CSCE 313 Quiz 4 Fall 2021

Christopher McGregor – 726009537

Q1. To change the given code for an unbounded buffer, we simply need to remove the wait for ‘emptySlots’. Since we have no bounds, we do not have a capacity. However, the other calls are still needed. For instance, to main thread safety we still need to use the mutex, and to notify the consumer we will use the ‘fullSlots’. More succinctly the change would be as follows:

```
Producer(item) {
    mutex.P();
    Enqueue(item);
    mutex.V();
    fullSlots.V();
}
```

Q2. BoundedBuffer.h :

```
queue<int> list;
int capacity;
condition_variable cv;
mutex m;
void push(int data) {
        m.lock();
        list.push_back(data);
        m.unlock();
        cv.notify_one();
}
void pop() {
        unique_lock<mutex> ul (m);
        cv.wait (ul, [this]{return list.size() > (capacity * .1)});
        int data = list.front();
        list.pop();
        ul.unlock();
}
```

Q3. We must use a common mutex between the producer and consumer to eliminate a race condition which would otherwise occur when accessing the list.

Q4. Load r1, x
Load r2, x
Load r3, x
Add r3, r3, 2
Store x, r3 -> here x store will be lost
Add r2, r2, 2
Store x, r2 -> here x store will be lost
Add r1, r1, x
Store x, r2 -> x now equals 2
Load r4, x
Load r5, x
Add r5, r5, 2
Store x, r5 -> here x store will be lost
Add r4, r4, x
Store x, r4 -> x = 4!

Q5. We can have up to 3 threads in the critical section. The first threadA will enter the critical section and lock the mutex. Since threadB is buggy it will unlock the mutex and also enter the critical section. Since threadB just unlocked, another threadA can now enter the critical section making the total 3.

Q6. Honestly, I couldn't get the given code to compile so I'm not 100% confident in my solution. That being said, the question seems very straight forward and thus my solution handles multiple producer signaling much like the consumer does. Q6.cpp and Semaphore.h attached in zip

Q7. Q7.cpp attached in zip