What is your name? (2 points)

There are two sections:
   I. Short Answer ......................... 20 points; (5 points each, 4 problems)
   II. Problems ........................... 78 points; (13 points each, 6 problems)

98 + 2 points for name = 100 points total

This test is worth 25% of your final grade. Use the back of the sheet if you need more room to write, although you shouldn't need to. This test is closed book and closed notes. You have 50 minutes.

I. Short Answer (5 points each, 4 problems)

1. In terms of the 3 states a process can be in, if a process is currently running in a CPU using Round-Robin process scheduling, which state will it be in after a clock interrupt?

   Ready

2. If the mean number of processes in memory for a computer system is 14, what is the mean number of holes?

   7 (50% rule)

3. In what type of architectures must we use message passing rather than TSL (Test & Set Lock)?

   Distributed System with multiple CPUs, each with their own memory.

   2 if one of these 2 is missing.

4. When using the Banker's algorithm for resource allocation, if the system is in an unsafe state, will that always lead to deadlock? Briefly state why or why not.

   No, because processes may not request their total possible resources, and may release some resources before acquiring others.
II. Problems (13 points each, 6 problems)

1. The two boxes below contain a proposed solution to the Producer-Consumer Problem. Assume that full is initialized to 0, and empty is initialized to the number of slots in the buffer.

**Producer Process**

```plaintext
do {
    ... produce an item in nextp
    ... wait(empt); /* block if no empty slots */
    ... wait(mutex);
    ... add nextp to buffer
    ... signal(mutex);
    signal(full); /* increment full slots count */
} while (true);
```

**Consumer Process**

```plaintext
do {
    wait(full); /* ensure there is something to consume */
    wait(mutex);
    ... remove an item from buffer to nextc
    ... signal(mutex);
    signal(empt); /* increment empty slots */
    ... consume the item in nextc
    ... 
} while (true);
```

**a)** Why does this proposed solution use 2 semaphores, empty & full, rather than just one (call it count)?

6 pts. 1. We need to block in 2 different conditions (buffer full for producers, buffer empty for consumers)

2. Semaphores can only block when their value is at 0 and a down is done.

**b)** Is it a problem if we switch the order of the two lines wait(empt) and wait(mutex) in the producer process shown above (see the arrow)? Defend your answer.

7 pts. Yes it is a problem because:

1. Assume the buffer is full, and the wait(mutex) is before wait(empt)
2. Producer does a wait(mutex), then a wait(empt) and goes to sleep on empty
3. Consumer can never empty out buffer, since it can’t get past wait(mutex).
4. Both sleep forever, (deadlock)
2. Consider the traffic deadlock depicted in the figure below, with one way streets indicated by arrows, with traffic blocking the intersections.

![Traffic Deadlock Diagram]

a) State the name of each of the four necessary conditions for deadlock and show that they indeed hold in this example.

1. **No Preemption**
   A vehicle can't force another out of an intersection.

2. **Hold and Wait**
   Vehicles occupy current space and are waiting for the space ahead.

3. **Mutual Exclusion**
   Only one vehicle at a time can be in an intersection.

4. **Circular Wait**
   There is a cycle of blocked intersections.

b) Come up with a method to avoid deadlocks. (Hint: It's the law!)

5 pts.  
Don't enter the intersection unless you can go all the way through.
3. Consider the following snapshot of a system:

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</th>
<th>Need</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₀</td>
<td>1 0 3 2</td>
<td>1 0 5 4</td>
<td>0 0 2 2</td>
</tr>
<tr>
<td>P₁</td>
<td>2 1 1 1</td>
<td>7 2 1 3</td>
<td>5 1 0 2</td>
</tr>
<tr>
<td>P₂</td>
<td>1 1 4 0</td>
<td>1 1 5 0</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>P₃</td>
<td>1 0 2 1</td>
<td>3 5 2 2</td>
<td>2 5 0 1</td>
</tr>
</tbody>
</table>

a) Fill in the contents of the matrix \( \text{Need} \) for each process in the space above.

b) Is the system in a safe state? Show why or why not.

\[ \text{No, it is not in a safe state.} \]

The only process that can be allocated full resources and run is \( P₂ \). After it runs and releases all its resources, no other process can run, so the system is deadlocked.
4. Consider the following proposed solution to the mutual exclusion problem: (Bakers Algorithm)

```plaintext
1  bool choosing[n];  // Initialized to false
2  int number[n];    // Initialized to all zeros
3
4  do {
5      choosing[i] = true;
6      number[i] = max(number[0], number[1], ..., number[n-1]) + 1;
7      choosing[i] = false;
8     for (int j=0; j<n; j++) {
9         while (choosing[j]) no-op;
10        while (number[j] != 0) && (number[j] < (number[i], i))
11            ; // no-op
12     }
13
14     critical section;
15
16     number[i] = 0;
17
18     remainder section;
19
20  } while (true);
```

Given that:
- process numbers in the system are between 0 and n-1
- (a,b) < (c,d) if a < c or if a = c and b < d.
- max(a₀, ..., a_n-1) is a number k, such that k ≥ i for i = 0, ..., n-1.

Do a trace below for two processes, showing the development of a worst-case scenario if line 9 above were deleted. Assume that before these processes arrive, max(a₀, ..., a_n-1) = 12.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1-6</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>F</td>
<td>(13)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8-12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When comparing against choosing[3], it doesn't stop and passes through the while loop in line 10.

Since number[3] is 0, 14. critical section

14. Also enters critical section
5. Monitors can be implemented using Semaphores.

   a) What semaphores, variables, etc. are needed to implement this?
   
   2 pts. → A semaphore mutex for mutual exclusion, surrounding each monitor function
   3 pts. → A semaphore per condition variable

   b) In the space below, implement the monitor commands *WAIT(c)* and *SIGNAL(c)* using only the semaphore operators *UP(c)* and *DOWN(c)*.

   1. *WAIT(c)*:
      
      \[
      \text{Up (mutex)}; \\
      \text{Down (c)}; \\
      \text{Down (mutex)}; \\
      \]

   2. *SIGNAL(c)*:
      
      \[
      \text{Up (c)}; \\
      \]

6. Assume processes spend 85% of run time in an I/O wait state. Assuming:
   - The O.S. is in ROM (i.e. no RAM needed for it)
   - We need 256K per running process

   What is the minimum amount of memory (to the nearest K) you need to get CPU utilization \(\geq 99\%\)?

   Note that: \(\log_{0.85} y = \ln(y) / \ln(0.85)\)

   \[
   \text{CPU utilization is } 1 - p^n \text{ or } 1 - 0.85^n
   \]

   \[
   1 - 0.85^n \geq 0.99 \\
   0.85^n \leq 0.01
   \]

   Solving for \(n\):

   \[
   n = \log_{0.85} 0.01 \\
   = \frac{\ln 0.01}{\ln 0.85} \\
   = 28.3 \quad \text{8 pts.}
   \]

   \[
   \lceil 28.3 \rceil = 29
   \]

   \[
   29 \times 256K = 7,252,674 \Rightarrow 7.25 \text{ Meg}
   \]

   (7424 K.)

   5 pts.