REPORT
QUESTION 3

Submitted by : Tanvi Kamble
2018114004

Aim: Implementing a Cab system

Code:

- Global variables:
  - sem_t server; ---------- Semaphora
  - pthread_mutex_t mutex_cab[100]; ---------- mutex for each driver
  - pthread_mutex_t mutex_wait;---------- mutex for threads
    - int n, m, k;----------n = number of drivers, m = riders and k = number of servers
  - struct ride_info ----------info about each rider

    {

    int cabtype, wait_time, ride_time, cab;

    };---------- cab = the id of the cab alloted
  - struct ride_info *riders[100];
  - struct point

    {

    int num;

    };

- int cab_stat[100] = {0} ---------- Status of a cab (0 for empty cab 1 for premium ride, 2 for fullpool and 3 for pool with only one passenger)
- pool_one[100] = {0}, pool_two[100] = {0} ------- rider ids for pool

➢ Int main():

```
sem_init(&server, 0, k);
```

- Initialize semaphora

```
for (int i = 1; i <= n; i++)
{
    ki = (struct point *)malloc(sizeof(struct point));
    ki->num = i;
    pthread_create(&cabs_th[i], NULL, driverthread, ki);
    sleep(1);
}
```

- Create separate threads for each driver and store the thread id in cabs_th

```
for (int i = 1; i <= m; i++)
{
    printf("Enter Rider %d\n", i);
    ki = (struct point *)malloc(sizeof(struct point));
    ki->num = i;
    pthread_create(&riders_th[i], NULL, riderthread,
ki);
    sleep(1);
}
```

```
for (int i = 1; i <= m; i++)
{
    pthread_join(riders_th[i], NULL);
}
```

- Create new threads for each of the riders and store the thread id in riders_th and then finally wait for all threads to complete.

➢ `void *driverthread(void *i)`:

- Fuction accessed by the driver thread

```
{
    pthread_mutex_lock(&mutex_wait);
    struct point *ki = i;
    int num = ki->num;
    cab_stat[num] = 0;
    pthread_mutex_unlock(&mutex_wait);
}
```

- Initialize all the cars to waiting state

➢ void *riderthread(void *i):

- Each rider calls the function book cab

```
{
```

```
pthread_mutex_lock(&mutex_wait);
struct point *ki = i;
int num = ki->num;
int tp = riders[num]->wait_time;
pthread_mutex_unlock(&mutex_wait);
book_cab(num);
```

➢ void book_cab(int i)

```
{
    int cab = 0;
   int mode_i = riders[i]->cabtype;
    int tp = riders[i]->wait_time;
    int temp = 0, flag = 0;
    while (temp++ <= tp)
    {
       for (int x = 1; x <= n; x++)
       {
          if (cab_stat[x] == 0)
          {
             cab = x;
             break;
```

```
    }
}
if (mode_i == 1 && cab != 0)
{
    flag = 1;
    accept_state(i, cab, mode_i);
}
else if (mode_i == 2)
{
    for (int x = 1; x <= n; x++)
    {

        if (cab_stat[x] == 3)
        {
            cab = x;
            break;
        }
    }
    if (cab != 0)
    {
        flag = 1;
```

```
                accept_state(i, cab, mode_i);
            }
        }
        if (flag == 1)
        {
            break;
        }
        sleep(1);
    }
    if (flag == 0)
    {
        printf("TimeOut\n");
    }
}
```

- Check after each second if any cab is available. If a rider wants to book premium the cam should be empty but for pool the cab should either be empty or should have only one pool rider in it.
- Once A cab is available go to the accept function to start the ride and update the mode of the car
- If even after tp time no cab is available then the rider timesout

➢ On ride

- Determines time of the travel

- If mode is 1 so premium ride travel for rtime, ride ends the rider directly goes for payment and cab goes into wait mode
- If mode is 2 the rider with lower time finishes journey and leaves the cab is in mode 3 now
- For mode 3 after each second check for a new ride. If mode becomes two e have two cases
    - If the new rider time is more than the current rider then finish ride of the old one
    - Otherwise the new rider calls the onride function and finishes its journey and leaves

- After completion of ride the cab goes in wait mode and riders go for payment servers