Problem Set 6, Part b

Due: Thursday, May 8, 2008

Reading:

Chapter 9 from the Herlihy, Shavit book. (See course web site.)
Herlihy, Luchango, Moir, and Schere paper on Software Transactional Memory.
Dice, Shalev, and Shavit paper on Transactional Locking.

Reading for next week:

Chapter 17. Lamport's "Part-Time Parliament" paper.

Problems:

1. Prove that the hand-over-hand locking algorithm for a list-based set guarantees atomicity and does not deadlock.

2. Suppose we change the hand-over-hand algorithm to use reader/writer locks so that an operation always acquires a lock for reading before reading a node (even if only to read the key), and then when it finds a pair of nodes *pred* and *curr* such that $pred.next = curr$ and $pred.key < x \leq curr.key$, where x is the key passed to the operation, it "upgrades" the lock for writing. Prove that this algorithm is deadlock-free or give a counterexample execution.

3. Given a state in which some *contains* operation has been invoked but not yet returned, we say that the *contains* operation is *determined* if there is only one possible return value. That is, in all execution fragments starting from the state, the operation returns the same value.
   Give an execution for the lock-free list algorithm with wait-free contains in which there is no point at which a *contains* operation can be serialized such that the operation is determined at that point. Can you do this for a *contains* operation that returns true? Can you do this for the lock-based lazy list algorithm?

4. A double-ended queue (deque) is a data type that maintains a sequence of items, initially empty, and provides four kinds of operations: *pushL*, *popL*, *pushR*, and *popR*. A $pushL(x)$ appends the item to the left of the sequence, and $popL()$ removes and returns the leftmost item in the sequence. If the sequence is empty, $popL()$ returns a special null value. *pushR* and *popR* are operate analogously on the right side of the sequence.
   Given a language that provides obstruction-free atomic blocks (as described in lecture), give an obstruction-free implementation of a deque. Your algorithm should provide "disjoint-access parallelism" in the following sense: when the deque has at least three elements, then two operations that operate on different sides of the deque should not access any locations in common (assuming there are no other operations concurrent with them). Prove that your algorithm is correct.

5. For the obstruction-free software transactional memory with invisible readers, give an execution in which a committed transaction cannot be serialized at the point that it changes its status from "active" to "commit".