Operating Systems CS3523

Programming Assignment 4

ES21BTECH11022

High-Level Design of Code

---

1. Taking Input: The program uses command line arguments to take the input file containing values of p, c, lambda_p, lambda_c, and k. Here p represents the number of passenger threads, c the number of car threads, and k represents the number of times each thread attempts to enter the critical section. Apart from p ,c and k, we also input lambda_p and lambda_c. We use these constants of exponential time distribution to recreate the Jurassic park problem by using lambda_p for the time between two passenger requests and lambda_c for the ride time of the cars. Here we also check for two errors an inappropriate number of command line arguments and the case of fopen returning a NULL pointer. In both cases, we print the error onto the output file.

2. CreatingThreads: For thread creation, we declare a vector of cpp threads. Then the program pushes back p + c threads using a for a loop. The pushed-back thread is initialized via the constructor calling two different functions. The first p threads act as passengers and execute func_passenger while the rest of the c threads perform the func_car function. We also pass an integer to each function as input: the thread number for the corresponding passenger or car thread.

3. Function func_passenger: Each passenger thread runs this function. First, we note the time the passenger starts, i.e., when the thread enters the museum. Next, we initialize the variables used to simulate the time between two requests of a passenger thread. Now we analyze how a request takes place. We start by noting the time of the request and print the same. Next, we access a boolean array called cars which indicates the availability of car threads. Now only c passengers can ride at any time, so we use a semaphore to ensure this. Once a passenger thread gets permission to access the cars array, it loops through to check for an available car. Once it finds an available car say car x, it changes cars[x] to true, indicating the car to start the ride. Now while the passenger is riding the car, it waits for the ride to get over using a binary semaphore. Once it receives the signal from the semaphore, the ride gets over and notes the time. As per the problem, the passenger waits for a time period exponentially distributed with constant lambda_p. This process entails one ride request from a passenger. Each passenger makes such request k times using a for loop. After all k requests, the passenger notes the time as the exit time.

4. Function func_car: Each car thread calls this function. First, we initialize the variables used to simulate the ride time of a car thread. Now the car enters into a while loop until the museum shuts down, i.e., all passenger threads join. Now a car thread j checks if the car array's corresponding value is true. Once true, the car starts the ride process by sleeping for the time connected to lambda_c. Once the ride ends, it signals the passenger thread using the corresponding binary semaphore to the car. This checking of cars[j] continues till the museum shuts down, after which the threads terminate and join.

5. Final Output File: We have only one main output file named "OutMain.txt" for this program. This contains the logs of the passenger and car threads. This contains the following timestamps:
i) The time at which each of the p passengers enters the museum,
ii) The time at which a passenger requests a ride. This includes the passenger number and the request number, which can go up to k.
iii) The time at which the request is accepted. This includes the passenger number, the car number which accepts the request, and the request number.
iv) The time at which the ride finishes. This includes the passenger number, the car number which accepted the request, and the request number.
v) The time when the passenger exits the museum.

Low-Level Design of Code

Considering the low-level design, we first create a vector of threads. And we then push p + c threads, each calling the thread constructor, which tells each thread to run the function func_passenger for the first p threads and function func_car for the remaining c threads with the argument as the corresponding thread number.

In each function func, we first declare the default random engine used to generate pseudo-random numbers named generator. Then we declare two exponential distribution variables named wait_p and wait_c, which are initialized using the constructor to their corresponding mean values. Then we can later use the operator () to generate the random numbers.

Next, we analyze the use of semaphores in the program. The first is the semaphore arr, which ensures that only c passenger threads can access the cars array. This is so because only c passengers can ride the car at any time. First, to initialize the semaphore, we use the sem_init function, which takes three arguments, a pointer to the semaphore, an int value indicating whether the semaphore is global or in shared memory, and finally, the count of the semaphore. Thus we use

```
sem_init(&arr, 0, c);
```

Whenever we need the wait and signal functions of semaphore, we use sem_wait and sem_post, both of which take a pointer to the semaphore as an argument.

The next semaphore we use is an array of semaphores corresponding to each car. Each semaphore is a binary semaphore to communicate with the passenger and the selected car. Thus we initialize it as follows,

```
for(int i = 0; i < c; i++)
    my_sem_init(resume + i, 0, 1);
```

When the ride starts, the passenger thread uses the corresponding sem_wait, and after the ride, the car uses sem_post.

Another point is that, in some cases, the code gets stuck in some sem_wait. So to remedy this we use sem_timedwait, which just uses an additional argument of timespec struct, which we declare in the main function.

Analysis of Output:

Output file using p = 5, c = 3, lambda_p = 5, lambd_c = 20, k = 3.

```
Passenger 3 enters the museum after time 0.224000 ms
Passenger 3 made reqeuest number 1 after time 0.275000 ms
Car 1 accepts passenger 3 request number 1 after time 0.277000 ms
Car 1 finished passenger 3 ride number 1 after time 0.277000 ms
Passenger 4 enters the museum after time 0.288000 ms
Passenger 4 made reqeuest number 1 after time 0.296000 ms
Car 1 accepts passenger 4 request number 1 after time 0.299000 ms
Car 1 finished passenger 4 ride number 1 after time 0.301000 ms
Passenger 5 enters the museum after time 0.354000 ms
Passenger 5 made reqeuest number 1 after time 0.355000 ms
Car 1 accepts passenger 5 request number 1 after time 0.355000 ms
Passenger 3 made reqeuest number 2 after time 0.366000 ms
Car 2 accepts passenger 3 request number 2 after time 0.371000 ms
Car 2 finished passenger 3 ride number 2 after time 0.373000 ms
Passenger 4 made reqeuest number 2 after time 0.383000 ms
Car 2 accepts passenger 4 request number 2 after time 0.390000 ms
Passenger 1 enters the museum after time 0.390000 ms
Passenger 1 made reqeuest number 1 after time 0.401000 ms
Car 2 accepts passenger 1 request number 1 after time 0.403000 ms
Passenger 2 enters the museum after time 0.391000 ms
Passenger 2 made reqeuest number 1 after time 0.415000 ms
Car 3 accepts passenger 2 request number 1 after time 0.418000 ms
Car 3 finished passenger 2 ride number 1 after time 0.420000 ms
Car 2 finished passenger 4 ride number 2 after time 0.394000 ms
Passenger 3 made reqeuest number 3 after time 0.443000 ms
Car 3 accepts passenger 3 request number 3 after time 0.445000 ms
Car 3 finished passenger 3 ride number 3 after time 0.446000 ms
Passenger 2 made reqeuest number 2 after time 0.486000 ms
Car 3 accepts passenger 2 request number 2 after time 0.488000 ms
Passenger 4 made reqeuest number 3 after time 0.495000 ms
Car 4 accepts passenger 4 request number 3 after time 0.499000 ms
Car 4 finished passenger 4 ride number 3 after time 0.501000 ms
Passenger 3 exits the museum after time 0.508000 ms
Car 1 finished passenger 5 ride number 1 after time 0.528000 ms
Car 2 finished passenger 1 ride number 1 after time 0.563000 ms
Passenger 4 exits the museum after time 0.570000 ms
Car 3 finished passenger 2 ride number 2 after time 0.570000 ms
Passenger 5 made reqeuest number 2 after time 0.584000 ms
Car 1 accepts passenger 5 request number 2 after time 0.585000 ms
Car 1 finished passenger 5 ride number 2 after time 0.587000 ms
Passenger 1 made reqeuest number 2 after time 0.630000 ms
Passenger 2 made reqeuest number 3 after time 0.631000 ms
Car 2 accepts passenger 2 request number 3 after time 0.636000 ms
Car 2 finished passenger 2 ride number 3 after time 0.638000 ms
Passenger 5 made reqeuest number 3 after time 0.642000 ms
Car 1 accepts passenger 1 request number 2 after time 0.632000 ms
Car 1 finished passenger 1 ride number 2 after time 0.646000 ms
Car 2 accepts passenger 5 request number 3 after time 0.643000 ms
Car 2 finished passenger 5 ride number 3 after time 0.695000 ms
Passenger 2 exits the museum after time 0.699000 ms
Passenger 1 made reqeuest number 3 after time 0.706000 ms
Car 1 accepts passenger 1 request number 3 after time 0.707000 ms
Car 1 finished passenger 1 ride number 3 after time 0.709000 ms
Passenger 5 exits the museum after time 0.749000 ms
Passenger 1 exits the museum after time 0.767000 ms
```

Statistics:

Average Tour Time vs Number of Passenger Threads:

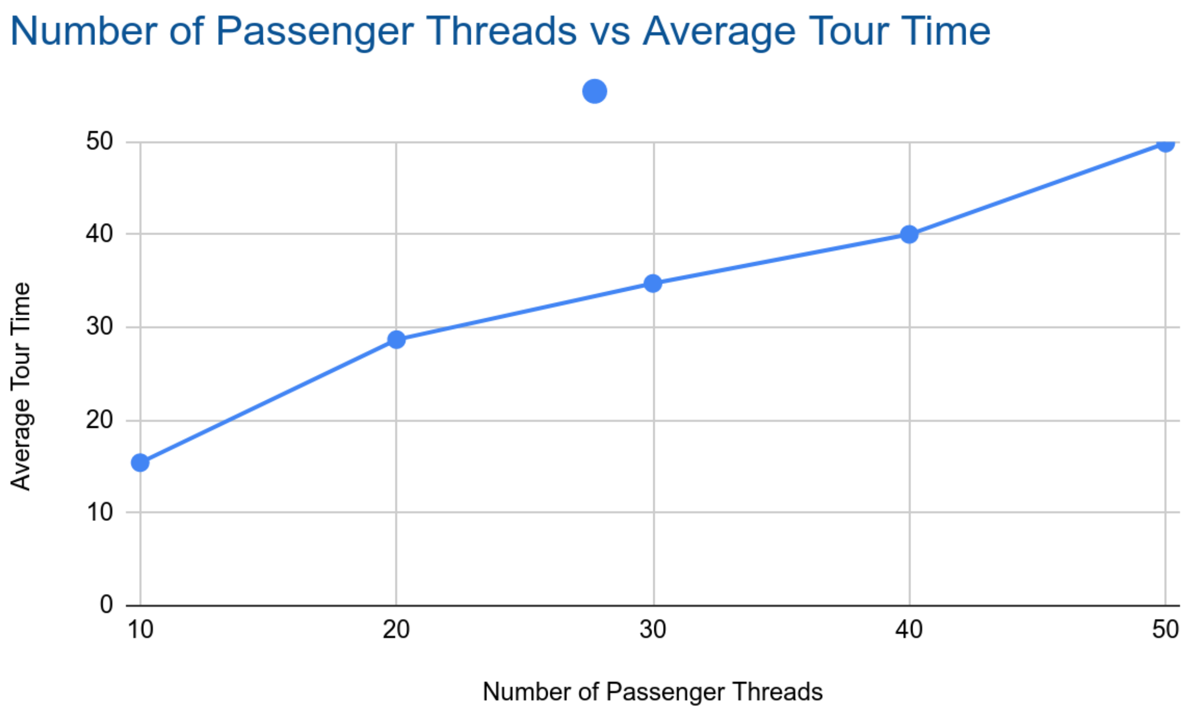
Passenger Thread Number	Average Tour Time Readings (milliseconds)					Average Tour Time
	Time 1	Time 2	Time 3	Time 4	Time 5	
10	15.059	16.724	22.914	9.5597	12.8252	15.41638
20	30.942	27.052	23.404	32.162	29.853	28.6826
30	34.877	38.226	37.822	27.48	35.319	34.7448
40	36.8967	45.122	45.481	36.453	36.206	40.03174
50	40.782	67.886	50.836	42.34	47.481	49.865

Average Ride Time vs Number of Car Threads:

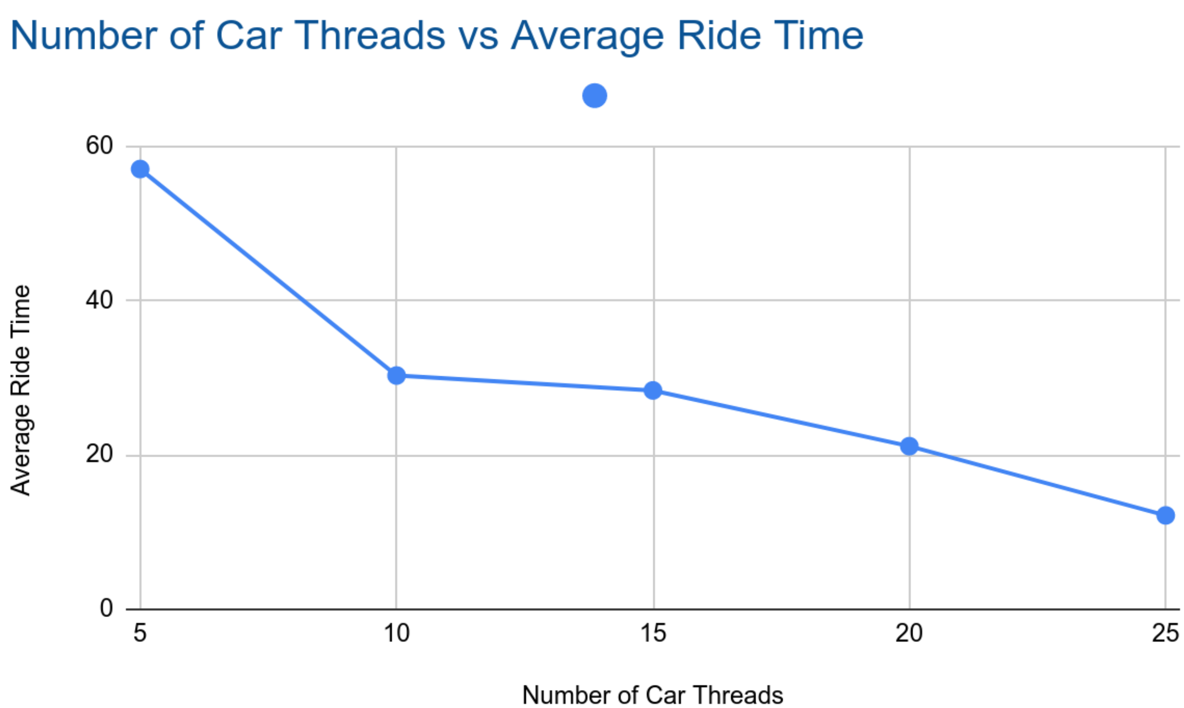
Car Thread Number	Average Ride Time Readings (milliseconds)					Average Ride Time
	Time 1	Time 2	Time 3	Time 4	Time 5	
5	66.263	55.234	68.367	50.821	44.516	57.0402
10	35.114	29.787	29.592	33.225	23.741	30.2918
15	11.271	44.763	18.699	28.398	38.677	28.3616
20	25.861	17.066	24.94	25.472	12.349	21.1376
25	13.162	12.64	11.859	17.102	6.08	12.1686

On average, increasing the number of passengers increases the average total tour time as the number of requests increases. Thus the wait time also increases for a constant number of cars.
Also, the ride time of cars decreases when the number of cars increases as more requests are satisfied parallelly.

Graphs:



Number of Passenger Threads vs Average Tour Time
Average Tour Time
50
40
30
20
10
0
10
20
30
40
50
Number of Passenger Threads



Number of Car Threads vs Average Ride Time
Average Ride Time
60
40
20
0
5
10
15
20
25
Number of Car Threads