3.UBER

AIM

//As Requse

IMPLEMENTATION

Asumes that the user will give only the number of cabs present and we can treat a cab as premium and pool as peras demand.

In each code car has been assumed as struct .

CODE FOR RIDER

```
void * Bookcab(void * arg)
{
    struct timespec ts;
    if (clock_gettime(CLOCK_REALTIME, &ts) == -1)
    {
        /* handle error */
         perror("ERROR\n");
    }
    r_info *R=(r_info *)arg;
    //printf("%d %d %d \n",R->cab_type,R->max_wait,R->ride_time);
    if(R->cab_type==1)
    {
        int rc;
        ts.tv_sec+=(time_t)R->max_wait;

        // printf("%lu\n",ts.tv_sec);
        // while ((rc = sem_timedwait(&premier, &ts)) == -1 && errno == EINTR)
        //     continue;
        rc=sem_timedwait(&total_cab,&ts);
        if(rc==-1)
        {
            if(errno==ETIMEDOUT)
```

```
            {
                printf("RIDER %d CANNOT WAIT ANY LONGER FOR PREMIER CAB\n",R->rider_n
o);
            }
            else
            {
                perror("SEMAPHOR:\n");

            }
            pthread_exit(NULL);

        }
        else
        {
            int l;
            pthread_mutex_lock(&mutex3);
            for(int i=1;i<=n;i++)
            {
                if(cab[i].state==0)
                {
                     l=i;
                     cab[i].state=1;
                     break;
                }
            }
            pthread_mutex_unlock(&mutex3);
            printf("RIDER %d FINDS THE CAB\n",R->rider_no);
            printf("PREMIER RIDER %d SITS IN THE  CAB %d IS PREMIUM TYPE CAB\n",R->rid
er_no,l);

            sleep(R->ride_time);
            pthread_mutex_lock(&mutex3);
            cab[l].state=0;
            printf("RIDER %d REACHES THE DESTINATION\n",R->rider_no);
            sem_post(&total_cab);
            pthread_mutex_unlock(&mutex3);
```

```
        }



    }
    else if(R->cab_type==2)
    {
        int sleep_time=0;
        int done=0;
        int j;
        pthread_mutex_lock(&mutex3);
        for(int i=1;i<=n;i++)
        {
            if(cab[i].state==0)
            {
                done=1;
                j=i;
                printf("RIDER %d SITS IN THE POOL CAB %d THIS CAB HAS 1 PASSENGERS\n",
R->rider_no,i);
                cab[i].state=2;
                sem_trywait(&total_cab);
                sleep_time=R->ride_time;
                break;
            }
             if(cab[i].state==2)
             {
               done=1;
               j=i;
               printf("RIDER %d SITS IN THE POOL CAB %d THIS CAB HAS 2 PASSENGERS\n",R
->rider_no,i);
               cab[i].state=3;
               sleep_time=R->ride_time;
               break;
             }
        }
       pthread_mutex_unlock(&mutex3);
```

```
        if(done==0)
        {
                int rc;
                ts.tv_sec+=(time_t)R->max_wait;
                rc=sem_timedwait(&total_cab,&ts);
                pthread_mutex_lock(&mutex3);
                ts.tv_sec-=(time_t)R->max_wait;
                if(rc==-1)
                {
                     if(errno==ETIMEDOUT)
                     {
                          printf("RIDER %d CANNOT WAIT ANY LONGER FOR POOL  CAB\n",R->r
ider_no);
                     }
                     else
                     {
                          perror("semaphor:\n");

                     }
                     pthread_mutex_unlock(&mutex3);
                     pthread_exit(NULL);

                }
                else
                {
                     printf("RIDER %d FINDS THE CAB\n",R->rider_no);

                     int i;

                     //int j;
                     for( i=1;i<=n;i++)
                     {
                          if(cab[i].state==0)
                          {

                               //cab[i].no_passenger++;
```

```
                        printf("RIDER %d SITS IN POOL CAB %d\n",R->rider_no,i);
                        cab[i].state=2;
                        sleep_time=R->ride_time;

                        j=i;
                        break;

                    }
                }


            }

            pthread_mutex_unlock(&mutex3);

            sleep(sleep_time);
            pthread_mutex_lock(&mutex3);
            if(cab[j].state==2)
            {
                cab[j].state=0;
                sem_post(&total_cab);
            }
            if(cab[j].state==3)
            {
                cab[j].state=2;
            }

            printf("RIDER %d REACHES THE DESTINATION\n",R->rider_no);
            pthread_mutex_unlock(&mutex3);

        }

        if(done==1)
        {

          sleep(sleep_time);
          pthread_mutex_lock(&mutex3);
```

```
            if(cab[j].state==2)
            {
                 cab[j].state=0;
                 sem_post(&total_cab);
            }
            if(cab[j].state==3)
            {
                 cab[j].state=2;
            }
            printf("RIDER %d REACHES THE DESTINATION\n",R->rider_no);
            pthread_mutex_unlock(&mutex3);

        }

    }
    sem_wait(&server);
    pthread_create(&pay,NULL,make_payment,R);
    pthread_join(pay,NULL);
    sem_post(&server);
}
```

For each rider we create a new thread and then we check if it is premium rider or pool rider .
For a premium rider we check if there is a car
which is in wait state ornot if there is we allocate the rider to the cab otherwise we wait till a given time.This is done using semaphore func. sem_timedwait().

For a pool rider we check if there is pool car available (i.e if there is cab which is pool type it has 1 persom sitting in it)
if there is no such cab then search for cab in wait state .Search till the max time given.
Locks are used to prevent simultaneous access to a particular element .

FUNCTION FOR PAYMENT SERVER

```
void * make_payment(void *arg)
{
    r_info *r=(r_info *)arg;
    pthread_mutex_lock(&mutex);
```

```
    int j;
    for(int i=1;i<=Counter;i++)
    {
        if(payment[i]==0)
        {
            payment[i]=1;
            j=i;
            break;
        }
    }
    pthread_mutex_unlock(&mutex);
    sleep(2);
    pthread_mutex_lock(&mutex);

    payment[j]=0;
    pthread_mutex_unlock(&mutex);
    printf("RIDER %d HAS COMPLETED PAYMENT THROUGH SERVER %d\n",r->rider_no,j);



}
```

After the rider reaches its destination it creates a thread then serch which ever server is free and pay through that server.