Question 1 - Answer the questions related to this document in q1-answer.txt (or create q1-answer.pdf)

Note that the first page of this document is copied exactly from the question on Quercus.

To support curbside pickup for a major retailer, an app has been developed so that customers can wait in the vicinity until an agent is ready to serve them.

A person arriving in the vicinity will click a button on the app, and will wait until the app returns with an agent number. The person then goes to the booth with the correct agent number on it to be served. When the person leaves, the agent lets the app know that this agent is available.

Assume that loading the value of a variable is atomic. In other words, you are guaranteed to load a valid value, not some in-between state between two valid values.

For the solution to be correct, it must satisfy three criteria:
1) An agent serves at most one person at a time
2) The count of the number of people being served at any point in time is accurate
3) If an agent is available, then a person calling find_agent should be able to get a counter number.

The app has 2 functions: `find_agent()` and `ready()`. The `find_agent()` function is invoked in a separate thread for each customer. The `ready()` function is invoked in a separate thread for each agent.

Two variables and one constant control the state:

- `available[i]` is set to true if agent number i is available.
- `num_people` is a global variable that counts the total number of people being served at a time.
- `N` is maximum number of agents and is constant

```
// Called by a customer thread to get the booth number for an
available agent

int find_agent() {
    while(1) {
        for(int i = 0; i < N; i++) {
            if(available[i] == true) {
                available[i] = false;
                num_people += 1;
                return i;
            }
        }
    }
}

// Called by an agent thread who passes in their agent number
void ready(int agent_num) {
    available[i] = true;
    num_people -= 1;
}
```

Consider 3 different versions to solve the synchronization problems in the original version. The added code is bolded and in red. C-like pseudo-code is used for simplicity .

VERSION 1:

```
Semaphore lock = 1;

int find_agent() {
    while(1) {
        sem_wait(lock);
        for(int i = 0; i < N; i++) {
            if(available[i] == true) {
                available[i] = false;
                num_people += 1;
                return i;
            }
        }
    }
}

void ready(int agent_num) {
    available[i] = true;
    num_people -= 1;
    sem_signal(lock);
}
```

VERSION 2:

```
Lock lock;  // assume lock is correctly initialized

int find_agent() {
   while(1) {
        lock(lock);
        for(int i = 0; i < N; i++) {
            if(available[i] == true) {
                available[i] = false;
                num_people += 1;
                unlock(lock);
                return i;
            }
        }
       unlock(lock);
    }
}

void ready(int agent_num) {
    lock(lock);
    available[i] = true;
    num_people -= 1;
    unlock(lock);
}
```

VERSION 3

```
Lock lock;  // assume lock is correctly initialized

int find_agent() {
   while(1) {
        for(int i = 0; i < N; i++) {
            lock(lock);
            if(available[i] == true) {
                available[i] = false;
                num_people += 1;
                unlock(lock);
                return i;
            } else {
                unlock(lock);
            }
        }
    }
}

void ready(int agent_num) {
    lock(lock);
    available[i] = true;
    num_people -= 1;
    unlock(lock);
}
```