NAME: Xiang Xu

**605.741.31**
**Module 7 Quiz**

1) Concurrency Control.

   a) A park rents rowboats online. Part of the online rental process requires each renter to reserve two oars, the number of lifejackets that the renter will need, and a boat before placing the reservation. Write a single pseudo-code algorithm that enables people to run their own software clients to rent and return boats. Use the two-phase locking protocol and make sure to avoid deadlocks! Explain why your algorithm avoids deadlocks.

   Assume that the locking command is *lock(resource, number)* where *resource* is the item that the customer wishes to lock and *number* is the amount of items to be locked. Assume the results of a call to the *lock* command are either SUCCESS or FAIL. Assume that the unlocking command is *unlock(resource, number)*, the command to reserve a boat is *reserveBoat()*, and the command to ride a boat is *rideBoat()*.

```
Class renter{
Timestamp = gettime()
Oarnum = num1
Lifenum = num2
Boatnum = num3

While(IsTimeStampMax(Timestamp)){wait()}

If(lock(oars,Oarnum)) {
    If(lock(lifejackets,Lifenum) ){
      If(lock(boat,Boatnum)){
reserveBoat()
TimestampReset()
rideBoat()
} else soldout()}else soldout() } else soldout()

unlock(oars,Oarnum)
unlock(lifejackets,LifeNum)
unlock(boat,Boatnum)
}
```

This pseudo-code algorithm will avoid deadlocks, because for each renter, I'll give him a timestamp, and I only give him a lock when his timestamp is the oldest. And after they finished the reservation, we will reset the timestamp allowing the second largest to process.

b) Write a pseudo-code algorithm for both the *lock(resource, number)* and *unlock(resource, number)* methods.

```
Lock(resource, number){
If( resource == “oar” && number >=2) {
        If(global.oarcount – number >=0){
                global.oarcount = global.oarcount – number
                }
                Else return false

If( resource == “lifejackets”) {
        If(global.lifejacketcount – number >=0){
                global.lieftjacketount = global.lifejacketcount – number
                }
                Else {  //reset oarcount if lifejacket not enough
                        global.oarcount  = global.oarcount + local.Oarnum
                        return false}


If( resource == “Boat” && number >=1) {
        If(global.Boatcount – number >=0){
                global.Boatcount = global.Boatcount – number
                }
                Else{ //reset oarcount and lifejacketcount if Boat not enough
                global.oarcount  = global.oarcount + local.Oarnum
                global.lifejacketcount  = global.lifejacketcount + local.lifejacketnum
                } return false
Return true
}
```

```
unlock(resource, number){
If( resource == “oar”) {
                global.oarcount = global.oarcount + number

If( resource == “lifejackets”) {
                global.lieftjacketount = global.lifejacketcount + number

If( resource == “Boat”) {
                global.Boatcount = global.Boatcount + number

}
```

2) In the following transactions, assume that a timestamp-based concurrency control algorithm is used to manage writing values to a database. Determine the value of x **in the database** and its read and write timestamps (rts and wts) at each stage of processing. Fill in the values of x, rts(x), and wts(x) in the arrays below. Indicate any special situation such as an Abort or the use of the Thomas Write Rule.

a) T1 has a timestamp of 40 and T2 has a timestamp of 39.

| **T1: TS=40** | **T2: TS=39** | **x** | **rts(x)** | **wts(x)** | **special situation** |
|---|---|---|---|---|---|
| | | 10 | 38 | 38 | |
| read(x) | | 10 | 40 | 38 | |
| | x = 50 | 10,50 | 40 | 38 | |
| x = x*2 | | 20,50 | 40 | 38 | |
| | write(x) | 20,50 | 40 | 38 | Abort |
| write(x) | | 20 | 40 | 40 | |

b) T1 has a timestamp of 39 and T2 has a timestamp of 40.

| **T1: TS=39** | **T2: TS=40** | **x** | **rts(x)** | **wts(x)** | **special situation** |
|---|---|---|---|---|---|
| | | 10 | 38 | 38 | |
| read(x) | | 10 | 39 | 38 | |
| | x = 50 | 10,50 | 39 | 38 | |
| x = x*2 | | 20,50 | 39 | 38 | |
| | write(x) | 20,50 | 39 | 40 | |
| write(x) | | 50 | 39 | 40 | Thoma's Write |