CS 361: Computer Systems **Lab Session 11 Worksheet**
Fall 2019

pthread_mutex_t

1. The pthread_mutex_t guarantees mutual exclusion. Only one thread at a time is allowed into code protected by the mutex.
2. Before you use the mutex, you will need to set it up by calling pthread_mutex_init(pthread_mutex_t *mutex, const pthrad_mutexattr_t *attr). This will initialize the lock and assign a unique id to the mutex. (*attr can be NULL.)
3. Before any code you don't want multiple threads in, call pthread_mutex_lock(pthread_mutex_t *mutex). This will allow one thread in, and block any other threads until that thread has released the mutex.
4. Make sure to call pthread_mutex_unlock(pthread_mutex_t *mutex) to release the mutex after the critical section.
5. When you are done with the mutex, you can destroy it with pthread_mutex_destroy(pthread_mutex_t *mutex).
6. The code below uses mutexes to protect shared variables: for example, the passenger_request function uses a mutex to protect where it checks and changes elevator variables, in order to ensure that multiple passengers do not edit the elevator at the same time.

pthread_barrier_t

1. Have you ever been to a restaurant where the hostess won't let you sit at your table until your entire party is there? A barrier is basically a restaurant hostess for threads: all threads will block until a certain number of them show up at the barrier.
2. To set up the barrier, call pthread_barrier_init(pthread_barrier_t *barrier, const pthrad_barrierattr_t *attr, unsigned count). Attr can be NULL, and count will be the number of threads that need to meet up for your barrier to be released.
3. Calling pthread_barrier_wait(pthread_barrier_t *barrier) will cause all threads to block at that line of code until the number of threads specified when the barrier was initialized

Coordinating passengers and elevators

```
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);
        }
}

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;

        pthread_mutex_lock(&lock);
        if(state == ELEVATOR_ARRIVED) {
                door_open(elevator);
                state=ELEVATOR_OPEN;
        }
        else if(state == ELEVATOR_OPEN) {
                door_close(elevator);
                state=ELEVATOR_CLOSED;
        }
        else {
                if(at_floor==0 || at_floor==FLOORS-1)
                        direction*=-1;
                move_direction(elevator,direction);
                current_floor=at_floor+direction;
                state=ELEVATOR_ARRIVED;
        }
        pthread_mutex_unlock(&lock);
}
```

1. The code for the elevator and passenger threads is above. As you can see, right now elevators stop and open their doors at every floor. Try adding code so that your elevator is assigned a passenger, and rather than stopping at every floor, goes directly to that passenger's from_floor, picks up the passenger, and takes them directly to their to_floor. This will probably require adding new variables to the existing elevator variables, as you will need some way for the elevator to know which passenger it is assigned to.

2. Remember that the elevator is full when it is holding a single passenger, so you do not have to worry about picking up multiple passengers.

3. Notice that right now the elevator also does not always wait for passengers. Try adding a barrier so that when the elevator gets to the passenger's floor, it stops until the passenger gets on.

4. Answer the question on Gradescope.