****Report for 'KOREAN RESTAURANT PROBLEM' :**

Design of the algorithm:

1.After creating the cpp file, starting writing the code by including necessary header files such as "iostream", "fstream" etc.,

Here, we should include "semaphore.h" header file to use semaphores.

2.Then declare necessary functions like expo_dist (for exponential distribution), uniform (for uniform distribution), threadfunc (function for threads)

3.Also declare global variables:

i.eating – To know number of customers eating at the table

ii.waiting – To know number of customers waiting for a seat

iii.mutex, block are semaphores for multithreading purpose.

Print semaphore for printing output into a file.

iv.must_wait – To know whether any group has been formed or not

4. Now, in the main function, using a file pointer Infile declared with ifstream, open the file "input.txt" and read the input for N,X,lambda,r,tou.

N- total number of customers

X- total number of seats at the table

Lamda- parameter for exponential delay between set of customers

r- value required to calculate sets

tou- average time of exponential distribution for any customer to eat

Also declare the file pointer 'Outfile' with ofstream.

5. The mutex,print semaphores are initialised with '1' and block semaphore with '0'.

5.Now, with the help of a while loop, create threads for each set of customers which is uniformly selected from 1 to r*X . And maintain a time delay generated by exponential distribution with lambda as parameter in milliseconds.

Then after, join all the threads.

6.In threadfunc, , calculate the 'system time' 2 times i.e, at the time of request access, at the time of entry into critical section i.e, at the time of given access and print the time.

(time of exit is also mentioned in comments for checking any group conditions)

7. Then in threadfunc, i.e, a new customer arrives at the restaurant and request access and it prints the time and then the customer checks if any group has already been formed(must_wait=t=true) or is there any empty seat(eating+1>X).

If then, the customer waits until all the members of the group complete their eating and then gets access.

Else, the customer directly get access.

8.Here, the waiting time is calculated and time is printed.

9.with the help of semaphores, the values of waiting, eating, must_wait are updated within them.

10.Later the customer starts eating and eating time is simulated with help of sleep function using tou.

11.After that customer finishes eating and exit the restaurant, And each time, eating will be checked if it is zero or not.

If eating is 0, (means previously a group is formed and others are waiting or customers entered, completed their eating), then the customers who are waiting are given access randomly and eating, must_wait values are updated accordingly.

12.This process terminates if all the N customers finish their eating.

System Modelling:

The above designed algorithm is a centralised approach.

In this, the master thread(main thread) will coordinate the creation of threads for the customers and ensures that the arriving customer threads are successfully created.

It even make sure that any new customer threads as a set of customers at correct time intervals with help of sleep function for time delay between set of customers.

Finally, it also ensures the termination of the algorithm when the while loop ends.

Graphs:

The following below are two graphs which are generated basing on the following fixed values:

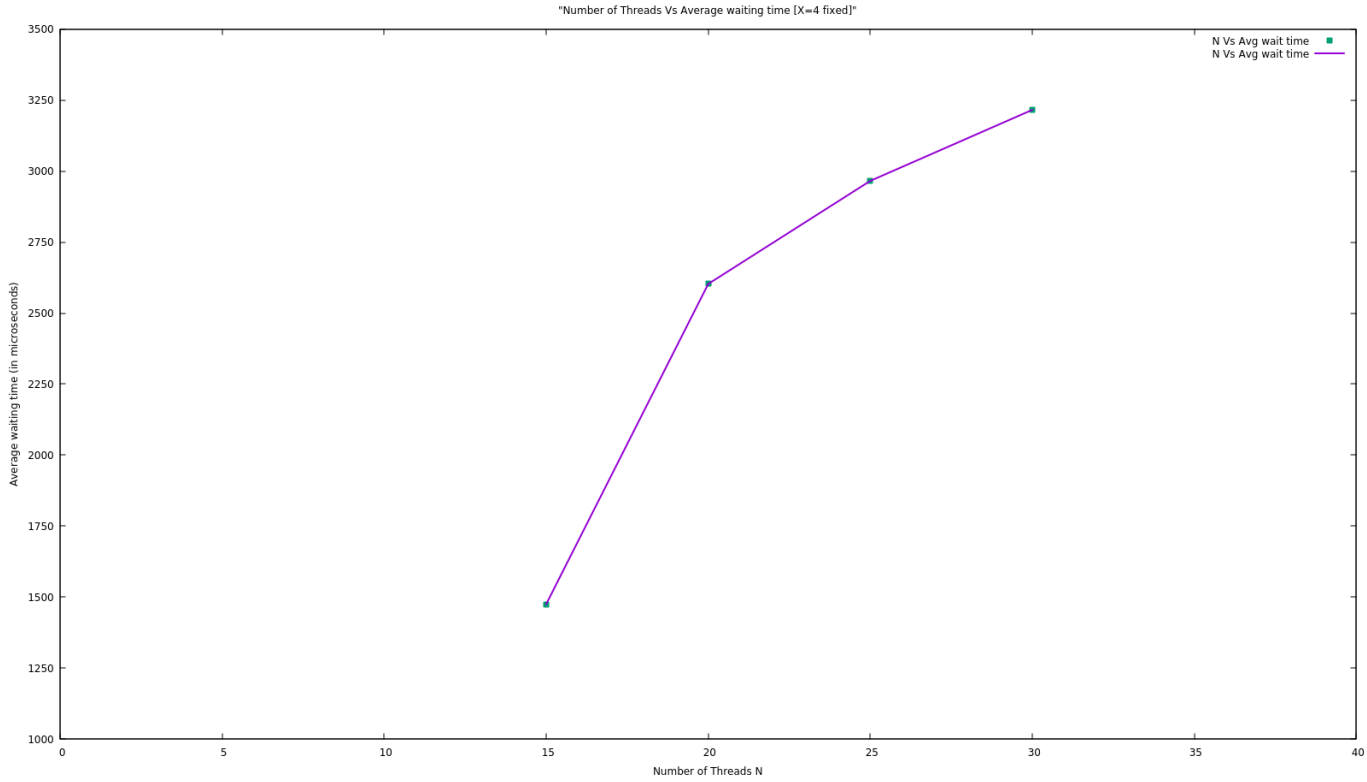
Lambda=0.1, r=2, tou=10 (N and X are based on the corresponding graphs)

Graph1:showing the “Number of Threads Vs Average waiting time of customer (in microsec)”.

Graph2: showing the “Value of X Vs Average Waiting time of customer (in microsec)”.

Basing on the results displayed in the graphs, a small analysis is made on the output.

Graph1: “Number of threads N Vs Average waiting time of each customer (in microseconds)”



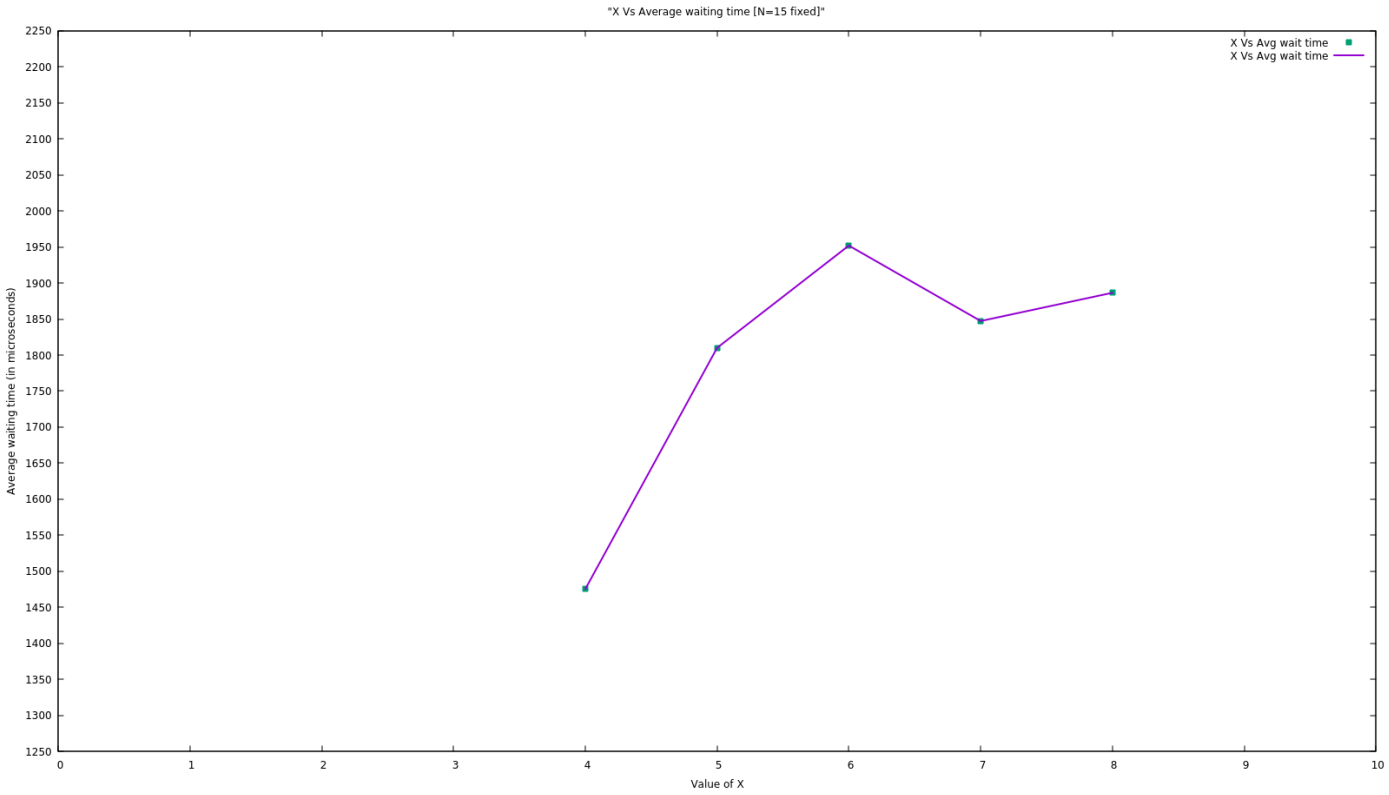
For generating this graph, value of X is fixed to ‘4’ and other values are taken as mentioned above, whereas N varies from 15 to 30.

Analysis:

Clearly, from the graph, as number of threads increases keeping X fixed, the waiting time of each customer increases.

Logically also, as there are fixed number of seats at the table, but as the number of customers N increase, each customer will have to wait more time, as N is more, it takes more time to finish their eating.

Graph2: “Value of X Vs Average waiting time of each customer (in microseconds)”



For generating this graph, number of threads N is fixed to 15 and other values are taken as mentioned above, whereas X varies from 4 to 8

Average of average waiting time for a customer = (1457.5+1810+1951.8+1847+1886.3)/5

= 1790.52 microseconds

Analysis:

Logically, if number of customers N is fixed but the number of seats at the table i.e, X increases, then there will be decrement in waiting time for customers as there are more seats available. But at the same time, if any group is formed, then the customers will have to wait for entire group to complete their eating, as X increase, a big group will be formed as new customers have to wait more time as well.

Hence, in a way, it decreases wait time and also increase based on group formation, which is uncertain.

Now, from the graph,

For X=4 to 6: the waiting time of each customer is increasing

For X=6 to 8: the waiting time of each customer is almost same.