Ober Cab Services

There are N cabs, M riders and K payment servers. M, N and K are user inputs and rest of the variables are randomly generated using *rand* function in C. Each rider arrives at a random time and books a cab (either premier or pool). All riders and payment servers are threads. One mutex lock per cab and payment server is used. One mutex lock is used to keep track of remaining riders.

Cabs

- Each cab is a struct.

```
typedef struct cabs
{
    int waiting_status,premier,pool_1,pool_2;
}cabs;
struct cabs cabs_array[10000];
```

- cabStatus denotes the current status of the cab.
    - waitState : can accept both pool and premier customers.
    - onRidePremier : currently riding a premier passenger
    - onRidePoolOne : currently riding a single pool passenger and can accept another pool passenger (priority will be given to it in case a person books a pool cab)
    - onRidePoolTwo : filled pool cab with 2 passengers

  In the beginning, all the cabs have their waitState = 1 and onRidePremier = onRidePoolOne = onRidePoolTwo = 0. where 1 represents the current state of the cab and all the other states become 0.

```
for(int i = 1; i <= m_riders ; i++)
{
    cabs_array[i].waiting_status = 1;
    cabs_array[i].premier = 0;
    cabs_array[i].pool_1 = 0;
    cabs_array[i].pool_2 = 0;
}
```

  Mutexlocks(*pthread_mutex_lock(&lock)* and *pthread_mutex_unlock(&lock)*) are used for changing the status of the cabs. Semaphores are used by the riders to wait until a cab becomes free if all the cabs become busy.

Riders

Implemented using array of threads *riders_threads* i.e for each of the M riders, a new thread is created.

```
pthread_t riders_thread[m_riders+1];
```

- Rider's arrival time is simulated using sleep.
- Wait time is a random number%10.
- Ride time is a random number%10.

```
int wait_time = (rand() % 10);
int ride_time = (rand() % 10);
```

- If the rider is pool, it searches for a poolOne type cab and if it does not find any it searches for wait cab.
- If the rider does not find any cab he sleeps(using semaphore *total_num_of_cabs*) and repeat the same search until it finds a cab or times out if current time exceeds his maxWaitTime.
- After getting a cab he sleeps for ride time and then frees the cab.
- It then invokes *payment_func* function and searches for a free server. On finding a free server, it activates that server(changes status from 0 to 1 in the payment array).

Payment Servers

Implemented using array *payment* and for each of the K servers, a new thread is created.

- It waits for it to be activated. status to change from 0 to 1.
- It then sleeps.
- Payment is then accepted. status is changed from 1 to 0.
- It then decrements the variable total_no_of_riders(keeps track of remaining riders).