PROGRAMMING ASSIGNMENT-4 THE JURASSIC PARK PROBLEM

DASARI SRINITH
CS21BTECH11015

REPORT:

OVERVIEW OF CODE:

- The code for the assignment takes 5 inputs ; P, the number of passengers threads; C, the number of car threads; lam1 , the parameter for the exponential wait between 2 successive ride requests made by the passenger, lam2 , the parameter for the exponential wait between 2 successive ride request accepted by a car and k, the number of ride request made by each passenger.
- The code uses two queues to add available cars , available passengers and two mutexes related to them ,i.e locking the queue so that no other thread can accesses them and change them at the same time.
- The code also keeps two counting semaphores , one for available passengers , it is initialized to 0 and one for available cars , it is initialized to number of cars .
- And also the code has a globally defined array of semaphores(actually used as mutexes) with size equal to number of cars which will be used later.
- Initially , we take input from a file "inp.txt" and store them in global variables.
- Then we create and allocate memory for the threads.
- Here i am using the same thread struct for both the passenger and car threads ,(although the feature long double* ele is not needed for passenger threads)
- In the thread struct , vector pointer named "times" is used to store the values of time events which will be used later to print the output file.
- NOTE : I have printed the output file , after ordering the times data , so that it would be easy to see the events , even though the order is asynchronously printed.
- The struct also has a vector pointer named "pairs" which would store the pair with which a car or passenger is matched after requests are made and accepted.
- So , we will create p passenger threads which will run of passenger function and create c car threads which will run on car function
- In the passengers function ,
  - After we enter , we push the value of entry time of passenger into the vector

  - Then we sleep for a random time simulating wandering around the museum
  - Then we push the instance of time at that point as the request time for a car.
  - We add the passenger into the available passengers queue and increase the semaphore value .
  - While doing the above we use a mutex for passengers queue, to avoid the race conditions.
  - Then the passenger thread will wait on the available cars semaphore , and after it enter it will push the current time instance into times vector as the accepting time.
  - Then if the available cars queue is not empty we would pop the top one , and pair it with our passenger and push it into pairs vector.
  - This is also done with use of another mutex for cars queue , to avoid the race conditions.
  - Then we also wait a semaphore acting as a personal mutex for a car thread , i,e ensuring that the passenger thread waits there until the car thread completes the ride and calls the sem_post function on that semaphore.
  - Then after that we store that time instance into the times vector as the end time.
  - finally at the end , we push the exit time into the array as well.

- In the Car function ,
  - We loop in the car thread infinitely waiting on the available passengers semaphore , and we break of the while loop , if the value of global variable is changed to -1.
  - Here , to much sure not thread is infinitely struck because of waiting on the available passengers semaphore even though the value of global variable is changed, we increase the semaphore values , so it can go the global variable checking condition.
  - Coming back , if the available passengers queue is not empty , we would pop the first passenger and pair it with our car, (NOTE:THIS PAIR WOULD BE THE SAME IN PASSENGER MENTIONED ABOVE)
  - We would also lock the personal mutex , so that the passenger thread would have to wait until the car thread finishes its ride (i,e sleep for a random time)
  - Then we sem_post the personal mutex value after the sleep is done
  - Then we find the time the thread slept and add it to a variable , later used to find the average time taken by car to complete the tour.
  - Here also we use the necessary mutexes to avoid race conditions.

- So , after all the passenger threads are done , we change a value of global variable and sem_post(semavaipass) so that if any car thread is waiting , then it would come out and break from the while loop and exit.

- Then from the times and pairs vector into which we inserted the values of threads, we print them into output file "output.txt" in a specific order .

- And also , we find the average time taken by a passenger to complete the tour by ,
  - subtracting the exit time from the museum and enter time into the museum
  - adding the values of all the passengers and then dividing by the number of passengers.

- We find the average time taken by car as follows ,

  - we substract the two time instances before and after the sleep used for simulating ride time
  - add them all until the car thread exits and then store it into long double* arr
  - then after all the car threads exit, we add the values of all the cars and divide it by the number of cars

- Finally we also free the memory we allocated

ANALYSIS:

1. **EXPERIMENT 1 :**

   - For this experiment I have taken the values of lambda's to be 5(λ_P) and 20(λ_C) respectively.
   - In this , we are varying the number of passengers from 10 to 50 with imcrements of 5 and keeping the number of cars as 25 and value of k as 5 constant through out the experiment.
   - The following is the plot.

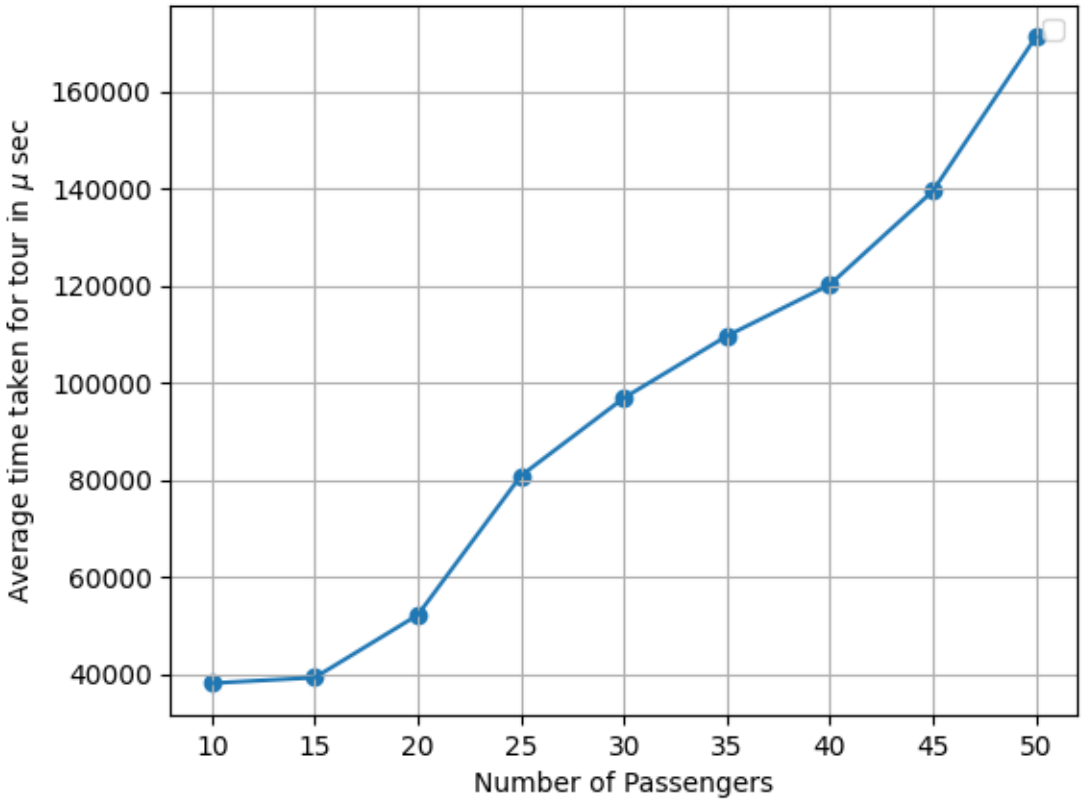

Figure 1: Average time taken for tour Vs Number of Passengers

   - As from the plot we can see that as the number of passengers increases ,the average time taken by a passenger does not increase by much for initial values of passengers.
   - This is because as we took the number of cars to be 25 , for number of passengers less than 25 , we can say that at a time all the passengers can be guaranteed a car if needed .
   - So , that whenever a request is made by a passenger , there is almost no waiting time for passengers .
   - And from the graph we acn see a higher increase at 25-30 passengers and then increase of average time as number of passengers increases.

- Because as the number of passengers increase to more than or equal to 25 , we can say that there can be cases where the passengers have to wait for the car , and which would result in the increase of average time take to complete the tour compared to the previous situation.

2. **EXPERIMENT 2:**

- In this , we are varying the number of cars from 5 to 25 , with increaments of 5 and keeping the number of passengers to be 50 and the value of k as 3 constant through out the experiment.
- The following is the plot

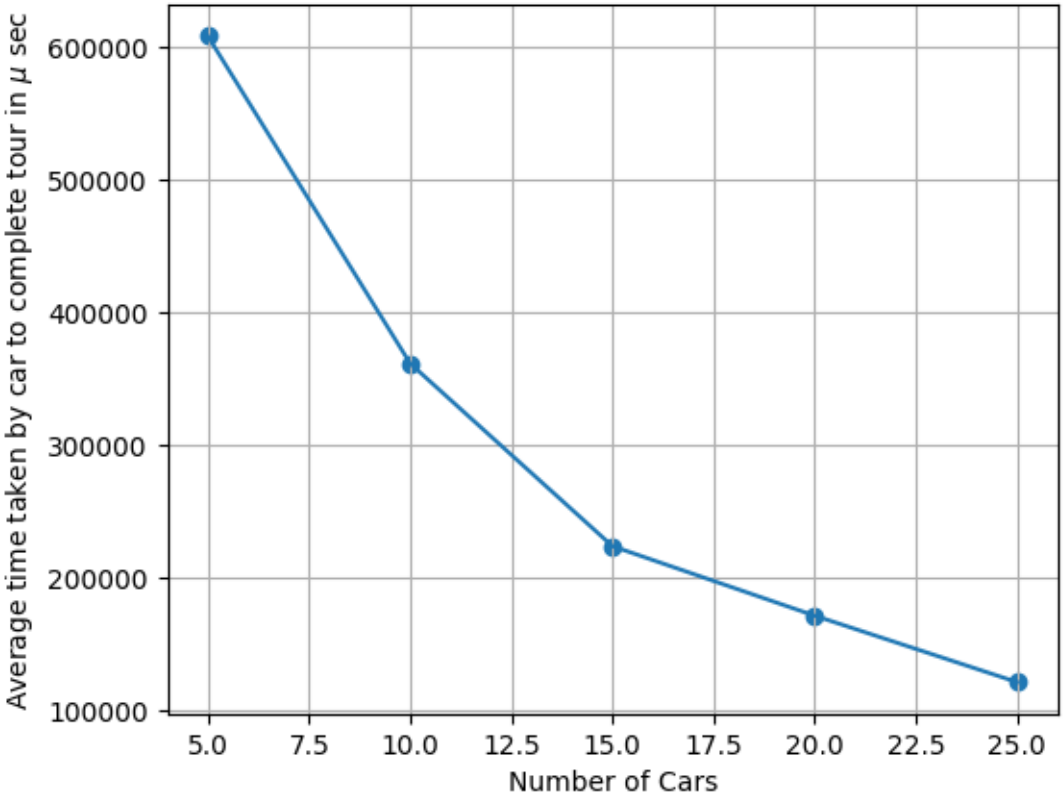

Figure 2: Average time taken by car Vs Number of Cars

- NOTE : Here we are considering the time taken by car to complete the tour to be the sum of ride times of the car and average value would be the average of all the cars.
- As from the plot we can see that as the number of cars increases , the average time taken by cars to complete the tour decreases through out.
- This is because, in the formula of finding the average value , as the number of passengers remain constant , the numerator doesn't change much (as numerator is sum of ride travel times , which is random but sum would be effectively same if we take same lambda value) ,and the number of cars increases so the denominator(number of cars) increases , so the average time would decrease.
- From the above point we can also notice that the graph is of the form y = c/x .