1. Fill in the last four rows of values in the middle three columns for the following trace table that reflects the lost update example in the slides. Each thread has its own copy of register r1 and is able to access the shared global variable x using load and store instructions. ('?' means unknown value.)

**time thread A A's r1 x B's r1 thread B**

0 ? 0 ? (initially)

1 load r1,x 0 🡨 0 ?

2 add r1,r1,#1 1 0 ?

3 \_1\_ \_0\_ 🡪 \_0\_ load r1,x // B’s r1 = memory[x];

4 \_1\_ \_0\_ \_1\_ add r1,r1,#1 // B’s r1++;

6 \_1\_ \_1\_ 🡨 \_1\_ store r1,x // memory[x] = B’s r1;

5 store r1,x \_1\_ 🡪 \_1\_ \_1\_

2. Assume that you have a “Lock” data type available as discussed in the textbook. Add the appropriate global declaration for a lock and the necessary lock operations to each of the three threads so that the shared counter x is updated with mutual exclusion.

// global data

int x = 0;

Lock my\_lock;

// ---- thread 1 ---- // ---- thread 2 ---- // ---- thread 3 ----

int local; int local; int local;

my\_lock.acquire(); my\_lock.acquire(); my\_lock.acquire();

local = x; local = x; local = x;

local = local + 1; local = local + 1; local = local + 1;

x = local; x = local; x = local;

my\_lock.release(); my\_lock.release(); my\_lock.release();

thread\_exit(0); thread\_exit(0); thread\_exit(0);

3. In many cases, threads must wait until a condition becomes true, e.g., you must wait until at least one item is present in a buffer before being able to remove an item. This is the C++ code for the Blocking Bounded Queue class from the textbook. (BBQ::BBQ() is the constructor that initializes front and nextEmpty.)

int BBQ::remove() {

int item;

lock.acquire();

while (front == nextEmpty) {

ItemAdded.wait(&lock);

}

item = items[front % MAX];

front++;

ItemRemoved.signal();

lock.release();

return item;

}

class BBQ{

1. Lock lock;
2. CV ItemAdded;
3. CV ItemRemoved;
4. int items[MAX];
5. int front;

(B) int nextEmpty;

public:

BBQ();

~BBQ() {};

void insert(int item);

int remove();

}

BBQ::BBQ(){

front = nextEmpty = 0;

}

void BBQ::insert(int item) {

lock.acquire();

while ((nextEmpty - front) == MAX) {

ItemRemoved.wait(&lock);

}

items[nextEmpty % MAX] = item;

nextEmpty++;

ItemAdded.signal();

lock.release();

}

(a) Circle the synchronization variables and add an identifying label “a”.

(b) Circle the state variables and add an identifying label “b”.

(c) Apart from the constructor, are the accesses to the bounded queue state variables synchronized with locks and condition variables?

Yes

(d) Is the lock acquired at the beginning of each method and released right before the return?

Yes

(e) Is the lock held whenever a condition variable operation (wait or signal) is called?

Yes

(f) Are all wait operations done within a loop?

Yes

(g) Is thread\_sleep() used anywhere?

No