Homework 6

1. (10 points) A network is shared among different jobs, each of which wants to have multiple concurrently open TCP connections. The network limits the total number of open TCP connections across all jobs to n. Write a monitor (use the pseudocode language of the textbook) to manage access to the network. The monitor should provide the functions:

```
init(int n); /* initialize the limit to n */
request(int n); /* request n connections,
                   does not return until request is granted */
release(int n); /* release n connections */
```

2. (10 points) A solution to the Readers/Writers Problem that avoids starvation is described here: https://www.cs.umd.edu/~hollings/cs412/s96/synch/synch1.html

The pseudocode is shown for the readers, the code for the writers just swaps all references to "r" and "w". A translation of the pseudocode that follows our book's notation follows.

`nw_active` – number of active writers initialized to 0.
`nr_active` – number of active readers initialized to 0.
`nw_waiting` – number of waiting writers initialized to 0.
`nr_waiting` – number of waiting readers initialized to 0.
`mutex` – a semaphore initialized to 1.
`r_sem` – a semaphore initialized to 0.
`w_sem` – a semaphore initialized to 0.

```
 1   wait( mutex );
 2   if ( nw_active + nw_waiting == 0 )
 3   {
 4     nr_active++; // Notify we are active
 5     signal( r_sem );  // Allow ourself to get through
 6   } else {
 7     nr_waiting++;  // We are waiting
 8   }
 9   signal( mutex );
10   wait( r_sem ); // Readers wait here, if they must wait.
11
12   READING...
13
```

```
14    wait( mutex );
15    nr_active--;
16    if (( nr_active == 0 ) && ( nw_waiting > 0 ))
      // If we are the last reader
17    {
18      while ( nw_waiting > 0 )
        // Allow all waiting writers to enter
19      {
20        signal( w_sem );  // wake a writer;
21        nw_active++;  // one more active writer
22        nw_waiting--; // one less waiting writer.
23      }
24    }
25    signal( mutex );
```

Usually, semaphores are used to count the number of resources of a particular type available. Describe what `r_sem` and `w_sem` are counting in the above solution. How do these semaphores prevent starvation?

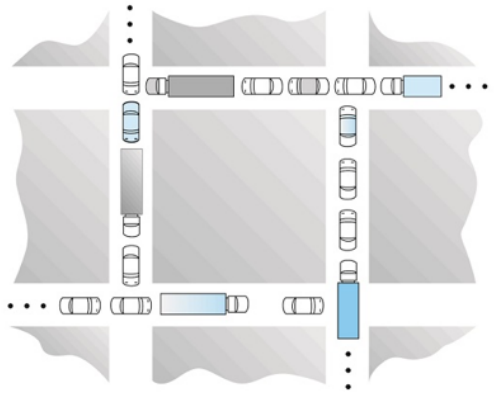
3. (15 points) Consider the following sushi bar problem. There is a sushi bar with one chef and N seats. When a customer arrives, they take a seat at the bar if there is an empty one or they leave if no seats are empty. After taking a seat, they wait for the sushi chef to be available and then give their order. After the sushi chef servers them they eat and leave. Write a solution to the sushi bar problem that uses semaphores. You should write three procedures: `init()` `chef()` and `customer()`.

```
void init() {
    /* provide any initialization */
}

void chef() {
    while (true) {
        /* wait for ready customer */
        /* serve customer */
    }
}

void customer() {
    /* take seat if available, otherwise leave */
    /* wait for chef */
    /* give order */
    /* eat and leave */
}
```

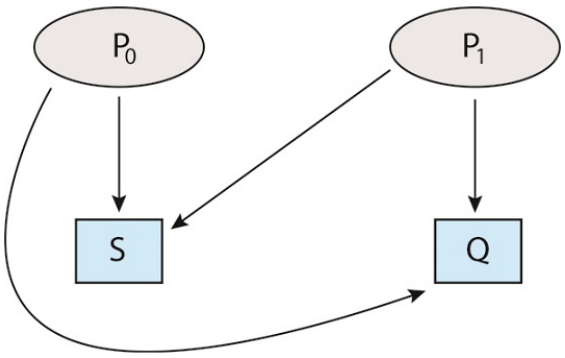
4. Exercise 8.12 (10 points) Consider the traffic deadlock depicted in the figure below.



a. Show that the four necessary conditions for deadlock hold in this example.
b. State a simple rule for avoiding deadlocks in this system.

5. (10 points) Consider a system consisting of three resources of the same type that are shared by two threads, each of which needs at most two recourses. Is deadlock possible? Explain your answer.

6. Exercise 8.16 modified (10 points) The resource-allocation graph shown below does not always lead to a deadlock, the scheduler plays a role. Assume P_0 and P_1 have CPU bursts of 20 ms each, draw example Gantt charts that demonstrate FCFS and RR (with quanta=5). Use the charts to explain which algorithm is more likely to result in a deadlock?



7. Exercise 8.18 (15 points) Which of the six resource-allocation graphs shown below illustrate deadlock? For those situations that are deadlocked, provide the cycle of threads and resources. Where there is not a deadlock situation, illustrate the order in which the threads may complete execution.