OS 2 - Programming Assignment 4
The Jurassic Park Problem

CS21BTECH11051
Rajiv Shailesh Chitale

Abstract:
In Jurassic Park, passengers wander around the museum for a while, then line up to take a ride in a safari car. When a car is available, it loads the one passenger it can hold and rides around the park for a random amount of time. If there is no passenger available, the car waits and vice versa. This report examines the program to synchronise the P passenger processes and C car processes using semaphores.

Code Design:

Summary of functions:

➔ main() - calls input(), initialises and destroys semaphores, creates and joins P passenger threads as well as C car threads, calculates average times.

➔ input() - reads input parameters from input file

➔ output() - appends given string to output file

➔ genExpVariable() - returns a sample from exponential distribution with a given mean.

➔ getSysTime() - returns string with time in mm:ss:xxx format (x is milliseconds). If a pointer is passed as an argument, it will set its value to the current timestamp.

➔ passengerThread() - simulates events (exploring the museum and safari, k times) and generates logs with timestamps. Requests and waits for a car ride using semaphores. Also waits for the signal of the ride ending. Stores its wait time.

➔ carThread() - waits and accepts a request from a passenger using semaphores. Simulates ride time. Signals the ride ending. Generates logs and updates its total ride time.

Input parameters:

P - number of passengers
C - number of cars
lambdaP - mean wait time at museum
lambdaC - mean wait time of safari
k - number of safari rounds per passenger

Binary semaphores used for synchronization between C and P:

availableCar - signals availability of a car (initially 0)
availablePassenger - signals availability of passenger (ie. a request for a car) (initially 0)
cmutex - acts as mutex for multiple available cars (initially 1)
pmutex - acts as mutex for multiple available passengers (initially 1)
rideEnd[C] - array of semaphores to signal end of a ride (initially 0)

Pseudocode for passengerThread() synchronization:

```
for k times
{
  Sleep based on lambdaP              // time at museum

    wait(pmutex)
        write pid
        signal(availablePassenger)    // make request with passenger id
        wait(availableCar)            // wait for car to fulfill request
        read cid                      // read car's id
    signal(pmutex)

    rideEnd[cid].wait                 // wait for end of ride from that particular car
}
```

Pseudocode for carThread() synchronization:

```
while(toursComplete == false)
{
   wait(cmutex)
        wait(availablePassenger)          // wait for passenger request
        if(toursComplete==true) break;    // used for termination of car threads
        read pid                          // read passenger's id
        write cid                         // provide own car id
        signal(availableCar)              // fulfil request
    signal(cmutex)

    sleep based on lambdaC                // time for safari

    rideEnd[cid].signal                   // signal end of ride
}
```

Note 1: Semaphores are used to implement mutual exclusion in a way that only one passenger can make a request at a time while others have to wait. Similarly, only one car can accept a request at a time.

Note 2: at the end, carThreads will get stuck on the wait for available passengers. The bool toursComplete is set to true. Then availablePassenger is signalled P times by main(). The car threads end their waiting, satisfy the following if statement and break out of the while loop. In this way, the car threads are terminated.

Graphs:

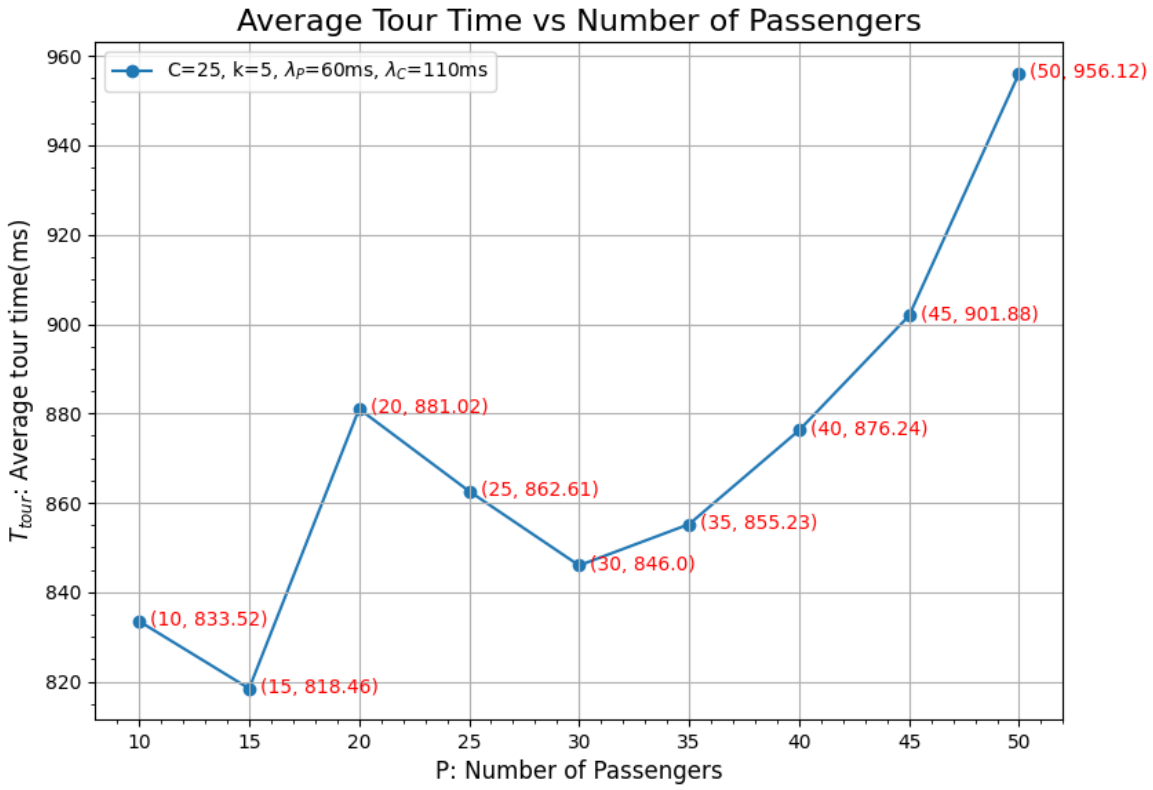
Plot1. Graph of average tour time of passengers vs number of passengers (P).

P varies from 10 to 50. C = 25, k = 5, lambdaP = 60 ms, lambdaC = 110 ms are constant.

Table of readings of average tour time of passengers. (Averaged over 5 readings)

Number of passengers (P)	Average Tour Time (ms)
10	833.52
15	818.46
20	881.02
25	862.61
30	846.0

Number of passengers (P)	Average Tour Time (ms)
35	855.23
40	876.24
45	901.88
50	956.12



Analysis:
The average tour time of passengers increases from P=30 to P=50.
With given parameters, the expected time of a ride is longer than the expected time between rides. When the number of passengers P exceeds the number of cars C=25, passengers start queueing up for a car ride. With C fixed, increasing P causes a greater ratio of passengers to wait. This increases the average waiting time and hence the average tour time. Moreover the higher number of processes slow each other down at the mutex parts.

In general, the waiting time also increases as more passengers try to access the semaphore.

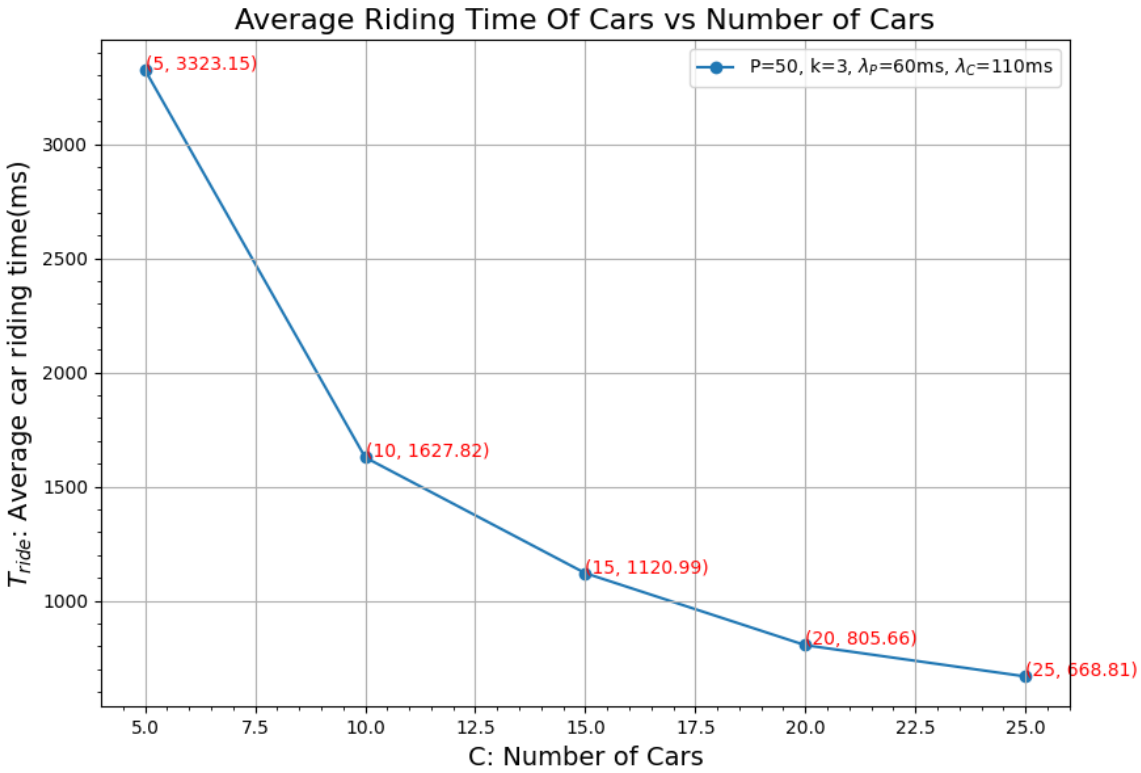
For P less than 25, the graph has more fluctuations. It can be due to the reduced sample size leading to more variance. It can also be due to more car threads having to sleep and reawaken, thus slowing down corresponding passengers.

Plot2. Graph of average ride time of cars vs number of cars (C).

C varies from 5 to 25. P = 50, k =3, lambdaP = 60 ms, lambdaC = 110 ms are constant.

Table of readings of average ride time of cars. (Averaged over 5 readings)

Number of cars (C)	Average Ride Time (ms)
5	3323.15
10	1627.82
15	1120.99
20	805.66
25	668.81



Analysis:

Average riding time of cars decreases as the number of cars increases. Increasing the number of cars reduces the number of trips each car makes, thus reducing the average riding time.

Average riding time of cars = $\sum_{i=1}^{C}$ (Riding time of car i) / C

= Total car riding time / C

= $\sum_{j=1}^{P}$ (Riding time of Passenger j) / C (due to one-one pairing of cars and passengers)

Expected riding time of passenger = k × lambdaC

⇒ Expected average riding time of cars = P × k × lambdaC / C

This is inversely proportional to C. Hence the graph is hyperbolic and decreasing with C.