MUST BE TURNED IN WITH RESONABLE ATTEMPTS AT SOLUTIONS BY 12:05pm OCTOBER 13
HOMEWORK WILL NOT BE GRADED

Homework 4
COM S 352
Fall 2021

1. Race conditions are possible in many computer systems. Consider an online auction system where the current highest bid for each item must be maintained. A person who wishes to bid on an item calls the `bid(amount)` function, which compares the amount being bid to the current highest bid. If the amount exceeds the current highest bid, the highest bid is set to the new amount. This is illustrated below:

```
void bid(double amount) {
  if (amount > highestBid) {
     highestBid = amount;
  }
}
```

Describe how a race condition is possible in this situation and what might be done to prevent the race condition from occurring.

2. Consider the code example for allocating and releasing processes.

```
#define MAX_PROCESSES 255
int number_of_processes = 0;
/* the implementation of fork() calls this function */
int allocate_process() {
  int new_pid;
  if (number_of_processes == MAX_PROCESSES) {
    return -1;
  } else {
    /* allocate necessary process resources */
    number_of_processes++;
    return new_pid;
  }
}

/* the implementation of exit() calls this function */
void release_process() {
  /* release process resources */
  number_of_processes--;
}
```

a. Identify the race condition(s).

b. Assume you have a mutex lock named mutex with the operations lock() and unlock(). Indicate where the locking needs to be placed to prevent the race condition(s).

3. Assume a network is shared among different services, each of which wants to have multiple concurrently open TCP connections. The network sets a limit on the total number of open TCP connections. The purpose of the methods below is to keep track of the number of open connections so that the limit is never exceeded. For example, if a service want to open 5 new connections it must first call `request(5)`.

Implement the methods (pseudocode is fine) using only **locks** and **condition variables** to manage the concurrency.

```
/* initialize the network capacity to n connections */
void
init(int n);


/* The purpose of this function is to block (not return) until
 * n network connections are granted to the caller.
 *
 * On return, the caller assumes they are allowed to open n
 * network connections.
 */
void
request(int n);

/* release n connections */
release(int n);
```

4. 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 serves them, they eat and leave. Write a solution to the sushi bar problem that uses only **semaphores** to manage the concurrency. You should write three procedures: `init()`, `chef()` and `customer()`.

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

}

void chef() {
    while (true) {
        /* wait for ready customer */
        printf("Welcome to the sushi bar.\n");
        /* take order */
```

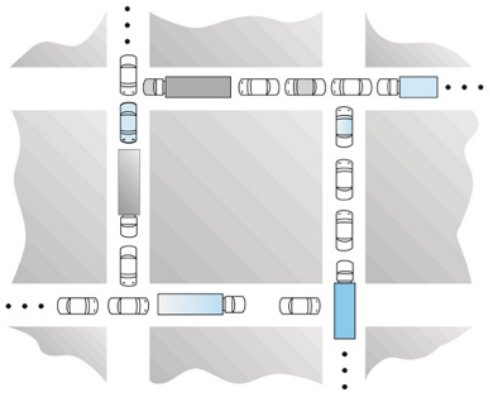
```
        printf("What will you have?\n");
        /* serve customer */
        printf("Here is your food.\n");
    }

}

void customer() {
    /* take seat if available, otherwise leave */
    printf("This seat looks good.\n");
    /* wait for chef */
    printf("Hello.\n");
    /* give order */
    printf("I will have one roll.\n");
    /* eat and leave */
    printf("That was good.\n");
}
```

5. Consider the traffic deadlock depicted in the figure below.

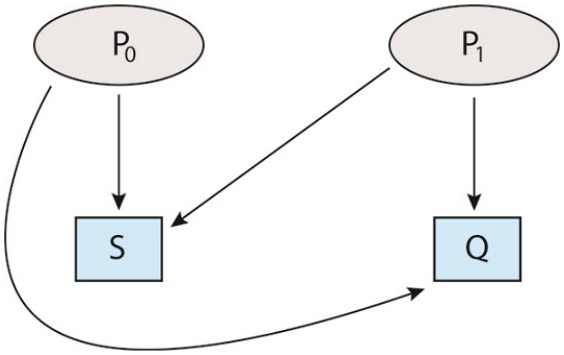
a. Show that the four necessary conditions for deadlock hold in this example.
b. State a simple rule for avoiding deadlocks in this system.



6. 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.

7. The resource-allocation graph blow shows that both processes P0 and P1 need both resources S and Q to complete.

a. Explain how a deadlock is possible in this system.
b. Although a deadlock is possible, the system will not always enter 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?


P_0
P_1
S
Q