REPORT

SRI HARI. M
CS19BTECH 11039

Assignment - 4

Aim: Demonstrate the system modelling and analysing performance with average waiting times for variation in threads and variation in size of the table.

System modelling:

This modelling involves the main thread, creates a number of threads and assigning a task to each thread with the given constraints. The constraints are that a set of new customers arrive every time with an exponential delay and the size of the set is equal to S, which is uniformly distributed from 1 to r*X. The thread management is only to be done for the entering of threads into the critical section and is exiting the critical section does not require any monitoring.

Let's see the code segment and see how that is done.

Three variables are taken to complete the system modelling and are initialized as follows for avoiding random behaviour. The temp variable is initialized to the number of

customers and the rest with 0. The while loop next to this requires the temp to get to 0 and set and i reaches the number of customers.

```
//variables for proper ensurance of handling threads
int temp = number_of_customers;
int set = 0, i = 0;
```

The idea is simple, that get a set by using a uniform distribution generator and create threads. With an exponential delay, another set of customers is taken into account and create threads for them until all the customers enter into the critical section.

```
//a while loop in which thread management is done
while(temp)
{
    //gets a value
    set += uniform(r*size_of_table);

  //a for loop for looping up the order and creating new threads
    for( ; i < set and i < number_of_customers ; i++)
    worker_threads[i] = thread(enter_restaurant,i);

    //reducing the number of people yet to join the table
    temp = number_of_customers-i;

   //getting an exponential time for the main thread to sleep on
    int time = exponential(lambda);
    this_thread::sleep_for(chrono::milliseconds(time));
}
```

As one can see that initially the variable temp is initialized with the number of customers and while loop depends on the

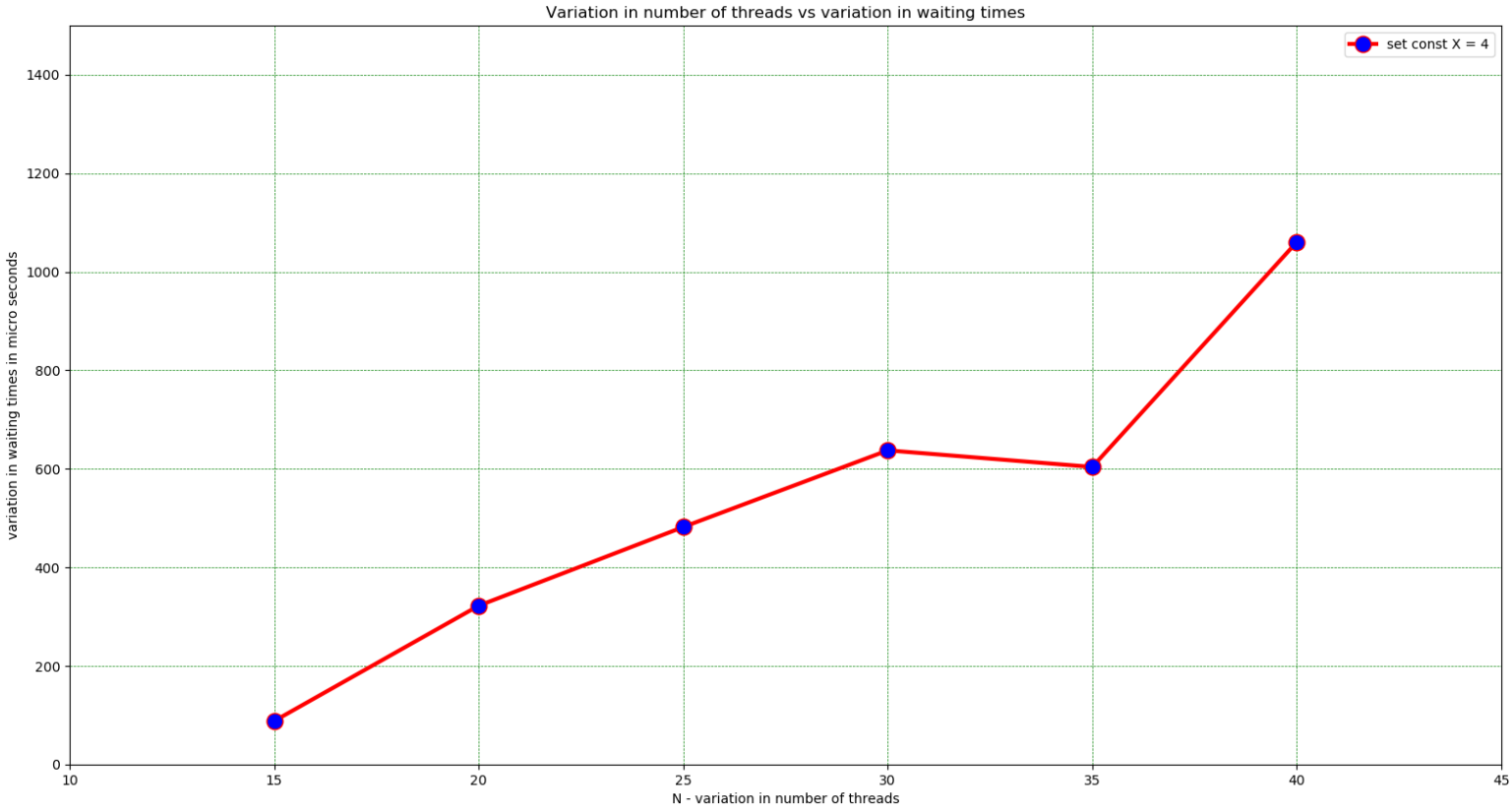
value of temp, which updates at every iteration. First, the variable set gets a number that is uniformly distributed from 1 to r*X(size of the table). Then a for loop comes into the picture which runs with the variable 'i'. As the variable is initialized with 0, the creation starts from thread 0 and goes up to the value of the set. Here, the variable set is getting continuously updated at every iteration, and there is a problem that we might run out of threads created if the final value of the set is greater than the number of customers. So, the conditional statement in for loop is set to `i < set and i < number_of_customers`, which ensures that every time, 'i' reaches and hits the value set or else if all the customers are entered into the critical section, then it stops. And next to that there is a statement updating the value of temp. Each time the value is updated such that in final after all the customers enter the critical section, it becomes 0 and thus halts the while loop.

After a particular set of customers enters the critical section, it was mentioned that the next set of customers comes with an exponential delay. For that, the time variable gets a variable that is exponentially distributed over the average value of lambda and then the thread sleeps for that many milliseconds.

This is how the system modelling has been done.

Graphs:

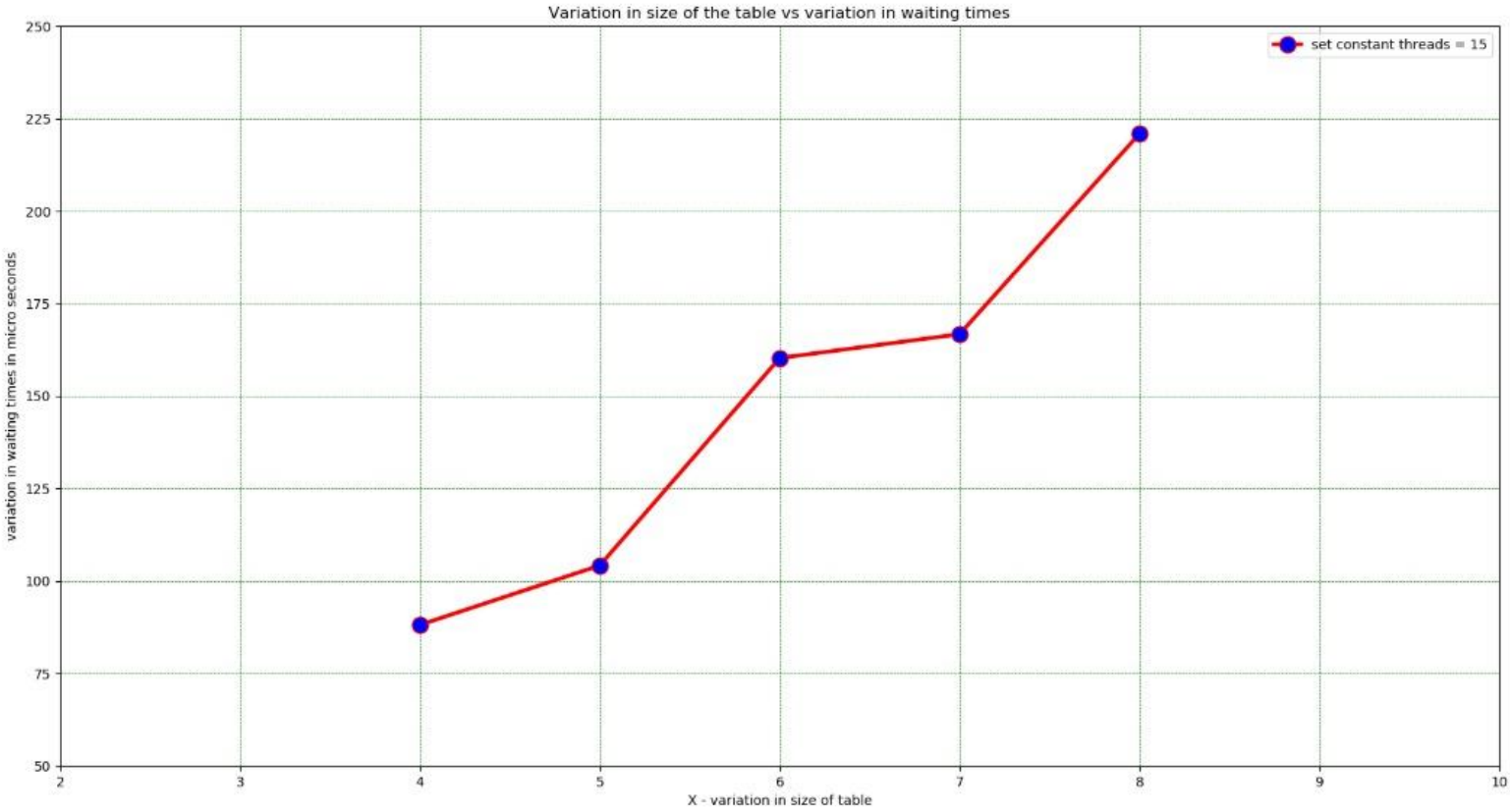
a. Variation in threads vs waiting times(size of the table is constant)



This is the graph that is plotted with variation in the number of threads on the X-axis and the respective waiting times on the Y-axis. Here we can see that the graph is rising as the number of threads increases with the size of the table kept constant at 4. The graph goes on increase as the number of threads increase but there is a slight depression in the curve at N = 35, which is countered by the rise of the graph at N = 40. So that can be considered as some edge cases running over. It is obvious that the average waiting time increases with the

increasing number of customers or threads as the table size is fixed, more the number of customers, there is more the number of threads, which gets into the condition that the table is occupied and more number of customers been waiting to join the table. So, keeping the size of the table constant, with the increase in the number of customers, there is increased waiting time.

b. Variation in size of table vs waiting times (number of threads is constant)



This is the graph plotted with variation in size of the table on the X-axis and their variation of waiting times on the Y-axis.

The graph is rising all the way as the size of the table increases. But from our observation, it is to be decreasing as the size of the table increases as the number of people who can form a group increases, thus letting more people form a group at once and this ensures that as the size of the table increases, there will be less waiting at a time and hence there should be less average waiting time but in contrast, we are getting an increase all along with the graph. This is because of the overhead of the calculations over these small numbers. For a system with a very large number of threads and a large size table, with the increase in the number of threads, the waiting time remains almost constant as it considers only the overhead of calculating the time requested to accessed. But for these values, the graphs get increasing. Also, take a case where 16 customers are actually in queue and 15 as the size of the table. That one last customer can significantly affect the waiting times.

THE END!