Name: Abburi Venkata Sai Mahesh Roll No.: CS18BTECH11001

Quiz2
CS5300: Parallel and Concurrent Programming Autumn 2020

Instructions for the exam:

- You must submit your final answer copy as pdf.
- You should avoid submitting scan copies of hand-written notes. Only if you wish to attach any figure, you can attach the scans of the figures in your pdf.

Q1 (5 points). Consider ALock algorithm given in the book. What is the effect of swapping lines lines 23 & 24.
A. If the lines 23 & 24 are swapped, it may lead to a failure in mutual exclusion. In the algorithm, line 23 is setting its slot to false, which means it is clearing its slot and then in line 24 it is informing the next thread to proceed. Let us consider the scenario when the lines are swapped. Now let us consider some threads in which thread A has completed critical section and calling unlock() function. Let thread A has completed line 23(after swapping - notifing for the next slot) and a context switch to other threads happened before executing line 24. If some other thread is assigned to same slot before again context switch to thread A, then it automatically proceed to critical section(as the slot isn't set to false) leading to fail of mutual exclusion.

Q2 (5 points). In CLH lock, what happens if line 30 is changed as follows: myNode.set(qnode)?
A. It leads to a deadlock. Let us consider 2 threads trying to enter into the critical section. Let us consider the implementation with the following lines.

```
1. public void lock() {
2.     QNode qnode = myNode.get();
3.     qnode.locked = true;
4.     QNode pred = tail.getAndSet(qnode);
5.     myPred.set(pred);
6.     while (pred.locked) {}
7. }
8. public void unlock() {
9.     QNode qnode = myNode.get();
10.    qnode.locked = false;
11.    myNode.set(qnode);
12. }
```

Let us assume thread A completed line 6 and entered critical section and the thread B looping in line 6 after setting the pred of thread B to node A. Now the thread A completed critical section and calling unlock method. Then if thread A after calling unlock method and setting the myNode as itself and enter the lock section again, the pred of thread A will be set to node B and as the pred of thread B has set to node A(before). So each thread will be pointing other thread and both will end up spinning in the line 6 and thus ending up with deadlock. The probability of deadlock increases as the number of threads attempting to critical section increases.

Q3 (8 points). Consider the concurrent set implementation based on fine-grained locking on linked-lists that we discussed in the class. Now consider the following LPs for the add and remove methods:

- successful add method returning true: L19 - 'node.next = curr;'
- successful remove method returning true: L46 - 'return true;'

Can these statements be correct LPs for these methods. Please justify your answers.

A. Yes both these statements can be Linearizable points for these methods.

1. <u>add() method:</u>

As the add() method is happening withing the lock which protects the concurrent access to both the predecessor and node of interest, the thread that is trying to interfere with these values must wait for the lock. As these operations or of other nodes can be ordered in any order, we can consider any line within the acquisition of predecessor and concurrent locks as linearisable point. So the given statement is true.

2. <u>remove() method:</u>

As this method happens just after deleting the node and because any concurrent access to predecessor and current node would acquire the lock to both of them we can linearise any line with the acquisition of lock.

Q4 (5 points). Please explain how the contains() method in lazy-list is wait-free.

A. Let us consider the contain method given in the book:

```
1.  public boolean contains(int key) {   // lazy list's contains() method
2.      Entry curr = this.head;
3.      while (curr.key < key)
4.              curr = curr.next;
5.      return curr.key == key && !curr.marked;
6.  }
```

We know that the universe of keys is well-founded there are only a finite number of keys that are smaller than the one being searched for. From the algorithm we can see that entries with lower or equal keys to a given entry will never be added ahead of it and each time the traversal moves to a new node which can be happen

for only a finite number of times. And thus the contains() method traverses the list once ignoring locks and returns true if the node it was searching for is present and unmarked, and false

Q5 (5 points). Show a scenario in the optimistic algorithm where a thread is forever attempting to delete a node. Note, this question is exercise 104 of the book (page 220 in the revised 1st edition). Please see the hint given there.
A. If we want to delete node forever, the validate method should return true continuously. As the validate method() traverses through the whole list, if we could find any processes which can insert nodes between pred and curr every time(maintaining at-least one node in order to prevent emptying the list), it can be the scenario, where the thread can try forever to delete a node.