EE 382C/EE 361C Multicore Computing Fall 2018

Assignment 3

Due: October 16

Instructor: Professor Vijay K. Garg (email: garg@ece.utexas.edu)

1. **(10 points)** For each of the histories below, state whether it is (a) sequentially consistent, (b) linearizable. Justify your answer. All variables are initially zero.

```
Concurrent History H1

P1           [ read(x) returns   1]
P2       [ write(x,1)                    ]    [ read(x)  returns 2]
P3             [write(x,2)      ]

Concurrent History H2

P1           [ read(x) returns   1]
P2       [ write(x,1)                    ]    [ read(x)  returns 1]
P3             [write(x,2)      ]

Concurrent History H3

P1           [ read(x) returns 1]
P2       [ write(x,1) ]                        [ read(x)  returns 1]
P3                         [write(x,2)      ]
```

2. **(5 points)** Consider the following concurrent program.

```
Initially a, b and c are 0.
P1:  a:=1 ; print(b) ; print(c);
P2:  b:=1 ; print(a) ; print(c);
P3:  c:=1 ; print(a) ; print(b);
```

   Which of the outputs are sequentially consistent. Justify your answer.

   (a) P1 outputs 11, P2 outputs 01 and P3 outputs 11.

   (b) P1 outputs 00, P2 outputs 11 and P3 outputs 01.

3. **(5 points)** Suppose that an array does not have all elements that are all distinct. Show how you can use any algorithm that assumes distinct elements for computing the maximum to solve the problem when elements are not distinct.

4. **(20 points)** Give a parallel algorithm on a CREW PRAM with time complexity $O(\log n)$ and work complexity $O(n)$ to compute the inclusive parallel prefix sum of an array of size n by combining two algorithms: sequential prefix sum that takes $O(n)$ time and $O(n)$ work and a non-optimal parallel prefix algorithm that takes $O(\log n)$ time and $O(n \log n)$ work.

5. **(30 points, programming)** The goal is to develop a concurrent implementation of a sorted linked list of integers. that uses n threads, where $n = 1, 2, 4, 8$. The linked list provides the following methods: `boolean add(int x), boolean remove(int x), boolean contains(int x)`. The method `add` returns true if x was not in the list and was added to the list; otherwise, it returns false (and does not add it multiple times). The method `remove` returns true if x was in the list and was removed from the list; otherwise, it returns false. The method `contains` returns true if x is in the list.

Implement the following schemes:
(a, 5 points) Coarse-grained Locking
(b, 10 points) Fine-grained Locking
(c, 15 points) Lock-Free Synchronization.

6. **(30 points, programming)** (a, 20 points) Implement Lock-based and Lock-Free unbounded queues of `Integers`. For the lock based implementation, use different locks for `enq` and `deq` operations. For the variable `count` use `AtomicInteger`. For the lock-free implementation, use Michael and Scott's algorithm as explained in the class. The *deq* operation should return *null* if the queue is empty.

   (b, 10 points) Implement Lock-Free stack of `Integer`. You should provide `push(Integer x)` and `Integer pop()`. The `pop` operation should throw an exception called `EmptyStack` if the stack is empty.
   For both the data structures use a list based implementation (rather than an array based implementation).