CSC 112—Midterm 1 Answer Sketches

1. Register values have to be saved when interrupts occur, even in uniprogramming systems.

2. If the amount of wait time is very short (for example, if the critical sections are not time consuming), blocking waits have too much overhead (the O.S. will have to be involved). In such cases, busy waits are preferable.

3. Race conditions:
   a. 0—Yes, it is possible to get this value. For example, assume T2 enters its loop after T1 finishes all iterations. When T1 is done, X will be 10. T2 decrements X back to 0.
   b. 10—Yes, it is possible to get this value. Assume that T1 and T2 start their loops almost simultaneously. Then, in T1 we would have a=0 and in T2 we would have b=0. Now assume that T1 is made to wait while T2 finishes all iterations. At this point, T2 will have decremented X to -10. T1 finishes the first iteration by doing a=a+1(=1), followed by X=a(=1). Let T1 finish its remaining 9 iterations. This will place X at 10. Both T1 and T2 are done.
   c. -3—Yes, it is possible to get this value. Assume that T1 gets into its loop first and finishes 7 iterations. At this point, the value of X = 7. Then T2 gets into the first iteration and in the first step gets b=7. Then let T1 finish its remaining iterations. At this point, X = 10. T2 finishes its first iteration by doing b=b-1(=6), and X=b(=6). Let T2 finish its remaining 9 iterations. This will place X at -3. Both T1 and T2 are done.

4. Monitor problem
   class Minotaur
   
   boolean A_finished = false, B_finished = false;
   cond for_A, for_B;

   public synchronized void syncAB()
   {
     A_finished = true;
     signal(for_A);
   }

   public synchronized void syncBA()
   {
     if (A_finished == false)
       wait(for_A);
   }

   public synchronized void syncBC()
   {
     B_finished = true;
     signal(for_B);
   }

   public synchronized void syncCB()
   {
     if (B_finished == false)
       wait(for_B);
   }

5. SJF scheduling
   a. They are in increasing order of processing times.
b. If there wasn’t a longer process in front of a shorter process, the schedule would be identical to one produced by SJF.

c. The waiting time for the longer process is $t$. The waiting time for the shorter process is $t+p_1$.

d. The waiting time for the shorter process is $t$. The waiting time for the longer process is $t+p_2$.

e. For part c) the total waiting time is $t+t+p_1=2t+p_1$. For part d) the total waiting time is $t+t+p_2=2t+p_2$. Since $p_2<p_1$, the waiting time in part d) is less than the waiting for part c). That is, SJF has less waiting time.

f. Q.E.D

6. Not in a safe state. P1 and P5 are stuck.

7. Fork question. A total of 5 processes are involved. One possible answer:

First process
Child process
Child process
Process ending
Parent Process
Process ending
Process ending

8. Semaphore based synchronization:

Semaphore AB, AC, BD, BE, CE, CF, DG, EG, FG; /* All init'ed to 0.*/

<table>
<thead>
<tr>
<th>Process A:</th>
<th>Process B:</th>
<th>Process C:</th>
<th>Process D:</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskA();</td>
<td>down (AB); taskB();</td>
<td>down (AC); taskC();</td>
<td>down (B); taskD();</td>
</tr>
<tr>
<td>up(AB);</td>
<td>up (BD);</td>
<td>up (CE);</td>
<td>up(DG);</td>
</tr>
<tr>
<td>up(AC);</td>
<td>up(BE);</td>
<td>up(CF);</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process E:</th>
<th>Process F:</th>
<th>Process G:</th>
</tr>
</thead>
<tbody>
<tr>
<td>down (BE);</td>
<td>down (CF); taskF();</td>
<td>down (DG);</td>
</tr>
<tr>
<td>down (CE);</td>
<td>up (FG);</td>
<td>down (EG);</td>
</tr>
<tr>
<td>taskE();</td>
<td>up(EG);</td>
<td>down(FG);</td>
</tr>
<tr>
<td>up(EG);</td>
<td></td>
<td>taskG();</td>
</tr>
</tbody>
</table>

9. I/O bound processes benefit because they use very little processor time at any point. However, processor-bound processes do not starve either, because as the time passes each of them will have “used little processor time in the recent past”, and hence get the CPU.