CS3523 : OPERATING SYSTEMS 2
Programming Assignment 4
The Jurassic Park Problem

CS21BTECH11055
SADINENI ABHINAY

March 2023

Contents

1 Aim

- Use semaphores to synchronize the m-passenger processes and the n-car processes.

2 Input

- Input is given in form of a file(inp-params.txt) which contains the following parameters in order as mentioned in the problem statement:

  1. No of Passenger Threads P
  2. No of Car Threads C
  3. the parameter for the exponential wait between 2 successive ride requests made by the passenger λ_P
  4. the parameter for the exponential wait between 2 successive ride requests made by the passenger λ_C
  5. The number of ride request made by each passenger k.

3 Output

- The program generates the log of all rides and write them to output.txt

4 Detailed Explanation of low level design

The design contains three parts:

1. Passenger thread
2. Car thread
3. Main function

4.1 semaphores:

```
sem_t *car_fill_seat;
sem_t *ride_over;
sem_t *car_seat_taken;
sem_t passenger;
pthread_mutex_t lock;
```

- The above semaphores are used for synchronization
- car_filled_seat for loading passenger
- car_seat_taken for starting the ride
- passenger for signalling passenger thread to load passenger
- lock for counting no of rides

4.2 Passenger thread:

```
     void *passgen_work(void *args)
{
    param *p = (param *)args;
    int index = p->id;
    for (int i = 0; i < iter; i++)
{
        sem_wait(&passenger);

        int avail_car;
        for (int i = 0; i < N_cars; i++)
        {
            if (avail_cars[i] == 0)
            {
                avail_car = i;
                avail_cars[i] = index + 1;
                sem_post(&car_seat_taken[i]);
                break;
            }
        }
        sem_wait(&ride_over[avail_car]);
        sem_post(&car_fill_seat[avail_car]);
        usleep(p->T * 1000);
    }
}
```

- First when sem_wait(passenger) makes the passenger to wait for the cars
- Next the in the loop we will find the first free car which has a vacant seat and will lock the seat of the car
- We will wait for the car to stop the ride(sem_wait(&ride_over[avail_car]))
- Then we will vacant the seat of the car (sem_post(&car_fill_seat[avail_car]))
- then passenger thread is put to sleep for time gap purpose

4.3 Car thread:

```
     void *car_work(void *args)
{
    param *p = (param *)args;
    int index = p->id;
    int tr;

    pthread_mutex_lock(&lock);
    tr = rides;
    pthread_mutex_unlock(&lock);

    timeval curTime;

    while (true)
    {
        sem_wait(&car_fill_seat[index]);
        sem_post(&passenger);
        sem_wait(&car_seat_taken[index]);
        int pass = avail_cars[index];
        if (pass == 0)
        {
            continue;
        }
        avail_cars[index] = 0;
        pthread_mutex_lock(&lock);
        rides++;
        tr = rides;
        pthread_mutex_unlock(&lock);
        sem_post(&ride_over[index]);
        usleep(p->T * 1000);
    }
}
```

- First we will wait until the seat is vacant(sem_wait (car_fill_seat [index]))
- Then we will signal the passenger to get passengers loading(sem_post (& passenger))
- Wait until the seat is taken (some passenger enters the car)
- then we will find which passenger entered the car(avail_cars [index])
- if the seat is still vacant we will try again
- we will update the total no of rides
- put the car thread to sleep for time gap purpose

4.4 Main function

```
    int main(int argc, char const *argv[])
{

    FILE *inp = fopen("inp-params.txt", "r");
    int lambda_P, lambda_C;
    fscanf(inp, "%d%d%d%d%d", &N_pass, &N_cars, &lambda_P, &lambda_C, &iter);
    fclose(inp);
```

- taking input from a file with parameters mentioned above in section 2

```
    default_random_engine gen(seed);
    exponential_distribution<double> rng_1(lambda_P);
    exponential_distribution<double> rng_2(lambda_C);

    pthread_t *passengers = (pthread_t *)malloc(sizeof(pthread_t) * N_pass);
    pthread_t *cars = (pthread_t *)malloc(sizeof(pthread_t) * N_cars);
    car_seat_taken = (sem_t *)malloc(sizeof(sem_t) * N_cars);
    car_fill_seat = (sem_t *)malloc(sizeof(sem_t) * N_pass);
    ride_over = (sem_t *)malloc(sizeof(sem_t) * N_pass);
    avail_cars = (int *)calloc(sizeof(int), N_cars);
    sem_init(&passenger, 0, 0);
```

- Initialize exponential distribution and car thread ids and passenger thread ids
- And also intializing semaphores

```
       for (int i = 0; i < N_cars; i++)
    {

        param *p = (param *)malloc(sizeof(param));
        p->fp = out;
        p->T = rng_2(gen);
        p->id = i;
        sem_init(&car_fill_seat[i], 0, 1);
        sem_init(&car_seat_taken[i], 0, 0);

        pthread_create(&cars[i], NULL, car_work, (void *)p);
    }
```

- create car thread and passing parameters
- in parameters sending id and generated wait time from the exponential distribution

```
    for (int i = 0; i < N_pass; i++)
    {
        param *p = (param *)malloc(sizeof(param));
        p->fp = out;
        p->T = rng_1(gen);
        p->id = i;
        sem_init(&ride_over[i], 0, 0);
        pthread_create(&passengers[i], NULL, passgen_work, (void *)p);
    }
```

- create passenger thread and passing parameters
- in parameters sending id and generated wait time from the exponential distribution

```
    for (int i = 0; i < N_pass; i++)
    {
        pthread_join(passengers[i], NULL);
    }

    return 0;
}
```

- Main function wait for all passenger threads to terminate
- Output log generation is done inside the threads all code lines of log generation mentioned below
- Here i truncated them because of the pdf width
- This lines are not included in the code listings(Car thread,passenger thread) because of the pdf width

```
In car thread:
        fprintf(p->fp, "car %d accepted passenger's No:%d request\n",...);
        fprintf(p->fp, "car %d completed passenger's No:%d request\n",...);
In passenger thread:
fprintf(p->fp, "Passengers Thread No:% d in the museum at %s:%d \n",..)
fprintf(p->fp, "Passenger %d made a ride request at %s:%d \n",...)
fprintf(p->fp, "Passenger %d started riding at %s:%d \n",...);
fprintf(p->fp, "Passengers Thread No:% d has finished his tour .\n",..);
```

5 Analysis of Output

5.1 No of cars(C) constant

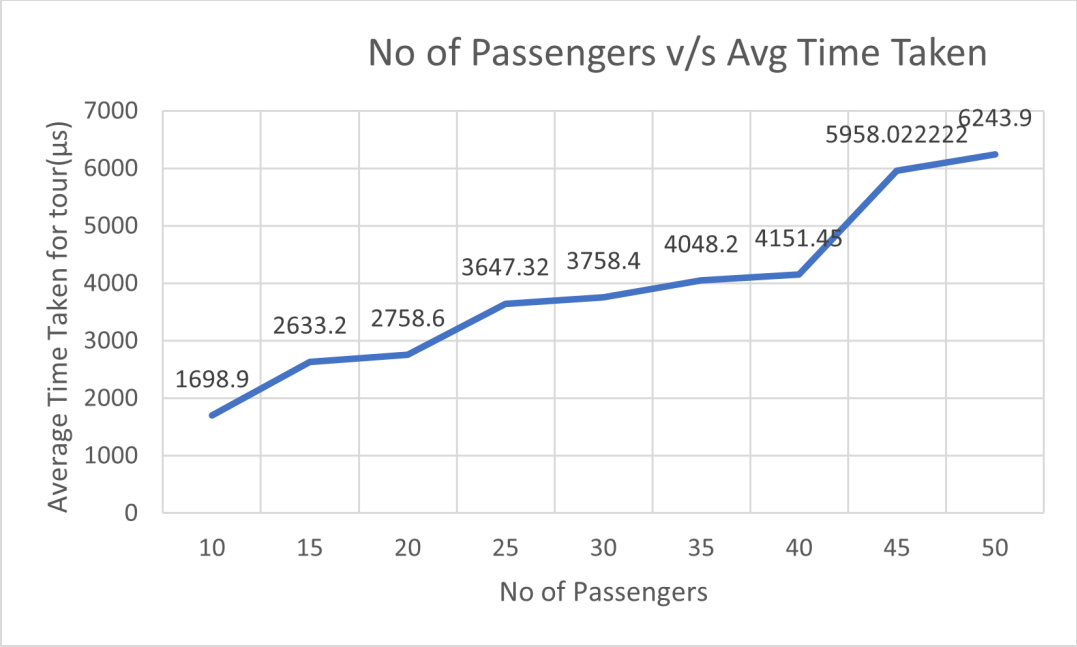

Figure 1: T_{avg} vs P (C=25,k=5)

5.1.1 Observations-1:

- Here the trend is as expected
- When No of passengers are increased ,there are two factors that increases the average time
- First, the pthread creation overhead
- second,the waiting time of the passenger here ,large no of passengers wait for car
- we didn't make sure to put an bounded waiting time condition
- As a result the passengers stand more time because only few cars are available for hire
- this reflects on the average time of the tour

5.1.2 Conclusion-1:

- As No of passengers increases the average tour also increases

5.2 No of passengers(P) constant

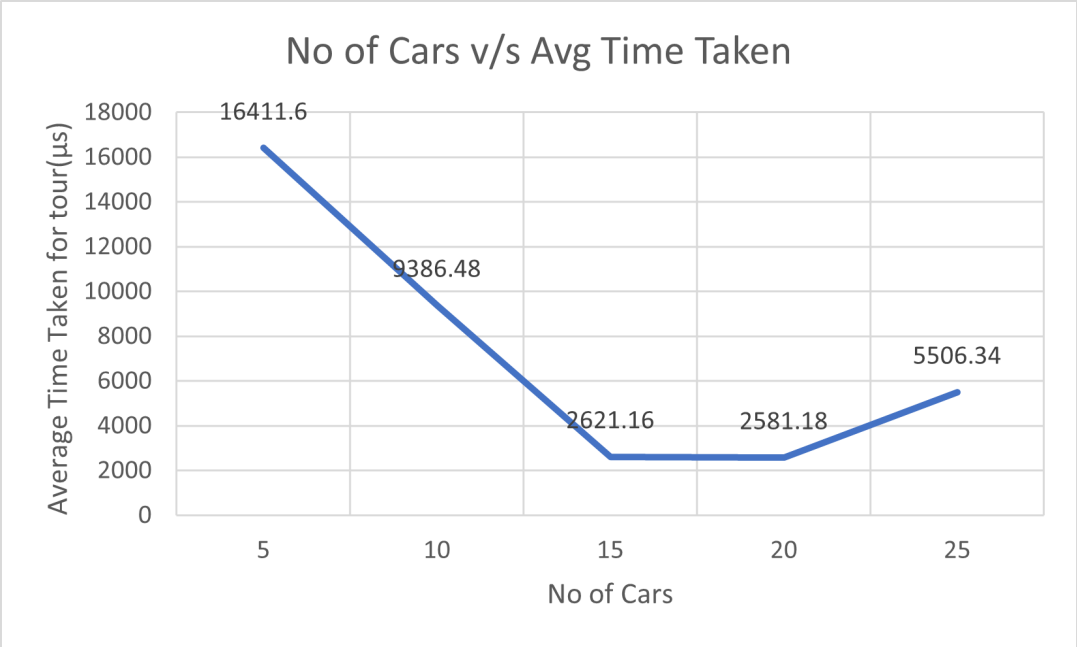

Figure 2: T_{avg} vs C (P=50,k=3)

5.2.1 Observations-2:

- Here the trend is as expected until C=20
- When No of cars are increased ,there are two factors that affect the average time
- First, the pthread creation overhead
- As seen in above graph pthread overhead is only significant in case where C=50
- second,the waiting time of the passenger here
- there more cars available now ,so the waiting time of the passenger decreases
- this reflects on the average time of the tour

5.2.2 Conclusion-2:

- As the no of cars increases the average tour time decreases until certain point here in this case it is 20