CS 361: Computer Systems **Lab Session 12 Worksheet**
Fall 2019

Adding Multiple Elevators

1. Hopefully by now you have working synchronization with your single elevator. This means your elevator should wait at a floor for a passenger to enter, and then take that passenger directly to their floor, and wait for the passenger to get off. If you do not have this working yet, please consult the worksheet from last week, and spend some time working on it.

2. Once you have your elevator picking up your passengers correctly, let's start using all the elevators!

3. Right now, we have one set of variables associated with the elevators:

```
pthread_mutex_t lock;
int current_floor;
int direction;
int occupancy;
enum {ELEVATOR_ARRIVED=1, ELEVATOR_OPEN=2, ELEVATOR_CLOSED=3} state;
```

4. You will want to modify this so we have one of each of these variables per elevator.

5. Next, look at this code from the elevator ready function

```
void elevator_ready(int elevator, int at_floor,
                       void(*move_direction)(int, int),
                       void(*door_open)(int), void(*door_close)(int)) {
       if(elevator!=0) return;
```

6. Answer the questions about the elevator code on blackboard.

7. Next, look at the passenger code, below

```
void passenger_request(int passenger, int from_floor, int to_floor,
                            void (*enter)(int, int), void(*exit)(int, int))
 {
// wait for the elevator to arrive at our origin floor, then get in
int waiting = 1;
while(waiting) {
        pthread_mutex_lock(&lock);

        if(current_floor == from_floor && state == ELEVATOR_OPEN && occupancy==0){
            enter(passenger, 0);
            occupancy++;
            waiting=0;
            }
            pthread_mutex_unlock(&lock);
        }

        // wait for the elevator at our destination floor, then get out
        int riding=1;
        while(riding) {
            pthread_mutex_lock(&lock);

            if(current_floor == to_floor && state == ELEVATOR_OPEN) {
                exit(passenger, 0);
                occupancy--;
                riding=0;
            }
            pthread_mutex_unlock(&lock);
        }
    }
```

8. Right now, this code assumes a single elevator. You will need to assign the passenger to an elevator, and then make sure you are using that elevator and its variables.

9. It may be helpful to add a struct for each passenger, as well as each elevator.

10. Some things you will need to update include the state and occupancy variables and the mutex (make sure they belong to the correct elevator!). Note that these are all design decisions and ultimately up to you! In particular, having a mutex per elevator is one way to move forward with this implementation but is by no means the only way.

11. Note that passenger_request calls a passed in enter function. The function that main passes in is

    ```
    void passenger_enter(int passenger, int elevator)
    ```

12. A similar function is passed in for the exit function.

13. Answer the last questions on blackboard.

14. After you've set things up to use multiple elevators, make sure you thoroughly test your code. Remember that race conditions mean your code may work some of the time, but not all of the time!