REPORT DETAILS:

- CS21BTECH11011
- B.Asli Nitej Reddy

First let's start with main:

Here 1st I am taking the input from the inp-params.txt for the variables named p , c , lp , lc , k which contains total no of passengers , total no of cars , the parameter for the exponential wait between 2 successive ride requests made by the passenger, the parameter for the exponential wait between 2 successive ride request accepted by a car , the number of ride request made by each passenger respectively

The creating a file named output.txt if not existed or truncating if already existed for containing the log for various details

Let's see what are the global variables declared other than these discussed above

```
int *pa;
int *ca;

// used for storing and checking car availability
queue<int> cars_queue;

sem_t *sem_cars;  // semaphore for available cars
sem_t *pas_mutex; // Semaphore for mutual exclusion between passenger processes
```

pa and ca pointers are for storing various time related to graphs semaphore pointers which will be initialized in the main function also a queue for storing which car is available and which is not available in the order they arrives

using named semaphores as (sem_init is not available in mac) so using sem_open() to initialize the semaphores like below code

```
    const char *sem_name1 = "/semaphore1";
    sem_cars = sem_open(sem_name1, O_CREAT | O_EXCL, 0644, c);
```

Then using the p creating an array of pthread_t which is a datatype used to uniquely identify a thread and the name of array of pthread_t is thread_pass

Then we used pthrtead_attr_t attr here what's happening is the attr argument points to a pthread_attr_t structure whose contents are used at thread creation time to determine attributes for the new thread

Now pthread_attr_int(&attr) is used here the function like type pthread_attr_int() initializes the thread attributes object pointed to the by attr with default attribute values

Now I am using for loop to loop and create the thread with help of pthread_create()
(I am looping p times)

The pthread_create() function starts a new thread in the calling process the thread starts execution by invoking start_routine();
arg is passed as sole argument start_routine() and pthread_create() looks like the down thing

```
int pthread_create(pthread_t *restrict thread,
                   const pthread_attr_t *restrict attr,
                   void *(*start_routine)(void *),
                   void *restrict arg ) ;
```

in our case it looks like

1.pthread_t *restrict *thread* = &threads_pass[i]
2.const pthread_attr_t *restrict *attr* = &attr
3.void *(**start_routine*)(void *) = passengers_and_car
4.void *restrict *arg* = &thread_ids_pass[i]
(it contains the passenger id not more than that)

In each for loop we are doing:

thread_ids_pass[i] will contain the value like i + 1

pthread_create(&threads_pass[i] , &attr , & passengers_and_car, &thread_ids_pass[i])

The above thing will run 0 to (p – 1) times

after seeing what's happening in the passengers_and_car function we will comeback to main to discuss the statements happening after returning from start_routine

Now let's discuss what's happening in the passengers_and_car function:

here 1'st we are starting the start time which is very important like below code

```
// starting time for each thread
auto starttime = high_resolution_clock::now();
```

then a random generator for the roaming time and the time the person stays in the car with the help of below code

```
std::random_device rd; // True random number generator
std::mt19937 gen(rd());

double a = lp;
a = 1 / a;

int rides = k;

double b = lc;
b = 1 / b;

exponential_distribution<double> t2(b); // critical section time
exponential_distribution<double> t1(a);
```

and we can get the random time with the help of t2(gen) or t1(gen)

now after the above code I am using the auto stop to check when the person enters the museum and printing it into the output.txt file

now comes main part that is the start of for loop like it will be looped k times which is number of times the passengers takes the ride

in the each for loop at start we are using the below code as roaming time like the time the person wanders in the museum

```
usleep(1000000 * t1(gen)); // it's more like before 1st ride also wandering
```

then after this the passenger will make a request for car so here comes the semaphore which will be a counting semaphore with max value as c which is nothing but number of cars in that museum

then we are printing the time that is after executing the usleep() which is nothing but the passenger request for a car into the output.txt

```
sem_wait(sem_cars);
// .... various details explained below
sem_post(sem_cars);
```

in the above if all cars are not available if now any passenger is requesting for a new car the he will be in the waiting until a car is available which incase will be available after some thread uses the sem_post(sem_cars);

Now let's see what are the things we are doing inside the semaphore

After entering the semaphore we are calculating time at which a car accepted our request

But now we don't know which car accepeted that passengers request so we are using a new semaphore which is a binary for mutual exclusion in the current function to get the car number without any race conditions like below

```
sem_wait(pas_mutex);
  carnumber = cars_queue.front();
  cars_queue.pop();
sem_post(pas_mutex);
```

in the above code we are retrieving the car number and popping it out of the queue so that we making it like the car is now not available and when we are doing this no other thread can access the car number as we are enclosing it with a binary semaphore named pas_mutex

now as we got the car number also we are now printing the information related to passenger and which car accepted it along with time into the output.txt with help of a string

now indirectly we are in critical section so now again we are sleeping the threads with the time which is equal to the time he ride in that car and that time we will get based on below code

```
// the time in the car is this value
usleep(1000000 * t2(gen));
```

after executing the above code again we are calculating the time duration from the start to print it in the output.txt as this is nothing but the time at which passenger completed his ride so now we have to add the car to the queue before the sem_post(pas_mutex) and we are doing it like down

```
sem_wait(pas_mutex);
   cars_queue.push(carnumber);
   ca[carnumber] = duration.count();
sem_post(pas_mutex);
```

In the above we used the same binary semaphore as we are dealing with queue which is done in above also and also storing the time at which this car became free into an array which we defined in the main this will be used in the future

Now to store the time at which a passenger completed his all tours and exits the museum we dealing this with help of if condition that is after completing his last ride (in main for loop value is == k – 1)

In the for loop when i == k – 1 and in the last part of for loop I am using the below code

```
if ( i == k - 1 )
    { // calculating the time after passengers last ride
       pa[idp] = duration.count();
       strp = "Passenger " + to_string(idp) + " exits museum at " +
              to_string(duration.count()) + "ms from the start \n";
       op_2 << strp ;
    }
```

So with above help I am storing the exit time into array named pa which we initialized in the main function these value are used for calculating the average time a passenger to complete his tour

And finally using the sem_post(sem_cars) so that we are saying that a car is now incremented in the availability so that any thread which is waiting in sem_wait(sem_cars) can now get that and the passengers_and_car() function is now fully explained and we will now return to main function

Back to main function:

after creating the threads in the main function we will immediately use the below code which is a for loop

```
   for (int i = 0; i < p; i++)
   {
       pthread_join(threads_pass[i], NULL); // joining threads
   }
```

And let's see what will above code do in our case

In the for loop we are using a function named pthrtead_join() it's like down thing

int pthread_join (pthread_t thread , void ** retval)

1. pthread_t thread = thread_pass[i] and
2. void** retval = NULL

the main use of pthread_join is it waits for the thread specified by thread to terminate and also

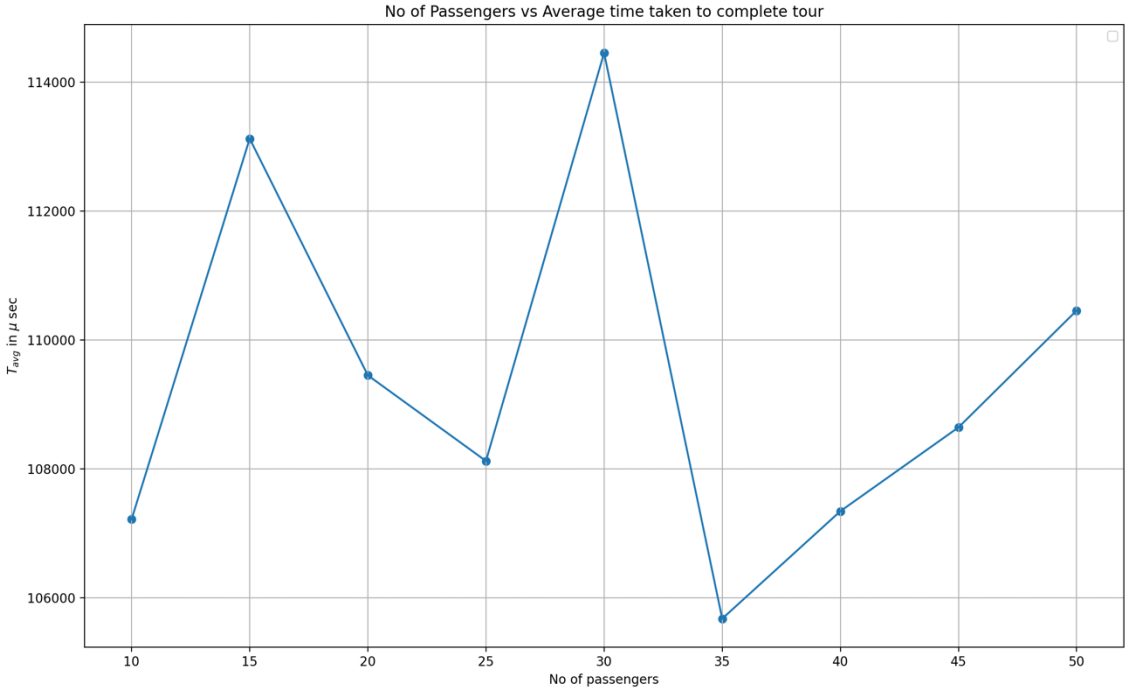
if retval is not NULL then pthread_join() copies the exit status of the target thread (i.e., the value that the target thread supplied pthread_exit(3)) into the location pointed to by retval which is nothing but the array we discussed before in our case we need not worry as it is NULL here

What will output.txt file contains:

Passenger 14 enters the museum at 206 ms from the start
Passenger 14 made a ride request at 138856 ms from the start
Car 1 accepts Passenger 14 request
Passenger 14 started riding at 138889ms from the start
Car 1 is riding Passenger 14
.
.
.
.
.
.
Car 2 has finished Passenger 11 tour
Passenger 11 finished riding at 10376707ms from the start
Passenger 11 exits museum at 10376707ms from the start

Graph1 Analysis: No of Passengers vs Average time taken to complete tour:

Taking
The x-axis contains the number of passengers. (Varying from 10 to 50 with an increment of 5.)
The y-axis represents the average time taken by the passengers to complete their tour.
The number of cars and K should be fixed to 25 & 5 respectively.



Justification:

As number of cars here are 25 and no of passengers varying from 10 to 50 with increment of 5 in my case as I took the lp = 10 and lc = 8

So in my test case lp and lc are very close and also the maximum no of passengers is less than 50 that is 2*25(no of cars) which will affect the graph greatly

And let's see why values in the graph are very close (graph may not look like that but the graph showing is very zoomed in)

And the reason is here maximum request can be for passengers = 50 where they can make 50 requests but see after the completion of 25 cars still 25 passengers are left for their 1st ride

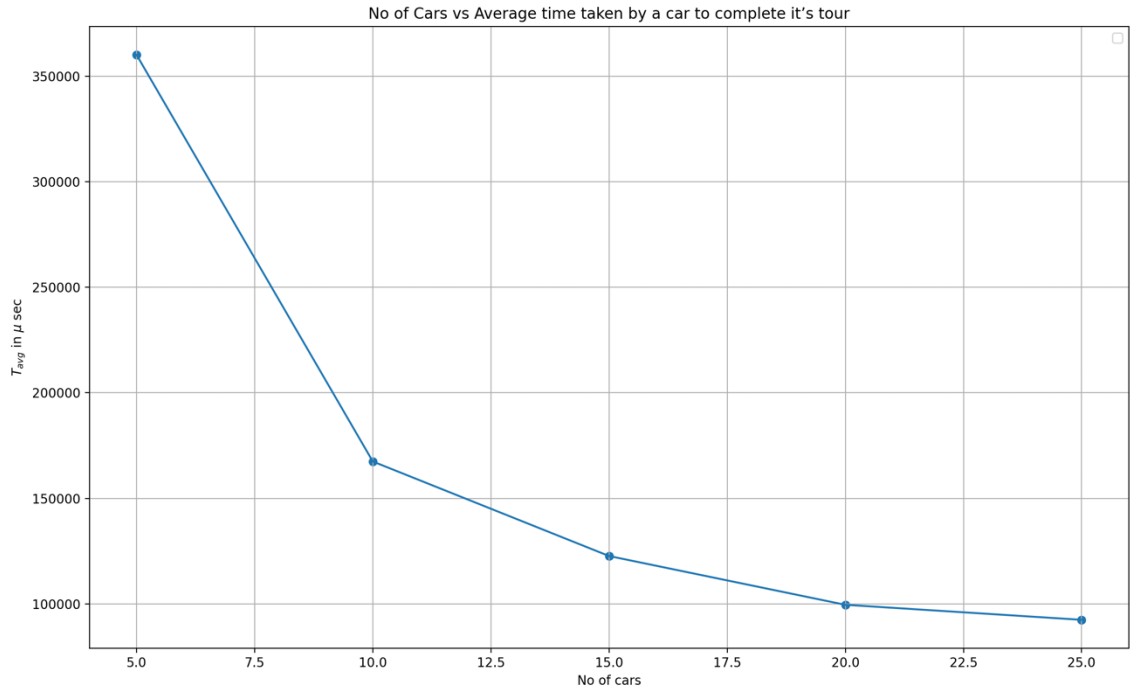
After the 1st 25 members who took their ride got completed then again nearly all this 25 people must wait for the roaming time while most of other 25 members have completed their roaming time already, they will go into the semaphore and locking it again while they complete their 1st ride the 1st squad should have completed their 2nd roaming time outside and will be waiting for the semaphore to unlock

So the loop keeps on going with small variations where some person got more roaming time while the other person who just entered the the car has taken ride and completed it and also again completed his roaming time before the 1st so he may go before the 1st person but these are very small things can be neglected

Hence the above graph explains the very thing we discussed above as long as passengers are <= 50 but more than that we will see the graph will be having steady increase

Graph2 Analysis: No of Cars vs Average time taken by a car to complete it's tour:

Taking
The x-axis should contain the number of cars. (Varying from 5 to 25 with an increment of 5.)
The y-axis represents the average time taken by the cars to complete its tour.
The number of passengers and K should be fixed to 50 & 3 respectively.



Justification:

From the graph we say that as no of cars are increasing their avg time taken to complete the tour is decreasing the reason is

The reason is as a car is increased there will be a passenger who is in ready queue and wants to have a ride here that extra car can give the ride so there will be less waiting time after a person requested for a ride.

So, it's more like this car is taking burden of another car which should have taken so total avg time taken by a car to finish it's tour will be decreased