Programming Assignment-4 Report

Prasham Walvekar | CS21BTECH11047

LOW-LEVEL DESIGN:

Overall low-level explanation of what the code does:

The given code is an implementation of a simulation of a museum ride with multiple cars and passengers using pthreads and semaphores. The program is designed to take input values for the number of passengers, cars, and ride duration. The program creates threads for each passenger and car and simulates the ride by allowing passengers to ride on available cars.

The main function starts by parsing command line arguments and setting variables for the number of passengers, cars, and ride duration. It then initializes semaphores for each car and the number of available passengers. It also initializes a queue to keep track of available cars.

After initialization, the program creates threads for each car and passenger and runs the simulation. Each passenger thread represents a passenger visiting the museum, while each car thread represents a car providing rides to passengers.

The passenger thread begins by generating a random duration for the passenger to wander around the museum. After the passenger is done wandering, the thread generates a ride request and checks if any cars are available. If no cars are available, the thread waits until a car is available. Once a car is available, the passenger gets on the car, and the ride simulation starts. The passenger thread keeps track of the entry and exit time of the passenger from the ride.

The car thread waits for a ride request from a passenger. Once a request is received, the car thread picks up the passenger and starts the ride simulation. After the ride is complete, the car thread becomes available again.

The simulation continues until all passengers have completed their ride. Once all passengers have completed the ride, the program terminates.

The program also logs each event during the simulation into a file. At the end of the simulation, the program calculates the average tour time for each passenger and car. To ensure thread safety, the program uses semaphores to control access to shared resources, such as the car queue and the output file.

Design of the code:

The design of the code can be described as follows:

1. Global variables:

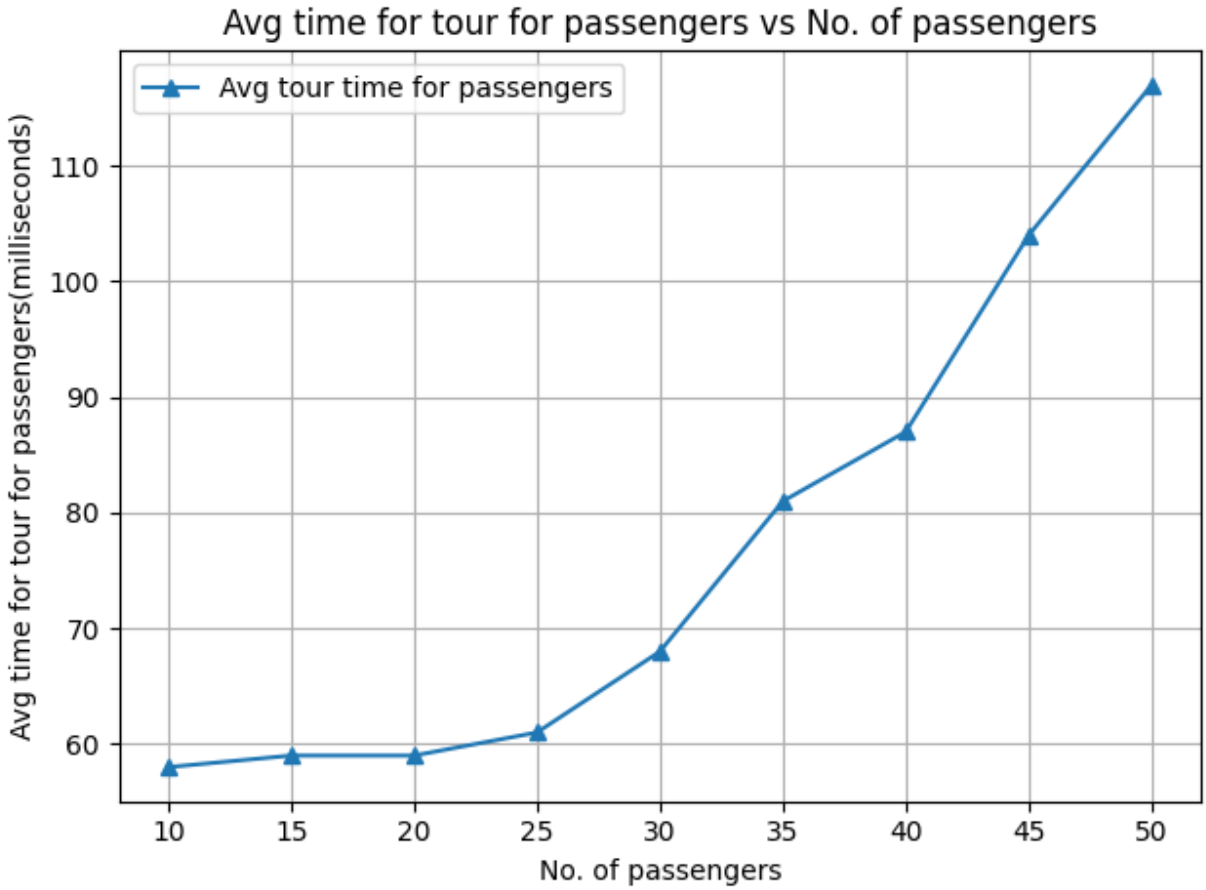
    - *P*: the number of passengers
    - *C*: the number of cars
    - *k*: the number of tours each passenger takes
    - *lambdaP*: the parameter for the exponential wait between 2 successive ride requests made by the passenger
    - *lambdaC*: the parameter for the exponential wait between 2 successive ride request accepted by a car
    - *no_of_passengers_available*: the number of available passengers
    - *num_passengers*: a semaphore to keep track of the number of available passengers
    - *car_sem*: an array of semaphores to keep track of available cars
    - *carq*: a semaphore to ensure thread safety of the car queue
    - *fp2*: a file pointer to output the simulation results
    - *file_write*: a semaphore to ensure thread safety of file writes
    - *start_time*: a time variable to keep track of the start time of the simulation
    - *passenger_tour_times*: an array of Time structures to keep track of each passenger's tour time
    - *car_tour_times*: an array to keep track of each car's tour time
    - *passenger_avg_tour_time*: the average tour time for all passengers
    - *car_avg_tour_time*: the average tour time for all cars

2. Data structures:
    - *Pass_Input*: a struct to hold the input for a passenger thread
    - *Car_Input*: a struct to hold the input for a car thread
    - *Time*: a struct to hold the entry and exit times for a passenger

3. Functions:
    - *passenger*: a function to simulate a passenger's behavior
    - *car*: a function to simulate a car's behavior

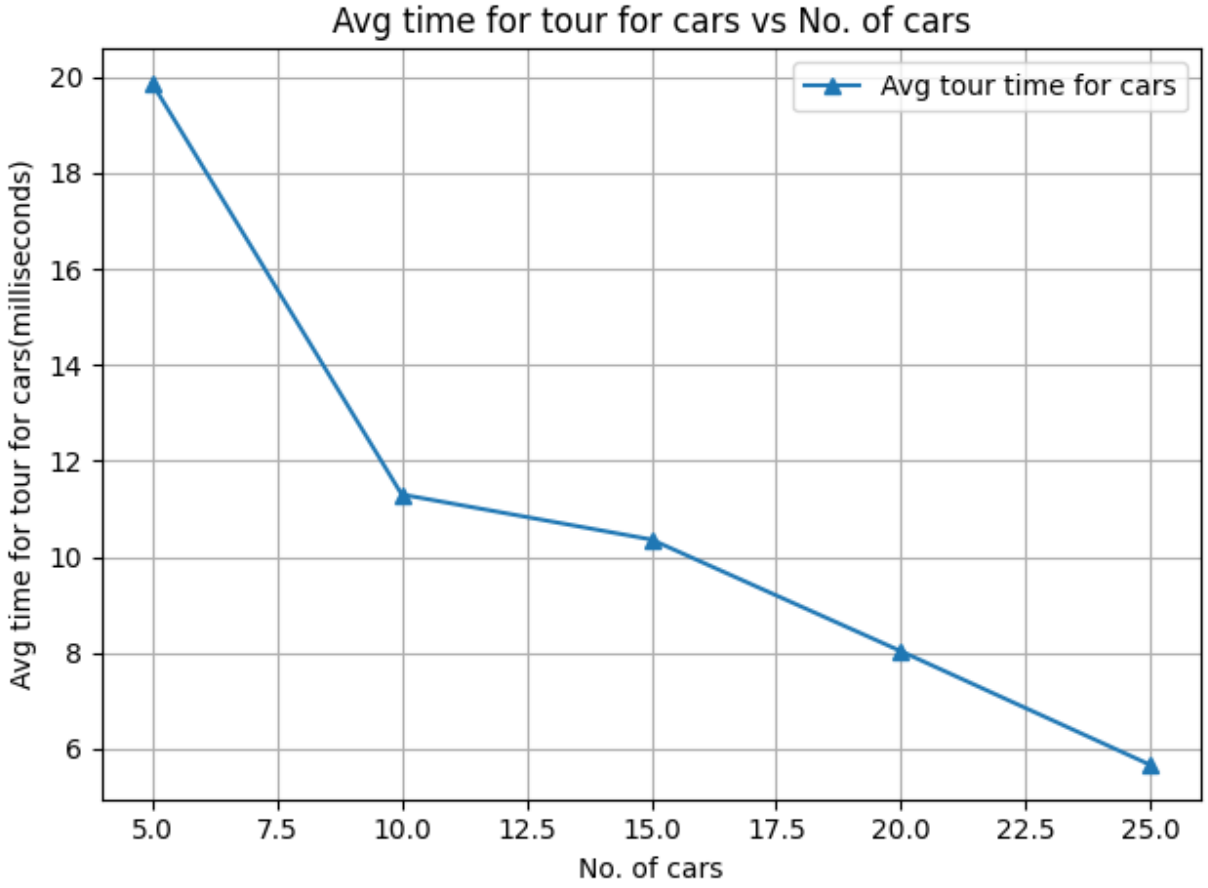
ANALYSIS:

1. **Average tour time for passengers (in milliseconds) vs No. of passengers (n)**
   The value of C, k have been fixed to 25 and 5 respectively. No. of passengers (n) is varied from 10 to 50.



The average tour time for passengers increases gradually from no. of passengers = 10 to 25. After 25, it increases at a larger and almost constant rate.
This is because we have fixed the number of cars to 25. So till no. of passengers < 25 the cars are in excess and there is almost a 1:1 ratio of passengers to cars, so the average time taken for passenger tour almost remains constant (since waiting time for the passengers would almost be 0) for no. of passengers < 25.

2. **Average tour time for cars (in milliseconds) vs No. of cars (n)**
   The value of P, k have been fixed to 50 and 3 respectively. No. of cars (n) is varied from 5 to 25.



The average tour time for cars decreases as the number of cars increases. This happens because since the number of passengers and rides per passengers are fixed, so total rides are fixed and ride time is almost constant, hence average tour time for cars decreases since we divide a constant value by C, which increases from 5 to 25, thus the overall graph is decreasing. This can also be thought intuitively since the number of passengers are fixed, so as the number of cars increases, each car would have to go on lesser number of rides so the average car tour time must decrease with increase in the number of cars.