computing.llnl.gov/tutorials/pthreads/
2. http://beej.us/guide/bgnet/output/html/multipage/index.html
3. http://cs.gmu.edu/~white/CS571/pthreadTutorial.pdf
4. http://antonym.org/2009/05/threading-with-boost---part-i-creating-threads.html
5. http://stackoverflow.com/questions/6303884/program-compiles-fine-with-boost-libs-but-error-when-running-it
6. http://www.boost.org/doc/libs/1_51_0/more/getting_started/index.html
7. http://www.boost.org/doc/libs/1_51_0/doc/html/thread/thread_management.html
8. http://www.cplusplus.com/reference/
9. http://en.cppreference.com/w/cpp/thread/thread