Exam-like Questions

1. Implement a binary semaphore using the atomic exchange instruction (XCHG). Give the code for both P(mutex) and V(mutex).

2. Assume that upon a 'csignal' ***c*** action, a process on 'c.queue' (i.e., the monitor queue that holds processes blocked on condition ***c*** could simply be transferred to the urgent queue. Alternatively, a monitor suspends the current process (i.e., adds it to the *urgent queue*), and resumes a process waiting in *c.queue*. Refer to the monitor below:

```
monitor
          int s = 1;
          condition c;
          procedure p(s)
          {
          if (--s < 0)
               c.wait;
          }
          procedure v(s)
          {
          s++;
          c.signal;
          }
   end
```

If the two lines of *procedure v* and *p* were accidentally switched in the implementation, mark one answer in the next two items that best describes what would happen.

a) Assuming the 2nd option: upon a 'csignal' ***c*** action, a process on 'c.queu' is simply transferred to the urgent queue:

(a) This new code could cause a deadlock
(b) This new code would perform the same tasks as the old code
(c) This new code would cause starvation
(d) This new code would cause a violation of critical section mutual exclusion requirements
(e) None of the above

b) Assuming the 1st option: upon a 'csignal(c)' action, a monitor suspends the current process (i.e., adds it to the *urgent queue*), and resumes a process waiting in ``c.queue":

(a) This new code could cause a deadlock
(b) This new code would perform the same tasks as the old code
(c) This new code would cause starvation
(d) This new code would cause a violation of critical section mutual exclusion requirements
(e) None of the above

3. Scheduling During its lifetime a process alternates between CPU bursts and I/O bursts. During the CPU bursts the process executes on the CPU without performing I/O operations. During the I/O bursts the process waits for I/O completions without using the CPU. We propose the following scheduling policy. The scheduler maintains an (effectively) infinite number of queues, Q[0], Q[1], ..., Q[infinity]. We assume that Q[k] has an associated quantum of 2^k milliseconds. When a job requests the CPU, it is placed on queue Q[0]. Whenever the CPU becomes free, a job is dequeued from the lowest non-empty queue and run for that queue's quantum. If the job finishes its CPU burst before the quantum expires, (i.e., voluntarily releasing the CPU, then it is placed back on queue Q[0] when it requests the CPU again, at the start of its next CPU burst. Otherwise, the CPU is preempted, and the job is reenqueued on the next higher queue. Each such context switch (preempting the CPU and reenqueuing the job) takes one millisecond.

i) Is this algorithm subject to starvation? If not, explain why, and if so, suggest a fix.

ii) Define *the convoy effect* as a long CPU burst that causes jobs with a short CPU burst to queue up for a long time. Does this scheduling algorithm cause the so-called *convoy effect*? If not, explain why, and if so, suggest a fix.

iii) Give a simple closed-form expression, either exact or approximate, for the time spent in context switches by a process that executes *k* CPU bursts of *n* milliseconds, each followed by an I/O burst of *m* milliseconds (i.e., total *k* CPU bursts and *k* IO bursts, interweaved).

4. Answer each question in about a sentence or two. You will be penalized for overly long answers.

i) We have discussed a number of techniques for enforcing mutual exclusion, on top of which we have built more sophisticated synchronization primitives (such as semaphores). **List** two techniques that can work for more than two concurrent processes (hardware or software solutions).

ii) List two mutual exclusion techniques that will work in a multiprocessor environment.

5. Answer the following questions:

i) List the 4 conditions for deadlocks.

ii) For the following set of processes:

Process 1	Process 2	Process 1
P(mutex1)	P(mutex1)	P(mutex2)
P(mutex2)	P(mutex2)	P(mutex1)
V(mutex1)	V(mutex2)	V(mutex1)
V(mutex2)	V(mutex1)	V(mutex2)

a) Draw the Resource Allocation Graph for a situation when there is a deadlock. List what processes are in this deadlock.

b) If only processes P1 and P2 existed, would there be a deadlock? Why or why not? Describe the technique used in this set of processes.

6. You have been hired to simulate one of the CU fraternities. Your job is to write a computer program to pair up men and women as they enter a Friday night mixer. Each man and each woman will be represented by one thread. When the man or woman enters the mixer, its thread will call one of two procedures, *man* or *woman*, depending on the thread gender. Each procedure takes a single parameter, *name*, which is just an integer name for the thread. The procedure must wait until there is an available thread of the opposite gender, and must then exchange names with this thread. Each procedure must return the integer name of the thread it paired up with. Men and women may enter the fraternity in any order, and many threads may call the *man* and *woman* procedures simultaneously. It doesn't matter which man is paired up with which woman (CU frats aren't very choosy in this exercise), as long as each pair contains one man and one woman and each gets the other's name. Use semaphores and shared variables to implement the two procedures. Be sure to give initial values for the semaphores and indicate which variables are shared between the threads. There must not be any busy waiting in your solution.