Problem 1 (6 Points) [From Stallings]
Jurassic Park consists of a dinosaur museum, and a park for safari riding. There are $N$ single-passenger cars and $M$ visitors. Each visitor wanders around the museum for a while, and then lines up to take a ride in a safari car. When a car is available, it loads the one passenger it can hold; waits for the visitor to signal he/she is ready to start the ride; and travels around the park for a random amount of time before returning to the museum. If the $N$ cars are all being used, a visitor who wants to ride must wait; if a car is ready to load but there are no waiting visitors, then the car must wait. After the ride in the park, the car signals the visitor when it is safe to exit the car, and the visitor leaves the park.

The algorithm skeleton below simulates the above scenario. Note that the Observer process should be able to determine accurately the number of cars that are moving through the park at any random time. Complete the algorithm below using semaphores to synchronize the $M$ passenger processes and the $N$ car processes. Explain the purpose of each semaphore and shared variable.

```c
int nFullCars = 0; // number of full cars
Process Visitor (i) {
    ... walk around museum ...
    ... ride around park ...
}
Process Car (j) {
    do forever {
        ... ride around park ...
    }
}
Process Observer {
    do forever {
        ... sleep for a random amount of time ...
        printf("nFullCars = %d\n", nFullCars);
    }
}
```

Problem 2 (4 Points) [From Tanenbaum]
A system has four processes and five allocatable resources. The current allocation and maximum needs are as follows:

<table>
<thead>
<tr>
<th>Process</th>
<th>Allocated</th>
<th>Maximum</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1 0 2 1 1</td>
<td>1 1 2 1 3</td>
<td>0 0 x 1 1</td>
</tr>
<tr>
<td>B</td>
<td>2 0 1 1 0</td>
<td>2 2 2 1 0</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1 1 0 1 0</td>
<td>2 1 3 1 0</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>1 1 1 1 0</td>
<td>1 1 2 2 1</td>
<td></td>
</tr>
</tbody>
</table>

What is the smallest value of $x$ for which this is a safe state? Explain.
Problem 3 (4 Points)

Consider an I/O system in which input process I, user process P, and output process O are connected by two buffers. The processes exchange data in equal sized blocks. Furthermore, process P produces 1 output block for each input block and consumes input blocks one at a time. These blocks are buffered on disk using a floating boundary between the input and output buffers. The communication primitives ensure that the following resource constraint is satisfied:

\[ n_I + n_O \leq M \]  

(1)

where \( M \) is the maximum number of blocks on disk, \( n_I \) is the number of input blocks on disk, and \( n_O \) is the number of output blocks on disk.

a) If all processes will eventually continue execution if there is sufficient disk buffers, under what circumstances will the system deadlock if buffers are never preempted?

b) What additional constraint(s) is (are) required to prevent deadlock but still permit the boundary between input and output buffers to vary in accordance to the present needs of the processes?

c) Suppose that the disk space must also hold buffers for intermediate processing by process P as it computes the contents of the corresponding output buffer. Furthermore, suppose that there are only 4 input message types, and a type \( i \) message requires 1 input buffer and \( p_i, i = 1, 2, 3, 4 \), processing buffers. What additional constraint(s) is (are) required to avoid deadlock but still permit the boundary between input, output, and processing buffers to vary in accordance to the present needs of the processes?

Problem 4 (8 Points)

This is a warm-up for Project Ax, the extension to Project A. You should be able to reuse most of the code developed for this homework.

Write a program called `npipe` that has the following optional flags:

`npipe [-n N]`

where the default value for \( N \) is 3. \( N \) indicates the number of instances of `npipe`. These instances should form a pipeline of \( N \) processes, each reading from stdin and writing to stdout. As a verification that each process outputs the same bytes, each process should compute the sum of all bytes treating each byte as an unsigned integer value and print this sum one second after it detects EOF on stdin.

Submit the following:

- A listing of the source code.
- The output and an explanation of a test run for the case \( N = 3 \) that indicates that your program is functioning properly. A test input file will be provided.