CSC 139: Operating System Principles
Midterm Exam, Fall 2017
Friday, October 20, 2017
Section 1
Form A

Instructor: Ghassan Shobaki

Student Name: _____

Student Number: _____

Question	Points	Score
1	40	
2	30	
3	30	
Total	100	

Question 1: Short Conceptual Questions [40 points]

A. What OS feature or functionality does each of the following **hardware** features support or speed up? Briefly but clearly explain how. (10 points)

Dual mode of operation

A single instruction that stores all CPU register values into memory

B. In the questions below, circle the right answer. There is only one correct answer. (30 points)

1. Which of the following is true about threads?
   a. All threads within the same process share one stack.
   b. All threads within the same process share global variables.
   c. All threads within the same process must run on the same core.
   d. Context switching between threads is faster than context switching between processes.
   e. Both a and b are correct. f. Both b and d are correct. g. Both c and d are correct.

2. Which of the following is true about process states?
   a. A process can transition directly from the waiting state to the running state.
   b. All waiting processes are placed in one waiting queue.
   c. The CPU scheduler considers only the processes in the ready queue.
   d. If the ready queue is empty, the CPU scheduler may consider some waiting processes.

3. What is the difference between an I/O-bound process and a CPU-bound process?
   a. An I/O-bound process never uses the CPU.
   b. A CPU-bound process never requests I/O.
   c. A CPU-bound process requests I/O less frequently than an I/O-bound process does.
   d. A CPU-bound process has a larger number of CPU bursts than an I/O-bound process has.
   e. Both c and d are correct. f. Both a and b are correct. g. Both b and c are correct.

4. What are the limitations of Amdahl's Law?
   a. You can apply it only when all CPUs are on the same chip not on different chips.
   b. You cannot apply it to a system with more than 8 CPUs
   c. You cannot apply it when the number of processes is greater than 16.
   d. It assumes that the parallelizable code can be divided *equally* among the CPUs.
   e. It does not account for communication and synchronization overhead.
   f. Both a and d are correct. g. Both a and e are correct. h. Both d and e are correct.

5. Which of the following is **not** true about a **Zombie** process?
   a. It has an entry in the process table.
   b. Its parent has terminated without calling wait().
   c. Its parent has not called wait(), but the parent has not terminated yet.
   d. Its resources have been deallocated by the operating system.

6. What is the maximum number of processes/threads that can be active in a **monitor** at the same time?
   a. One b. Two c. Eight
   d. The number is set by the system, depending on the number of hardware threads.
   e. The number is set by the application programmer, depending on the application.
   f. The number is unlimited to maximize parallelism.

Question 2: Concurrent Processes and Shared Memory [30 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 **<u>independent</u>** 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(X1);
3                val = GetRand(0, 1000);
4                WriteAtBufIndex(in, val);
5                in = X2;
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 = X3;
9          SetOut(out);
    }
```

1. Replace X1, X2 and X3 in the above code with the right expressions. (12 points)

   X1:

   X2:

   X3:

2. Use the table below to trace the consequences of making each of the following changes **in the Consumer,** 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 it work correctly?	Items Produced	Items Consumed	Explanation
Replace **GetOut()** on Line 6 of the Consumer with ***out***	Yes No			
Delete ***SetOut()*** on Line 9 of the Consumer	Yes No			

3. In Assignment 1, the shared memory object was mapped into a void pointer named gShmPtr in both the producer and the consumer. Will the code work correctly if we use a different name in each source file (for example, we name it gShmPtr1 in producer.c and gShmPtr2 in consumer.c)? Why? (6 points)

Question 3: Process Synchronization [30 points]

Consider the following solution studied in class for the **Readers-Writers Problem:**

```
        Writer() {
1         wait(X1);
2         perform_writing ();
3         signal(X2);
        }

        Reader() {
1         wait(X3);
2         counter++;
3         if (counter == 1)
4                   wait(X4);
5         signal(sem_2);

6         perform_ reading ();

7         wait(sem_2);
8         counter--;
9         if (counter == 0)
10                  signal(sem_1);
11        signal(sem_2);
        }
```

1. Replace X1, X2, X3 and X4 in the above code with the right semaphore (sem_1 or sem_2).
   X1: (8 points)

   X2:

   X3:

   X4

2. What is the right initial value for each of the following? (2 points)
   sem_2:

   counter:

3. Trace the consequences of replacing signal(sem_2) on Line 5 of the **Reader** with signal(sem_1). Make sure that you cover all the consequences and indicate whether the code will work correctly or not. (8 points)
   Impact on Readers:

   Impact on Writers:

4. Rewrite the above Writer() and Reader() functions to solve the following Readers-Writers Problem.
   - There are multiple writers and multiple readers.
   - Writing and reading happen in a periodic manner: one write, followed by multiple reads, followed by a second write, followed by multiple reads, and so on.
   - Initially, only one writer is allowed to access to the buffer (no readers are allowed).

- After the writer completes writing, X readers are allowed to access the buffer, but a maximum of Y readers can access the buffer at the same time (Y < X).
- The X readers are not necessarily distinct; a reader may access the buffer more than once.
- No writer is allowed to access the buffer until all X readers are done.
- When all X readers are done, a new Write/Read cycle starts. So, no more readers are allowed at that point and only one writer is allowed to access the buffer. This writer could be the same as the first writer or a different writer.
- After the writer completes, X readers are allowed again to access the buffer with a limit of Y simultaneous readers, and the same Write/Read pattern repeats periodically.

Your solution **must** avoid wasting CPU cycles when processes are waiting and must minimize critical section sizes. Your solution must also be as efficient and concise as possible. You will lose points for any unnecessary code, variables or semaphores. Make sure that you specify the initial values of any shared variables or semaphores that you use. (12 points)