# Process Synchronization

## Ober Cab Services

## Problem statement

Ober is a new cab service, which needs your help to implement the simple system. The requirements of the system are as follows given N cabs, M riders and K payment servers you need to implement a working system which ensures correctness and idempotency. Each cab has one driver. In Ober, payments by the riders should be done in Payment Servers only. Ober provides two types of cab services namely pool and premier ride. In pool ride maximum of two riders can share a cab whereas in premier ride only one rider. There are four states for a cab namely waitState (no rider in cab), onRidePremier, onRidePoolFull (pool ride with two riders), onRidePoolOne (pool ride with only one rider).

### Introduction

We need to implement some functionalities of rider, driver and payment server using threads, semaphores and mutex locks.

### Requirements

1. **pthread.h** for threads
2. **semaphore.h** for semaphores
3. **sys/sem.h** for semaphore timedwait

## Functionalities implemented

### Rider

1. BookCab
2. MakePayment

### Driver

1. AcceptRide
2. OnRide
3. EndRide

### Payment server

1. AcceptPayment

## Semaphores used

1. *waitcabsem*
2. *poolonecabsem*
3. *poolfullcabsem*
4. *premiercabsem*
5. *paymentserversem*

## Mutex locks used

1. *paymentmutex*
2. *progressmutex*
3. *driverstatemutex*
4. *queuemutex*

## Compiling the code

*gcc -g <filename> -lpthread*

## Input format

*N(Number of cabs/drivers), M(Number of riders), K(Number of payment servers)*

*(**Optional. Input can be given or generated randomly**) For each rider,*

*Arrival time, cab type(1->POOL, 2->PREMIER), Max wait time, Ride time*

## Implementation

We create a thread for each rider with arguments including riderid, arrival time of rider, cab type, maximum waiting time of rider and ride time. We sleep for arrival time of rider in rider thread so that rider becomes active after span of arrival time.

If the cab type ordered is POOL, then we need to first check whether there are any cabs with state onRidePoolOne, if yes we assign this rider to this cab or else we assign a waiting cab and driver to this rider. Is there is a cab with state onRidePoolOne, then we increment poolfullcabsem and decrement poolonecabsem. If we assign rider to empty cab we increment poolonecabsem and decrement waitcabsem. While exiting from thread after completion of ride, we change semaphores accordingly.

If the cab type ordered is PREMIER, then we need to assign a waiting cab and driver to this rider. Here, we increment premiercabsem and decrement waitcabsem. While exiting from thread (completion of ride), we change semaphore accordingly.

After this, we sleep for rideTime and then call makePayment() function. In makePayment() function, we add payment details to linked list (riderid and cabType).

In acceptPayment() function of payment server, we take the payment nodes from the linked list and complete the payment for that respective rider and update the payment progress through a array paymentprogress. So, in makePayment function, when this paymentprogress array is updated, we print suitable statement and exit.