MPhil course in Multicore Programming (R204)

Example exercise sheet (Tim Harris' section)

1. In the slides, the pseudo-code for the MCS "acquireMCS" operation shows that the new QNode is only linked into the queue after performing a CAS operation. This makes the "releaseMCS" operation more complicated because a call to "releaseMCS" might need to wait until it sees that a lock holder has linked its QNode into the queue. Show why it would be *incorrect* to optimize the "acquireMCS" operation by initializing prev->next to point to the new QNode before performing the CAS (that is, moving the line at Label 2 to occur earlier at Label 1).

   [2 marks]

2. A "ticket lock" is implemented using two shared counters, T and C, both initially 0. A thread wanting to acquire the lock uses an atomic fetch-and-add on T to obtain a unique sequence number. The thread then waits until C is equal to this sequence number. After releasing the lock, a thread increments C.

   What are the advantages / disadvantages of this ticket lock compared with (i) a test-and-test-and-set lock, and (ii) compared with the MCS queue lock?

   [3 marks each]

3. Some programmers misuse the phrase "lock-free" informally to mean that an algorithm is fast and scalable, even if it does not provide the lock-free progress property. Describe a workload where the singly-linked list in the slides will *not* be fast and scalable, but a normal lock-based list could be better.

   [3 marks]

4. Consider a simple shared counter that supports an "Increment" operation. Each increment advances the counter's value by 1 and returns the counter's new value – 1, 2, 3, etc.

   a) Explain whether or not the following history is linearizable:

   - Time 0 : Thread 1 invokes Increment
   - Time 10 : Thread 1 receives response 1
   - Time 11 : Thread 1 invokes Increment
   - Time 20 : Thread 2 invokes Increment
   - Time 21 : Thread 1 receives response 3
   - Time 22 : Thread 1 invokes Increment
   - Time 30 : Thread 2 receives response 2
   - Time 31 : Thread 1 receives response 4

   [2 marks]

b) In pseudo-code, give a lock-free implementation of “Increment” using an atomic compare and swap operation.

[1 mark]

c) Explain whether or not your implementation is also wait-free.

[2 mark]

[Optional: if your counter is not wait-free, then can you see a way to build a wait-free one from compare and swap, or can you see how to write a proof-sketch that it is impossible to build one?]

5. The array-based deque in the slides supports one thread on the “top” and, and multiple threads stealing from the “bottom” end (slide 59).

   Consider instead the case of a simpler array-based queue supporting a fixed maximum number of elements in the queue at any one time (N) and only a single producer (calling “pushTop”) and a single consumer (calling “popBottom”). A push should return “true” if it succeeds (adding the item to the queue), and “false” otherwise (if the queue is full). A pop should return a data item if there is one in the queue, or NULL if the queue is empty.

   In pseudo-code, give a lock-free linearizible implementation of this queue building on atomic compare and swap, read, and write.

   [4 marks]