Design

This is a C++ program that simulates a Jurassic park ride system that includes cars and passengers.

The program starts by including necessary libraries such as iostream, fstream, chrono, random, thread, semapore.h, and queue. It also declares some global variables such as avg, tc, tp, tr, k, c, p, out_file, sem_c, sem_p, sem_q, and q. A queue of available cars is maintained as a global variable.

The time() function is declared to get the current time.

The passenger() function is defined to simulate a passenger. Inside the function, the current time is recorded when the passenger enters the museum, and then the passenger waits for a car to become available. Once a car is available, the passenger rides it for k times. During each ride, the passenger records the start and end times and waits for the car to finish. The function calculates the average time spent by passengers to complete their tour and exits the museum.

The car() function is defined to simulate a car. The function waits (through semaphore) for a passenger to make a ride request and then waits for a time tr to simulate the time the passenger spends riding the car. After the ride is over, the car waits for a time tc to simulate the time it takes for the car to be ready for the next ride.

The main() function is defined to read input parameters from a file and initializes the semaphores and queues. It then creates threads for cars and passengers and waits for all passengers to complete their tour. After all the threads have completed their execution, the program calculates and displays the average time taken by passengers to complete their tour.

Finally, the program closes the output file and exits.

sem_c is used to limit the number of cars running at the same time. It is initialized with a value of c, which represents the maximum number of cars that can run at the same time. The car() function waits on sem_p which allows the car to start running. When the car finishes its ride, it signals sem_c by incrementing its value, which indicates that it has completed it’s ride.

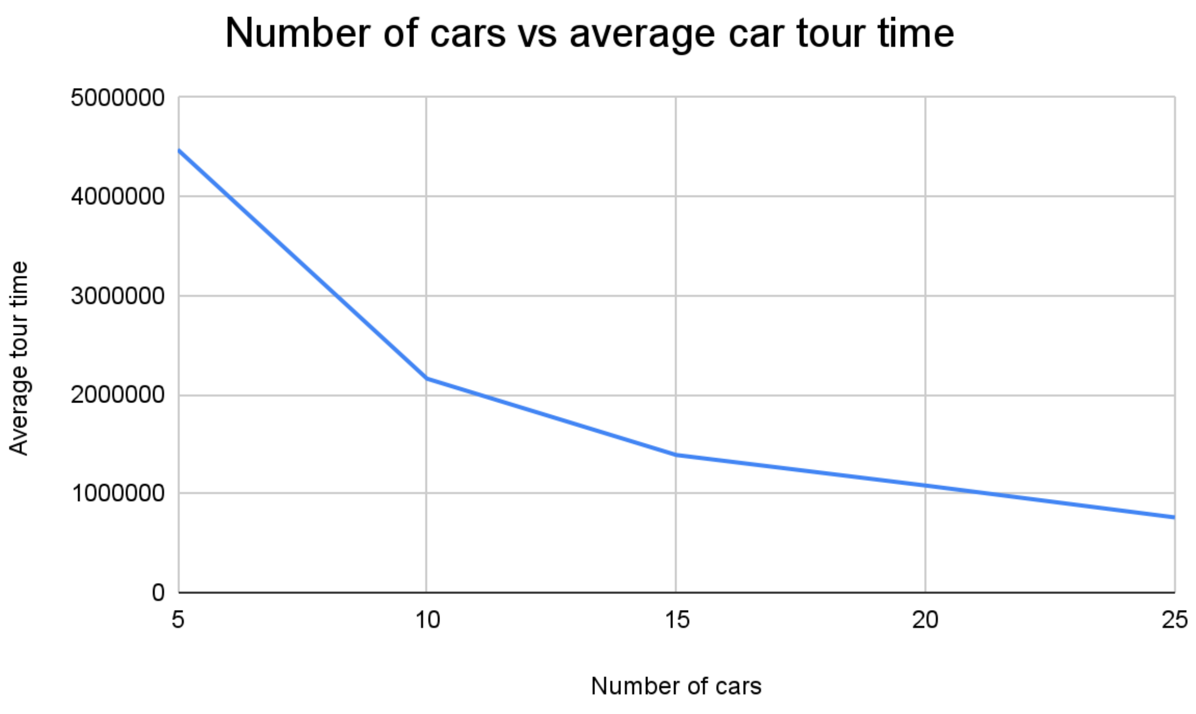
sem_p is used to indicate that a passenger is waiting for a ride. It is initialized with a value of 0. Each time a passenger wants to ride, it signals sem_p to increment its value.

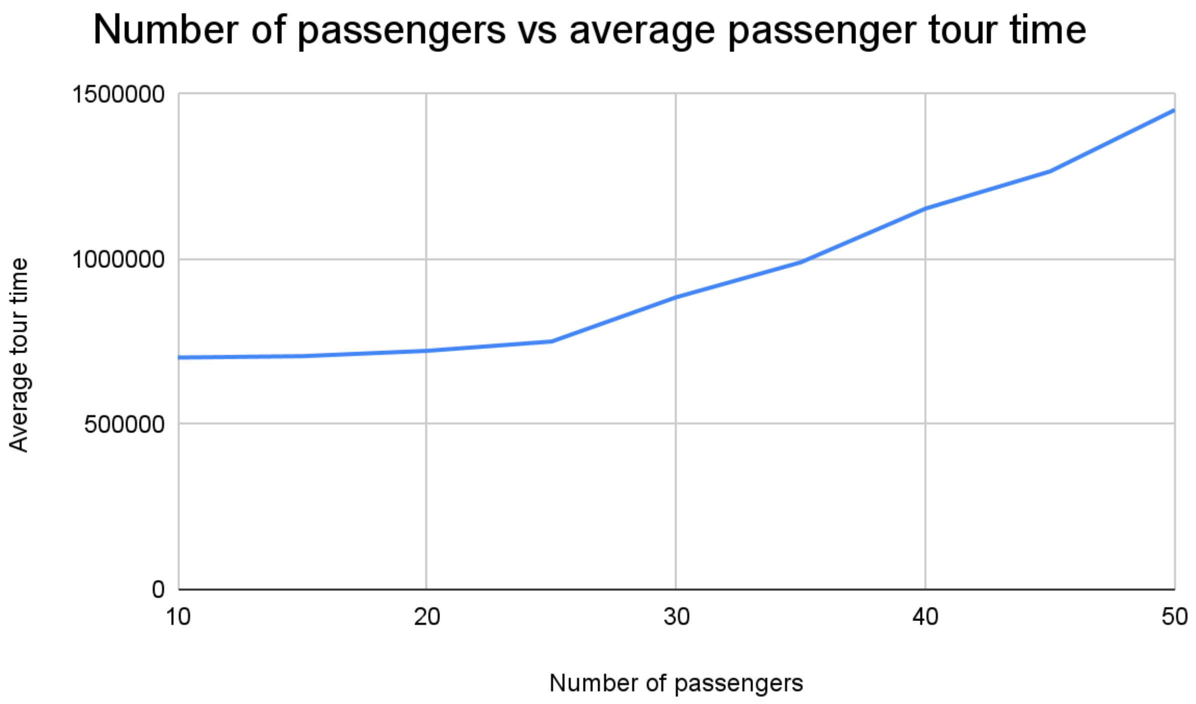
When a car is ready to accept passengers, it waits on sem_p to decrement its value, which allows it to select a passenger to ride.

sem_q is used to control access to the queue of available cars. It is initialized with a value of 1, which means that only one thread can access the queue at a time. When a car finishes its ride, it signals sem_q to release the lock on the queue, which allows another car to select the next passenger to ride.

By using semaphores, the program ensures that cars and passengers are synchronized and that there are no race conditions when accessing shared resources like the queue of available cars.

Analysis of results and plots





As one would expect, as the number of cars is increasing, the average car tour time is decreasing. This is because more cars means each car has to serve a lesser number of rides.

As the number of passengers is increasing, the average passenger tour time is increasing. This is because passenger threads are waiting longer on sem_c.

This happens because the code is designed such that sem_c is signaled by car thread but passenger waits on sem_c and sem_p is signaled by passenger although car waits on it. Thus there is an inverse relationship between passenger tour time and number of passengers as each passenger has to wait longer for a car to become available.