Multicore Semantics & Programming

Exercise sheet (Tim Harris' section)

Each question carries equal marks, for a total of 25% of the course total

Core exercises (Part II and ACS/Part III)

1. Lock implementation

A machine has 4 processors, each with its own cache. A program runs with 1 thread per processor, and 1 mutual exclusion lock used by all of the threads. Each thread must acquire the lock, execute a short critical section, and then release the lock.

For each of the following locks (i) describe the cache line transfers that occur in a simple MESI cache protocol in the best-case execution of the lock, and (ii) describe any additional cache line transfers that may occur if execution does not follow the best case.

- Test-and-set lock.
- Test-and-test-and-set lock.
- MCS lock.

You may assume that, by chance, the thread on processor 1 successfully acquires the lock first, then processors 2, 3, and 4. Also, you may assume that each QNode structure for the MCS lock is a single cache line in size.

2. Linearizability

Consider the following history of operations on a set implemented over a linked list. The set is initially empty. A call to insert(X) returns true if it succeeds in adding X to the set. A delete_ge(X) operation deletes the next value above or equal to X. It returns the value deleted, or -1 if there is no such value.

```
Thread 1 : Calls delete_ge(10)
                                Thread 2 : Calls insert(30)
                                Thread 2 : insert(30) returns true
                                                                Thread 3 : Calls insert(20)
                                                                Thread 3 : insert(20) returns true
                                                                                                Thread 4 : Calls insert(30)
                                                                                                Thread 4 : insert(30) returns false
Thread 1 : delete_ge(10) returns 30
```

Show that this concurrent history *is not* linearizable. Then, for each of the following alternatives, show that the resulting history *would be* linearizable:

(i) If the delete_ge(10) operation had returned -1 to thread 1.
(ii) If the delete_ge(10) operation could delete any value greater than or equal to 10 in the set.
(iii) If the operations in thread 4 executed before those in thread 3.

3. Lock-free lists and memory management

Consider a lock-free linked list of integers, held in sorted order and shared between a large number of threads. Threads perform search, insert, and delete operations on the list.

Initially, assume that a garbage collector is used to reclaim storage automatically. Describe workloads (i) where the lock-free list is likely to perform better than a list protected by a well-implemented mutual exclusion lock, and (ii) where the lock-free list is likely to perform less well than the lock-based list.

Suppose that a per-list-node reference counting scheme is used instead of garbage collection in the lock-free list. Are there now any cases where the lock-free list would be preferable? (Note that reference counting *would not* be needed in the lock-based list.)

Additional exercises (ACS/Part III only)

4. Transactional memory

Transactional memory implementations are often classified as making eager or lazy updates and performing eager or lazy conflict detection.

Describe two workloads, one of which would perform well under eager-eager, and one which would perform well under lazy-lazy. Justify your answer in terms of (i) the series of reads and writes that are being attempted within the transactions, (ii) the amount of work executing the transactions initially, (iii) the amount of additional work attempting to commit the transactions, and (iv) the amount of additional work caused by transactional re-execution.

5. Alternatives

An enthusiastic researcher writes that “In the future all shared memory data structures will be lock-free because they are fast and scalable”. With the aid of example data structures, and possible uses of these data structures, describe three cases in which you would agree with using lock-free data structures, and three cases where you would suggest using locking or using transactional memory instead.