CSC 139: Operating System Principles
Midterm Exam, Spring 2017
Friday, March 17, 2017
Section 1

Instructor: Ghassan Shobaki

Student Name: _____

Student Number: _____

Question	Points	Score
1	50	
2	50	
Total	100	

Question 1: Short Conceptual Questions [50 points]

1. What's the maximum number of processes (or threads) that are allowed to be executing inside a **monitor** at the same time? What's the purpose of this limit? (10 points)

2. The length of the time quantum (slice) given by the operating system scheduler to each process has to be selected carefully. What's the negative consequence of making this time slice too long and what's the negative consequence of making it too short? (10 points)
Too long is bad, because

Too short is bad, because

3. Some operating systems, such as Linux, use both **spinlocks** and **disabling kernel preemption** for locking in the kernel**.** Which of these two methods works better on a **single processor** system and which one works better on a **multi-processor** system**?** Why? (10 points)
On a single processor, is a better choice, because

On a multi-processor, is a better choice, because

In Questions 4 through 7 below, circle the right answer. There is only one right answer.

4. Which of the following is **<u>not</u>** true about message naming? (5 points)
a. In direct communication, the sending process specifies the receiving process (usually by ID).
b. In direct communication, multiple links may exist between a pair of processes.
c. In indirect communication, the sending process specifies a mailbox rather than a process.
d. In indirect communication a link may be associated with more than two processes.

5. Which of the following is **<u>not</u>** true about Remote Procedure Calls (RPCs)? (5 points)
a. Parameter marshalling involves packaging function parameters in a form that can be transmitted over a network.
b. Parameter marshalling is done by the kernel.
c. The matchmaker takes an RPC name and returns the corresponding port number.
d. The user program communicates directly with the matchmaker without kernel intervention.

6. Which of the following is **<u>not</u>** true about shared memory and message passing? (5 points)
a. In shared memory, kernel intervention is needed only to setup the shared memory block.
b. In message passing, every send/receive goes through the kernel.
c. Shared memory is slower than message passing on all multiprocessor systems.
d. Shared memory is faster than message passing on all single-processor systems.

7. How does protecting a critical section (CS) with a semaphore ensure mutual exclusion? (5 points)
a. When a process is in its CS, it never loses the CPU until it completes the CS.
b. When a process is in its CS, all other processes are placed in the waiting state.
c. When a process is in its CS, no other process is allowed to be in a CS.
d. When a process is in its CS, any process that tries to access a CS that is protected by the same semaphore is placed in the waiting state.
e. Both a and d are correct.
f. Both b and c are correct.

Question 2: Concurrent Processes and Shared Memory [50 points]

Consider the following functions that implement the producer and the consumer in Assignment 1. Assuming that the rest of the code is the same as in the give templates, answer the following questions. Each question is **independent** of other questions.

```
void Producer(int bufSize, int itemCnt, int randSeed){
        int i, in = 0, out = 0, val;

1        for (i=0; i<itemCnt; i++) {
2                while((GetIn()+1)%bufSize == GetOut());
3                val = GetRand(0, 1000);
4                WriteAtBufIndex(in, val);
5                in = (in+1) % bufSize;
6                SetIn(in);
        }
}
void Consumer(){
//Code to open shared memory block and map it to gShmPtr
1    int bufSize = GetBufSize();
2    int itemCnt = GetItemCnt();
3    int in = GetIn();
4    int out = GetOut();
5    for(i=0; i<itemCnt; i++){
6          while(GetIn() == GetOut());
7          val = ReadAtBufIndex(out);
8          out = (out + 1) % bufSize;
9          SetOut(out);
    }
```

1. Although there are four shared variables in the header, the above solution worked correctly without using any semaphores or mutex locks. Why? (4 points)

2. What's the range of possible values that may be returned by **GetIn()** on Line 3 and **GetOut()** on Line 4 of the **Consumer**? Let the buffer size be **m**, the number of items be **n**, **n** > **m**. (10 points)

   Range of all possible values returned by GetIn() on Line 3 in Consumer:

   Range of all possible values returned by GetOut() on Line 4 in Consumer:

3. Use the table below to trace the consequences of making each of the following changes on **Line 2 in the Producer,** assuming that everything else remains the same. Assume that there is a **single** producer and a **single** consumer. Let the buffer size be **m**, the number of items be **n**, **n** > **m**. (12 points)

Change	Will work correctly?	Items Produced	Items Consumed	Explanation
Replace ***GetIn()*** with ***in***	Yes No			
Replace **GetOut()** with ***out***	Yes No			

4. Suppose that there is a single producer and we would like to create multiple consumers to maximize the speed. An item may be consumed by any consumer, and once an item has been consumed by one consumer, the producer can use the same buffer cell to produce a new item. Due to the synchronization, there is a limit on the number of consumers that can actually speedup the execution; using 100 consumers will not necessarily be faster than using 10 consumers. For

each of the three cases in the table below, give the maximum number of consumers that can actually speedup the execution (increasing the number of consumers beyond this maximum number will not give any further speedup). Also, give the corresponding speedup ratio.
In the table, ConsTime is the time for consuming one item and ProdTime is the time for producing one item. The max speedup ratio is the total execution time using one consumer divided by the total execution time using the maximum number of useful consumers. Assume that each consumer or producer process will run on a different CPU with **ideal parallelism**, that is, ignore all kinds of overhead (process creation, communication, synchronization, etc.) (12 points)

Case	Max useful consumers	Max speedup ratio	Brief Explanation
ConsTime = ProdTime			
ConsTime = 3 x ProdTime			
ProdTime = 3 x ConsTime			

5. Modify the above **Producer** and **Consumer** functions to implement the following changes:
   - There is one consumer but multiple producers. The parent process creates and initializes the shared memory block and then forks multiple child processes: one child running the **Consumer** code and **k** children running the **Producer** code (don't worry about the initialization and forking code). Each producer produces itemCnt items. The parent writes to the header the value totalItemCnt, which is equal to **k** times itemCnt (what the consumer needs to consume).
   - The buffer can be filled completely; if the buffer size is **m**, **m** cells can be used, not **m-1**.
   - Busy waiting is **not** allowed; when a process waits because the buffer is full or empty, it cannot waste CPU cycles.

     Write your implementation using the **minimum** number of semaphores and the **minimum** number of lines within the critical sections. You will lose points for any unnecessary semaphore or any line of code that is unnecessarily placed in a critical section. Make sure you show the initial value of every semaphore or shared variable that you use. (12 points)