Computer Systems Engineering-1

Assignment 4

Tanish Lad - 2018114005

Ober Cab Services Question Report

OVERVIEW

The code is an attempt to simulate a working Cab Service System which consists of N Cabs, M Riders and K Payment servers.

Functions Used

1. rideEnded:

   This signifies that a rider has completed his ride, and the cab should further search for a rider who is waiting for a cab. This function essentially searches for a rider, who initially had searched for a cab, but hadn't got any because there were no more cabs left of that type.

   The screenshot below represents the code for one of the possible three states the cab could have been when it was in ride mode. The ride was premier, and since the ride got over, the cab is iterating over all the riders, and checking if any of the rider is in waiting state (i.e. is waiting for a rid). If it finds one, it changes it state according to the type of cab required by the rider and changes the struct of the rider to match the details of the cab it is riding on, and also changes it's waiting

flag to zero again. The later if condition is just for the conditional lock on the maximum waiting time that I used.

```
if(cabss[riders[id-1].cabID - 1].state == 3)
{
    cabss[riders[id-1].cabID - 1].state = 0;

    for(int i = 0; i < M; i++)
    {
        if(riders[i].waiting == 1)
        {
            riders[i].waiting = 0;

            if(riders[i].type == 1)
            {
                cabss[riders[id-1].cabID - 1].state = 1;
                riders[i].cabID = riders[id-1].cabID;
                riders[i].found = 1;
            }

            else
            {
                cabss[riders[id-1].cabID - 1].state = 3;
                riders[i].cabID = riders[id-1].cabID;
                riders[i].found = 1;
            }
        }

        if(cabss[riders[id-1].cabID - 1].state > 0)
        {
            pthread_cond_signal(&(riders[i].condLockRider));
            break;
        }
    }
}
```

2. BookCab:

   This is the function called by each of the thread, one thread signifying one rider, which initializes each of the rider, by

randomly deciding it's Ride time, it's Maximum Waiting time, and the type of cab (Premier/Pool) it requires. The code snippet below then shows how the rider finds, (or doesn't find) a rider according to the type of cab (in this snippet, pool) it requires.

```
pthread_mutex_lock(&lockCab);

if(r -> type == 1)
{
    for(int i = 0; i < N; i++)
    {
        if(cabss[i].state == 1)
        {
            cabss[i].state = 2;
            r -> cabID = cabss[i].id;
            r -> found = 1;
            break;
        }
    }

    if(r -> found == 0)
    {
        for(int i = 0; i < N; i++)
        {
            if(cabss[i].state == 0)
            {
                cabss[i].state = 1;
                r -> cabID = cabss[i].id;
                r -> found = 1;
                break;
            }
        }
    }
}
```

It initially tries to find a cab which already has one passenger riding in it. If it fails to find such cab, then it tries to find a cab

which is in zero state (no riders). If it finds any cab, then the details of that cab and the rider are changes accordingly.

If it found any cab, it then calls the function rideEnded.

3. makePayment:

   This function is called by each of the thread, one thread signifying one payment server, and it then searches for a rider who has completed his ride but hasn't completed his payment yet.

```
for(int i = 0; i < M; i++)
{
    pthread_mutex_lock(&riders[i].lockRider);

    if(riders[i].completed == 1 && riders[i].paid == 0)
    {
        sem_wait(&payment);

        riders[i].paid = 1;

        printf("Server %d has accepted payment on Rider %d\n", s -> id, riders[i].id);

        pthread_mutex_unlock(&riders[i].lockRider);

        sleep(2);

        pthread_mutex_lock(&lockPayment);
        paymentDone++;
        pthread_mutex_unlock(&lockPayment);

        sem_post(&payment);

        // pthread_mutex_lock(&riders[i].lockRider);
    }

    else
    {
        pthread_mutex_unlock(&riders[i].lockRider);
    }
}
```

   If it is able to find such a rider, then it uses the mutex lock on the rider, and waits if there are no more resources available

(using a semaphore) then makes the payment and then posts the semaphore signifying that server can be used again to make another payment.