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 (0 Points)

Tanenbaum, Problem 40 (Chapter 2).
**Problem 3 (2 Points)**

- Consider the case of two tasks \( N = 2 \) where the service demands are \( t(1) = 5 \) and \( t(2) = 3 \). What is the average response time for the two possible service orderings (i.e., job 1 then job 2 and job 2 then job 1)?

- Consider the more general case of \( N \) jobs with service demands of \( t(i), \ i=1:N \) where the service demands are strictly ordered; i.e., \( t(1) < t(2) < \ldots < t(N) \). Prove that SJF will result in the smallest average response time. HINT: Consider the average response time when Job i swaps service position with Job i-1.

**Problem 4 (2 Points)**

Suppose that a user's last four requests have used 10, 20, 30 and 40 milliseconds of CPU time respectively. What is the predicted CPU demand if we use an aging algorithm with \( a = 1/2 \)? Assume that the first estimate is equal to the first observation 10.

**Problem 5 (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></th>
<th>Allocated</th>
<th>Maximum</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process A</td>
<td>1 0 2 1 1</td>
<td>1 1 2 1 2</td>
<td>0 0 x 1 1</td>
</tr>
<tr>
<td>Process B</td>
<td>2 0 1 1 0</td>
<td>2 2 2 1 0</td>
<td></td>
</tr>
<tr>
<td>Process C</td>
<td>1 1 0 1 0</td>
<td>2 1 3 1 0</td>
<td></td>
</tr>
<tr>
<td>Process 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 6 (8 Points)**

This is a warm-up for Project B, the extension to Project A. Write a program called `npipe` that has the following optional flags:

```
npipe [-n N] [-v]
```

where the default value for \( N \) is 3. \( N \) indicates the number of instances of `npipe`. Basically, you are implementing the pipeline "npipe | npipe | ... | npipe" where the \( k \)th process should create the pipe to the \((k+1)\)th process and fork the \((k+1)\)th process. The `-v` flag indicates verbose output where each pipe file descriptor creation and close is accompanied by a message indicating the action (create or close), the process id and the affected file descriptor. When reading/writing from/to a pipe, use unbuffered I/O (i.e., read(2) and write(2)). 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 `npipe -n 3 -v` that indicates that your program is functioning properly. A test input file will be provided.