Theory Assignment 4: Locks
CO21BTECH11004

Que 7.1)
Sol:

Deadlock can occur with this implementation.

- Say thread A comes and acquires the lock. So, Tail points to qnode of A, and A acquires the lock. (A.lock)
- Now, thread B comes and wants to acquire the lock. So, the tail points to the qnode of B, and B spins on pred = A's qnode. (B.lock)
- Thread A releases the lock, so qnode of A becomes false. (A.unlock)
- Before thread B, see Thread A has released the lock, thread A again enters and tries to acquire the lock. (A.lock)
- Now thread A qnode is true, it spins on thread B's qnode, which is also true.
- For thread A, it sees thread B is acquiring the lock, same for thread B, it sees thread A has acquired the lock.
- Deadlock.

MCS lock unlock()

```
public void unlock() {
  QNode qnode = myNode.get();
  if (qnode.next == null) {
    if (tail.compareAndSet(qnode, null))
      return;
    // wait until successor fills in its next field
    while (qnode.next == null) {}
  }
  qnode.next.locked = false;
  qnode.next = null;
}
```

- This problem will not arise in MCS lock, as while unlocking, it will set locked variable of the next element in the list to true.
- Here, when A will unlock, it will set locked variable of B to false, and then leave.
- So, B would acquire lock.

Que 8.6)
Sol:

```
public class sharedBathromm{
     int numMale, numFemale;
     int numMaleWR, numFemaleWR;
     Lock lock;
     Condition condition;
     // whoNext, 0:- male, 1:- female
     bool whoNext;
public:
     sharedBathroom(){
         numMale = numFemale = 0;
          numMaleMR = numFemaleWR = 0;
         whoNext = 0;
         lock = new ReenterantLock();
         condition = lock.newCondition();
     }
     void enterMale() {
         lock.lock();
              while( numFemale>0 && whoNext)
condition.await();
              numMale++;
          lock.unlock();
     }
     void enterFemale() {
         lock.lock();
              while( numMale>0 && !(whoNext))
condition.await();
              numFemale++;
          lock.unlock();
     }
```

```
    void leaveMale() {
        lock.lock();
            numMale--;
            whoNext = 1;
            condition.signalAll();
         lock.unlock();
    }
    void leaveFemale() {
        lock.lock();
            numFemale--;
            whoNext = 0;
            condition.signalAll();
         lock.unlock();
    }
}
```

My implementation satisfies mutual exclusion and weak starvation-freedom.

- Mutual Exclusion:- In the enterMale or enterFemale function, it is checked if there are other sex people inside or not, then only lock is acquired. This check is done by maintaining the count of males and females. Two possible cases: if males are inside, then numMale>0 && numFemale = 0, else if females are inside, then numFemale>0 && numMale = 0.
- Starvation Freedom:- For starvation freedom, we introduce a bool flag turn. So, whenever a person leaves it set turn to other sex people. So, say turn = false means male, and turn = true means female. When maleLeave() is called, it will set turn to true, giving a chance to females after all males currently in the gym exit. So then, in an empty gym, if a male and female compete, the female wins and enters the gym.

Que 3)
Sol:

```
public boolean tryLock(long time, TimeUnit unit) throws
InterruptedException{
     long startTime = System.currentTimeMillis();
     long patience = TimeUnit.MILLISECONDS.convert(time,
unit);
     QNode qnode = myNode.get();
     QNode pred = tail.getAndSet(qnode);
     if (pred==NULL) return true;
     qnode.locked = true;
     pred.next = qnode;
     qnode.pred = pred;
     while (System.currentTimeMillis() - startTime <
patience)
     {
     if (!qnode.locked)
          return true;
     }
     qnode.pred.next = qnode.next;
     if (qnode.next != null) {
     qnode.next.pred = qnode.pred;
     }
     return false;
}
```

This code is a modification of the code given in book for tryLock using CLH lock and MCS lock. Full class is not written only tryLock function is written above, as else would be same as MCS.

Using a doubly linked list for the implementation of trylock using MCS.

Only the lock function is changed, unlock is similar to MCS lock, only maintaining the doubly linked list.
When a thread is abandoned, we update the thread's node predecessor next to the thread's node next. If the thread's node next is not null, update the thread's next node predecessor to the thread's node predecessor.