Concurrent Project Proposal

Evan Schlomann

Andrew Kee

February 16, 2015

Project Topic

When programming for multiprocessor/multithread computers, it is necessary to implement locking systems to prevent from overwriting shared resources. During our class we have already seen multiple locking mechanisms. We see that there are many ways to provide security for objects in our programming.

We plan to write and test multiple locking systems to discover where some locks perform well and where others do not. We will be testing how they perform over the number of cores available, the amount of memory the system has, the number of threads being used, etc. These statistics will allow us to determine which locks we should use when, and to help better understand locking systems.

Concurrency Issues

The main issue with concurrent programming is the manipulation of shared variables. To protect these shared resources from being overwritten by separate threads, locks are used. Implementing locks requires the design of locking systems and the construction of these systems as well.

One of the more common locking systems is the Peterson Lock. This lock uses a simple idea, and is used for two threads. The lock has two variables, flag (used to signal the desire to go to the critical section), and the victim (when the lock is called, the calling thread declares itself as the victim, and allows the other thread to go first).

The Peterson Lock can be expanded to be used for n-threads. Other systems, such as the Bakery Lock and the Filter Lock are designed for n-thread systems. We will have to implement these locks and write tests to acquire statistics for analysis. With the analysis, we will then develop a performance model that will explain the data.

Proposed Solution

In addition to the locks that we develop for assignments in class, we will implement at least 2 (likely more) other locks to do performance comparison. Additionally early testing will be done to determine a case where there is not a dominiant performer and develop a hybrid lock that will best meet the specific needs of this case.

Though this is by no means definitive. This is the tentative list of the locking methods to be used, this list may be modified and added to as the class continues and more algorithms and locking methodologies are discussed

- Java's lock interface
- Peterson Tree Lock (Class)
- Filter Lock (Class)
- Bakery Lock (Class)
- Eisenberg & McGuire algorithm
- Szymanski's Algorithm

Based on inital testing with Java's built in lock class and the locks written for homework assignments, a case where there is no clear best performer can be identified. A hybrid locking algorith can then be developed from the existing locks in order to best serve that specific case.

Once all of the locks are developed, the rigorous testing can begin. The following are variables that need to be modified and tested to obtain meaningful performance comparison:

- Number of threads
- Number of machine cores (and memory differences)

- High vs low shared resourse contention
- Usage requirement of program

Timeline

March 15th - Substantial development completed on locking mechanisms (those done in class are incorporated into the current project)

April 3rd (Project Checkpoint) - All different lock implementations covered in class as well as two others are working without issue, and the joint solution in the case of (NEED A CASE HERE) is close to functional. Optimization and development is ongoing but close to finished. Testing of the different locking mechanisms (including the hybrid model) is ready to begin.

April 20th - Testing is complete and all work with the project is done minus small changes that need to be reflected in the final project. All work from here to the due date is working on the final project submission and documenting code.

April 27th (Project Due Date) - Project is final and complete.