The value of the exam will be 100%. You can choose to answer questions to add up to 100, or answer all of them and have more chance to earn more points.

1. (10 points) Some computer systems do not provide a privileged mode of operation in hardware. How can you construct a secure operating system for these computer systems? Provide one solution and convincing justifications.

**Answer:** An operating system for a machine of this type would need to remain in control (or monitor mode) at all times. This could be accomplished by two methods:

a. Software interpretation of all user programs (e.g. Java, LISP, etc.). The software interpreter would provide, in software, what the hardware does not provide.

b. Requirement that all programs be written in high-level languages so that all object code is compiler-produced. The compiler would generate (either in-line or by function calls) the protection checks that the hardware is missing.

2. (10 points) Describe the actions taken by a kernel to context-switch between processes.

**Answer:** In general, the operating system must save the state of the currently running process and restore the state of the process scheduled to be run next. Saving the state of a process typically includes the values of all the CPU registers in addition to memory allocation. Context switches must also perform many architecture-specific operations, including flushing data and instruction caches.

3. Write pseudo-code to ensure that processes A and B both complete execution before process C begins. Note, there is no critical section here. We are using semaphores to **ENFORCE PRECEDENCE** - to coordinate processes.

   a) (10 points) Using semaphores. Define the semaphore(s), initialize, and write the code for each process to execute.

**Answer:**

Shared structures:

```plaintext
Semaphore SemA=0, SemB=0;
```

<table>
<thead>
<tr>
<th>Code for process A</th>
<th>Code for process B</th>
<th>Code for process C</th>
</tr>
</thead>
<tbody>
<tr>
<td>// do your work</td>
<td>// do your work</td>
<td>wait(semA);</td>
</tr>
<tr>
<td>signal(SemA);</td>
<td>signal(SemB);</td>
<td>wait(semB);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>// do your work</td>
</tr>
</tbody>
</table>
b) (10 points) Using TestAndSet. Define variables(s), initialize, and write code for each process to execute.

Answer:

Shared structures:

```
Boolean doneA=true, doneB=true;
```

<table>
<thead>
<tr>
<th>Code for process A</th>
<th>Code for process B</th>
<th>Code for process C</th>
</tr>
</thead>
<tbody>
<tr>
<td>// do your work</td>
<td>// do your work</td>
<td>while(TestAndSet(doneA));</td>
</tr>
<tr>
<td>doneA=false;</td>
<td>doneB=false;</td>
<td>while(TestAndSet(doneB));</td>
</tr>
<tr>
<td></td>
<td></td>
<td>// do your work</td>
</tr>
</tbody>
</table>

4. The Fibonacci sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8, .... Formally, it can be expressed as:

\[
\begin{align*}
  f_{ib_0} &= 0 \\
  f_{ib_1} &= 1 \\
  f_{ib_n} &= f_{ib_{n-1}} + f_{ib_{n-2}}
\end{align*}
\]

a) (10 points) Write a Java program using threads that that generates the n-th element of the Fibonacci sequence. A thread in charge of computing the n-th member of the sequence, will generate two threads, which in turn will compute the n-1st and n-2nd elements of the sequence. The thread in charge of computing the n-th term will gather those results and add them to produce the result.

Answer: See posted solution FibI.java

b) (30 points) A second version of the program will not spawn as many threads as the first one. All threads will share an integer array, whose elements are initialized to -1. If a thread wants to compute a member of the array, it first checks to see if it has been computed before (it is not -1 anymore). If it has been computed already, the thread finishes, otherwise, it performs its computation. So, no element of the sequence is computed more than once. Note that the array may be concurrently accessed by more than one thread, so your solution needs to provide the mechanism to ensure data integrity. That is, all threads need to share an array of semaphores to synchronize access to the integer array. Write a Java program to compute the first elements of a Fibonacci sequence.

Answer: See posted solution FibII.java

5. (10 points) What is the meaning of the term busy waiting? What other kinds of waiting are there in an operating system? Can busy waiting be avoided altogether? Explain your answers.

Answer: Busy waiting means that a process is waiting for a condition to be satisfied in a tight loop without relinquishing the processor. Alternatively, a process could wait by relinquishing the processor, and block on a condition and wait to be awakened at some appropriate time in the future. Busy waiting can be avoided but incurs the overhead associated with putting a process to sleep and having to wake it up when the appropriate program state is reached.
6. (20 points) The first known correct software solution to the critical-section problem for two processes was developed by Dekker. The two processes, P0 and P1, share the following variables:

```java
boolean flag[2]; /* initially false */
int turn;
```

The structure of process Pi (i == 0 or 1) is shown next; the other process is Pj (j == 1 or 0).

Prove that the algorithm satisfies all three requirements for the critical-section problem.

```
do {
    flag[i]= TRUE;
    while(flag[j]) {
        if (turn == j) {
            flag[i] = FALSE;
            while (turn == j) ; // do nothing
            flag[i] = TRUE;
        } // if
    } // while
    // critical section
    turn = j;
    flag[i] = FALSE;
    // remainder section
} while (TRUE);
```

**Answer:** This algorithm satisfies the three conditions of the critical section problem.

1) Mutual exclusion is ensured through the use of the flag and turn variables. If both processes set their flag to true, only one will succeed. Namely, the process whose turn it is. The waiting process can only enter its critical section when the other process updates the value of turn.

2) Progress is provided, again through the flag and turn variables. This algorithm does not provide strict alternation. Rather, if a process wishes to access their critical section, it can set their flag variable to true and enter their critical section. It only sets turn to the value of the other process upon exiting its critical section. If this process wishes to enter its critical section again - before the other process - it repeats the process of entering its critical section and setting turn to the other process upon exiting.

3) Bounded waiting is preserved through the use of the turn variable. Assume two processes wish to enter their respective critical sections. They both set their value of flag to true, however only the thread whose turn it is can proceed, the other thread waits. If bounded waiting were not preserved, it would therefore be possible that the waiting process would have to wait indefinitely while the first process repeatedly entered - and exited - its critical section. However, Dekker's algorithm has a process set the value of turn to the other process, thereby ensuring that the other process will enter its critical section next.
7. (20 points) Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>P4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>P5</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all at time 0.

a. Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1) scheduling.

Answer: Color identifies processes

b. What is the turnaround time of each process for each of the scheduling algorithms in part a?

Answer:
c. What is the waiting time of each process for each of the scheduling algorithms in part a?

**Answer:** turnaround time minus burst time.

<table>
<thead>
<tr>
<th></th>
<th>FCFS</th>
<th>SJF</th>
<th>Priority</th>
<th>RR</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0</td>
<td>9</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>$P_2$</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>11</td>
<td>2</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>$P_4$</td>
<td>13</td>
<td>1</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>$P_5$</td>
<td>14</td>
<td>4</td>
<td>1</td>
<td>9</td>
</tr>
</tbody>
</table>

d. Which of the schedules in part a results in the minimal average waiting time (over all processes)?

**Answer:** Shortest Job First

<table>
<thead>
<tr>
<th>Avg. Waiting Time</th>
<th>FCFS</th>
<th>SJF</th>
<th>Priority</th>
<th>RR</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.6</td>
<td>3.2</td>
<td>8.2</td>
<td>5.4</td>
<td></td>
</tr>
</tbody>
</table>