Name: Arbin Bhuiyan UIN:426005391

Score: out of Total 90

**Question 1 [10 pts]:** The following is the Producer(.) function in a BoundedBuffer implementation. What is the purpose of the mutex in the following? Can we do without the mutex? In what circumstances?

Producer(item) {

     emptySlots.P();

     mutex.P();

     Enqueue(item);

     mutex.V();

     fullSlots.V();

}

The mutex is used in this particular instance to ensure that no other producer/consumer thread accesses the queue when a particular Producer thread is accessing it. We can do this without a mutex by either running the threads sequentially or using conditional variables which would ensure that under no circumstance a race condition exists for the queue.

**Question 2 [10 pts]:** The following is the Producer(.) function in a BoundedBuffer implementation. Can we change the order of the first 2 lines? Why or why not?

Producer(item) {

     emptySlots.P();

     mutex.P();

     Enqueue(item);

     mutex.V();

     fullSlots.V();

}

We cannot change the order of the semaphore and the mutex because it would cause emptySlots to never be unlocked and thus a deadlock condition would occur.

|  |
| --- |
|  |
| **Question 3 [10 pts]:** If we run 5 instances of ThreadA() and 1 instance of ThreadB(), what can be the maximum number of threads active simultaneously in the Critical Section? The mutex is initially unlocked. Note that ThreadB() is buggy and mistakenly unlocks the mutex first instead of locking first. Explain your answer.   |  |  | | --- | --- | | **ThreadA(){        mutex.P()        /\* Start Critical Section \*/        …….        /\* End Critical Section \*/        mutex.V(); }** | **ThreadB(){    mutex.V()    /\* Start Critical Section \*/        …….    /\* End Critical Section \*/    mutex.P(); }** | |
|  |

Only three active threads can run the respective critical sections simultaneously (2 ThreadA’s and 1 ThreadB). Because the mutex is initially locked, the first Thread A will lock the mutex. Because the mutex is locked, Thread B can go ahead and unlock the lock and enter it’s critical section. Consequently, a second Thread A can go ahead and lock the mutex, thus resulting in three active threads in their respective critical sections.

**Question 5 [25 pts]:** Consider a Producer-Consumer problem where **M Producer** threads run simultaneously to fill up a buffer. Then, exactly **N Consumer** threads will run simultaneously to take all the data out from the buffer.  An example of such scenario is a file is striped into **M** different site. A downloader program starts M producer threads to get these chunks independently. Once M download threads (from a pool of threads) finish producing the items, the N Consumer threads simultaneously scan the data in their own way. Again, note that N out of many possible consumers succeed. Therefore, you cannot assume that only M producers and N consumers are presents. Look at the given program **1PNC.cpp** that works for 1 Producer and N Consumer threads. Be sure to run the program first to see how it behaves. You need to extend the program such that it works for all M producers instead of just 1.

Add necessary semaphores to the program. However, you will lose points if you add unnecessary Semaphore. To keep things simple, declare the mutexes as semaphores as well. Then test your program to make sure that it is correct. In your submission directory, include a file called **Q5.cpp** that contains the correct program.

**Question 6 [15 pts]:** There are 3 sets of threads A, B, C. First 1 instance of A has to run, then 2 instances of B and then 1 instance of C, then the cycle repeats. This emulates a chain of producer-consumer relationship that we learned in class, but between multiple pairs of threads. Write code to run these set of threads.

Assumptions and Instructions: There are 100s of A, B, C threads trying to run. Write only the thread functions with proper wait and signal operation in terms of semaphores. You can use the necessary number of semaphores as long as you declare them in global and initialize them properly with correct values. The actual operations done by A, B and C does not really matter. Submit a separate C++ file called **Q6.cpp** that includes the solution.

**Question 7 (20 pts):** Implement a Mutex using the atomic swap(variable, register) instruction in x86. Your mutex can use busy-spin. But assume that you cannot access a register directly other than using this swap instruction. (Note in some lecture, we solved this problem with direct access to the register allowed. For instance, you could simply write $register = 1. But you are not allowed to do that directly. You must do the following: int x = 1; swap (x, $reg).

Int value = 0;

Acquire()

Do {

Int old = value

Swap(1, &reg)

if(old == 0)

break

} while(old == 1)

Release() {

value = 0;

}