Korean Restaurant Problem

Problem Statement - Imagine a hypothetical restaurant with a single table, consisting of X seats. If a customer arrives while there is a vacant seat at the table, the customer can take a seat immediately. Whereas, if a customer arrives when all X seats are occupied, this signifies that all X people currently dining form a group. The new customer will have to wait for the entire party to leave before he can sit down.
Implementing a solution for this problem in c++ using semaphores

Submitted By: Kushagra Indurkhya
CS19B1017

System model

For simulating the arrival of customers I have used a centralized approach wherein the main thread creates threads for a group of customers (size s: uniformly distributed between from 1 to X.R) with a delay of some time (exponentially distributed with lambda as average)

```
  default_random_engine generator;
  //Uniform distribution for number of customers
  uniform_int_distribution<int> customer_distribution(1,r*x);
  //Exponential distribution for delay in customer entry
  exponential_distribution<double> delay_expo(1/lambda_param);
  //Exponential distribution for eating time
  exponential_distribution<double> eat_time_expo(1/gamma_param);

for(int i=0;i<n;)
  {
      int s=customer_distribution(generator);
      double delay = delay_expo(generator);
      this_thread::sleep_for(chrono::duration<double,milli>(delay));
      for (int j=0; j < s; j++)
      {

          a[i].local_id = i;
          a[i].eat_time = eat_time_expo(generator);
          threads.push_back(thread(korean,&a[i]));
          i++;
          if(!(i<n)) break; //if n threads have been creaated break
      }
  }
```

i is incremented after each thread is created and when n threads are created the loop breaks hence n threads are created which when joined main waits for n threads to exit (n customers to finish eating)

Implementation detail

Algorithm

I have used algorithmic pseudocode provided to solve the given problem which uses
2 semaphores: mutex and block initialized to 1,0 respectively.

Global variables: eating, waiting keeps a record of the number of customers eating and the number of customers waiting and bool must_wait is true if the group has formed tells if the incoming customer has to wait.

When the thread starts execution request time is logged then the thread waits for mutex when it acquires mutex we check if the must_wait is true or must_wait should be updated (the customer releases mutex and waits on the block) else the customer can go inside (the group is not formed yet) after coming out of if-else construct the customer eats after eating if eating becomes 0 we update global variables accordingly and signal block k times which is the min of the number of customers waiting and X as at most X customers can enter so block becomes available to k waiting costumers which can then eat.

```
void korean(t_arg* data)
{
  logger* r_log=(new logger(data->local_id,"requested access "));
  data->buffer.push_back(*r_log);
  sem_wait(&mutex); //acquire mutex
  if(must_wait || eating+1>x)
  {
    waiting++;
    must_wait = true;
    sem_post(&mutex);
    sem_wait(&block);
  }
  else
  {
    eating++;
    must_wait = (waiting>0 && eating == x);
```

```
    sem_post(&mutex);
  }
  logger* eat_log=new logger(data->local_id,"started eating ");
  data->buffer.push_back(*eat_log);
  //Simulate Eating
  this_thread::sleep_for(chrono::duration<double,milli>(data->eat_time));

  sem_wait(&mutex);
  eating--;
  //The group has left
  if(eating == 0)
  {
    int k= min(x,waiting);
    waiting -= k;
    eating += k;
    must_wait = (waiting>0 && eating == x);
    for(int i=0;i<k;i++)
      sem_post(&block);
  }
  logger* exit_log=new logger(data->local_id,"exited ");
  data->buffer.push_back(*exit_log);
  sem_post(&mutex);

}
```

Logging

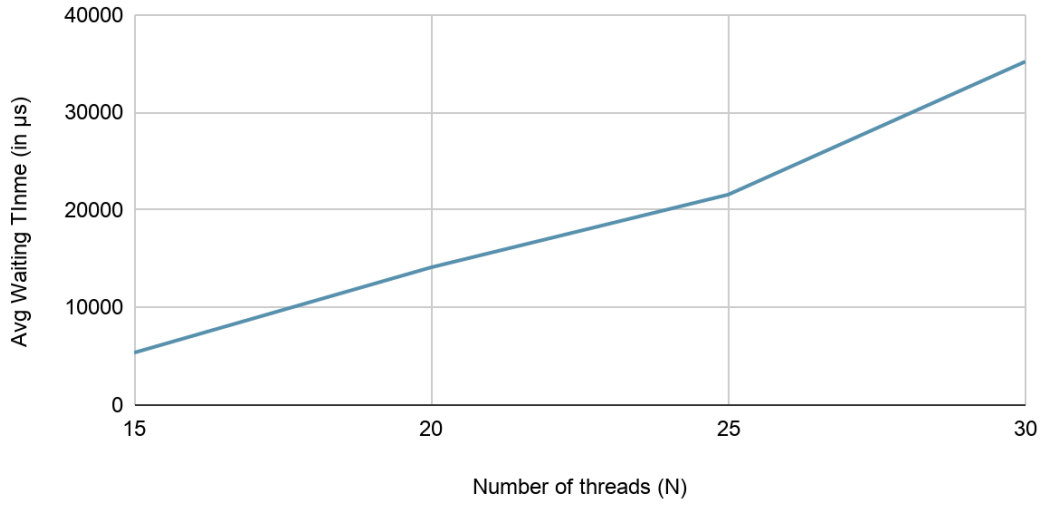
For logging, I have made a separate class logger in file “logger.cpp” in which I have created a logger class for logging each thread logs its request, entry, and exit times in a local buffer then in the main thread all the buffers are merged in one log and then sorted by time.

Graphs

I have plotted two graphs of avg waiting time (in microseconds) against varying n and x

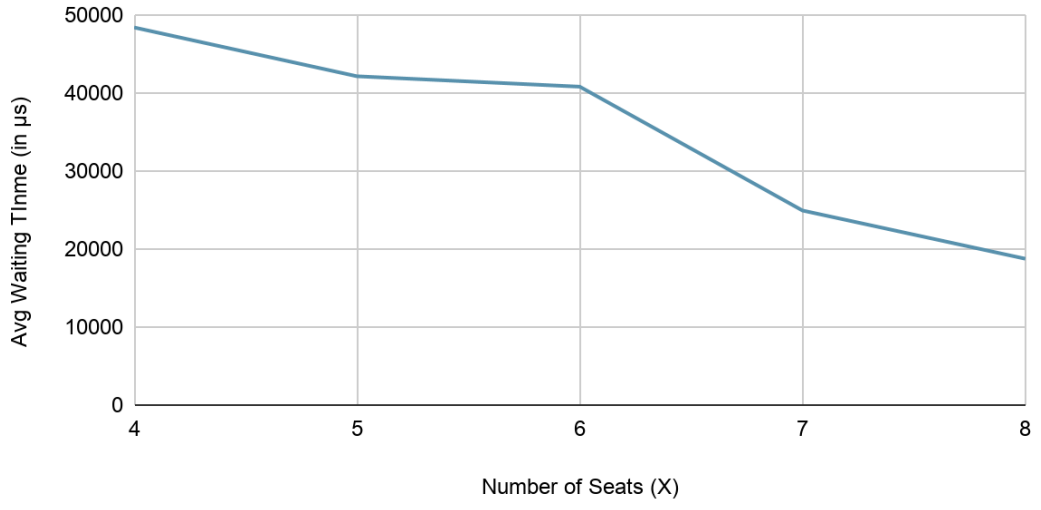
Graph-1

X=4, λ=10 , r= 0.7 , Γ=20



Graph-2

n=15, λ=10 , r= 1.5, Γ=20



Observations

Although the avg waiting times are dependent on the customer arrival and eating time which are both randomly generated from distributions over the provided inputs, and the scheduling of threads, a general trend can be observed:

- In graph-1 as we increase the number of threads keeping seats constant the avg waiting time **increases** as in general customers will have to wait more to get a seat.
- In graph-2 as we increase the number of seats keeping the number of customers constant, as more seats become available the avg waiting time **decreases**.