CONCSYS - ASSIGNMENT 3

Upload your solution on ILIAS as a ZIP or TAR.GZ including all your files (sources, text file with explanations, ...). In case of questions, please address them during the lab or by email to yaroslav.hayduk@unine.ch

Deadline: April 7th 2015

Exercise 1

Implement a class called CASLock, which implements the java.util.concurrent.locks.Lock interface. You are required to implement the *lock* and *unlock* methods (for the other interface required methods you can just keep them blank or returning a not significant value depending on the case). For the locking functionality make use of an AtomicInteger object from the types provided by the java.util.concurrent package in a similar fashion as the TASLock described during the course. The AtomicInteger will be considered "locked" when its value is 1 or "unlocked" when its value is 0. Refine your class as CCASLock in a similar way as the TTAS lock described during the course.

Compare your classes performance to the Peterson lock (the unfair version) in the same context from the last assignment - protect a shared counter incremented by the threads by using the lock, and maintain also one local counter per thread to track the accesses in the critical section. Use a counter of 300000. Perform experiments on 4 and 8 threads (without the single processor restriction) and include in a file named E1.txt statistics as in the previous assignment (case, counter value, number of increments done by each thread - the value of their local counter, number of threads, execution time). Include these statistics, along with your opinions about the results in a file named E1.txt. (Note: If you did not implement the Peterson lock for the previous assignment just skip the comparison considerations and report what you can observe regarding the performance of the two locks in this exercise).

Exercise 2

Use your newly designed lock from (preferably the CCAS version) or the TTAS lock (in the spin lock course) to extend the 2-thread lock-free cyclic array queue presented in the Linearizability course (code provided below). Do this in the following two ways, so the result will support any number (up to n) of enqueuer and dequeuer threads, not just one of each type.

1. Use a single Lock to protect both the head and tail pointers. (Hint: pay attention to when you need to hold the lock and when you need to release it to allow progress.)
2. Use two CASLocks, one to protect head and on to protect the tail.

Compare the two algorithms on the Sun Fire machine. Run an execution time benchmark in which each thread executes 100000 enqueue or dequeue operations (use two types of threads: enqueuer threads and dequeuer threads). Run the benchmarks for 2,4 and 8 threads (one half enqueuer threads and one half dequeuer threads). For 2 threads run also a benchmark on the original 2-thread lock-free queue. Report the average of 3 runs in a file named E2.txt. Write also in the same file what can you conclude regarding the performance of your implementations.

Guidelines

Suggestion: For a more elegant implementation, you can define a queue interface, that you can implement for the three different queue types (lock-free, with one lock and with two locks).

The algorithm for the 2-thread lock-free queue (presented in the course) for Exercise 2:

```
public class MyQueue {
  int head = 0, tail = 0;
  int items[] = new int[QSIZE];

  public void enq(int x) {
    while (tail-head == QSIZE);
    items[tail % QSIZE] = x; tail++;
  }
  public int deq() {
     while (tail == head);
     int item = items[head % QSIZE]; head++;
     return item;
  }
}
```