Jake Christensen W01023683

CS 322 Wtr 2014

Assignment 5 writeup

We have left the comfortable confines of MPI behind in this weeks assignment, and have traded in the message passing interface for the POSIX threads library. This time our goal was to create a producer/consumer thread relationship using some of the tools available in the library. We first built the paradigm using conditional variables to synchronize our threads. We then repeated the same process, this time using semaphores.

The setup of the program in POSIX threads did not feel as intuitive as MPI to work with. I spent many a run deadlocked somewhere in my code. Threads are very difficult to debug, as you never really know exactly what's going to happen yet. I found the best method was to add lots of printf statements and run the program several times. Slowly I could begin to pick out suspect areas. Memory management and my general lack of experience in c further played a factor, especially in the semaphore implementation. It took longer than it should have to intitialize the semaphore array and correctly set it in a global scope for the program. When trying to use a stack variable, the program kept crashing, and forced me to rely on malloc and the heap instead. Once I got over that little snag, I managed to get the program running stable.

I wanted to add the ability to run tests with large numbers, so I added two arguments to the creation of the program. The first controls how many iterations each thread will do through the produce/consume cycle, the second controls how many threads are actually created. I then came up with a testing scheme. Inside of the critical sections of both the consumers and producers, I added debug information that would allow me to track how many times the producer produced, and each consumer consumed. This allowed me to ensure that every consumer was checking his data the correct number of times, allowing me to assert with some certainty that the threads are doing what they're supposed to. Mutexes also came in handy, as I could have some assurance as to the timing of my output if it was in a mutex. Using the mutexes, I could separate a single round of produce/consume from each other and further ensure that each thread was making a read before resetting the buffer.

If I had to make a choice between the two, I would definitely prefer doing work in MPI. I imagine that like any good library, POSIX threads have certain situations in which they are the best candidate for the job. The trouble is they're so volatile, and it takes a lot of head scratching just to make the program work. In a larger application, it could easily become impossible to assure that there are no deadlocking situations within the code. If I have the choice, I will stick to some more robust abstractions for my multithreading needs.