CSE 6010: Computational problem Solving for Scientists and Engineers

Professor Richard Fujimoto

Assignment 3: Discrete Event Simulation

10/19/2018

Team

Sunil S. Jaishankar (sjaishankar3)

Leyi Cai (lcai63)

Workload Distribution

Component	Person
Simulation Library *[application.c, components.h]* *[sim.h] – from sample code* *[engine.c] – modified*	Leyi Cai
Configuraton program *[configuration_program.c, configuration_functions.c, configuration_functions.h]*	Sunil Jaishankar
Interface/API [*functions in interface.h*]	Joint
Report	Joint

Contents

Introduction

In this assignment, we implemented discrete event simulation(DES) to perform simulation for queueing network.

Queueing networks are widely used to model systems where customers must utilize some service, and wait if the service is being utilized by other customer(s). Various methods are adopted to simulate the queueing network performance. Usually, we run the simulation driven by time, which may cause troubles if a server gets very busy at certain time that huge amount of customers have to be processed simultaneously; and when the server is idle, the simulation is still running and hence cause waste of memory. To improve the performance of simulation, DES is introduced.

Discrete Event Simulation (DES)

Discrete event simulations advance time from one event to the next, where each event represents something "interesting" occurring in the actual system. For example, in our software, three kinds of events are interested: *GENERATE*, *ARRIVAL*, and *DEPARTURE*. *GENERATE* event is that a customer is created, in other words, entering the system. *ARRIVAL* event is that a customer arrives in a component. The next step of the customer may varies based on the component type. *DEPARTURE* event is that the customer is about to leave current component, in this case, it only happens in Queue Station.

In DES, each event contains a timestamp, a simulation time value indicating when that event occurs in the physical system. The timestamp is analogous to the time step number in a time-stepped simulation, however, events occur at irregular points in time, not at regular, periodic time steps. Simulation time advances at irregular intervals as the computation proceeds from one event to the next.

Configuration Program

The configuration program has the following purposes

1. Accept arguments: simulation runtime, input filename, and output filename
2. Read in data from the input file
3. Perform check on the data read and display error messages if required
4. Create components (the network) for the application program
5. Run simulation
6. Extract statistics about the simulation
7. Print statistics to output file

The configuration program as defined here contains three parts

1. *configuration_program.c*

   This file has routines to read in data from input file line by line. For each line, *portions* of the line separated by white space characters are read in another loop and checked for validity using functions defined in the following 2 parts. Once the first portion of a line is read, it is converted to an integer if valid and is stored as the ID of the component that is to be created. Then the second portion, the component type is read and stored. Based on the component type

further words/portions in the lines are read and stored as time parameters, destination IDs, probabilities and number of fork components. Once the whole line is read, the data read is used to create a local (in the configuration program) instance of the component. This component is then added to a component array in the configuration program. The components array is then arranged in order of its IDs. This is because the input file may not have IDs in order. Once all lines are read and all data is valid the program calls functions in the interface/API to create components in the application. The data for this is supplied from the local components array previously mentioned. [1]

Once components are created in the application, the configuration program then calls a function to run the simulation (defined in the interface/API). After this, a function to get statistics is called. This data is stored and then printed to the output file using another function.

2. *configuration_functions.c* [2]

- Functions to check data and return valid data

***int toInt**(**char** * value_read)*

Function to read character string, check validity and return integer

***double toFloat_time**(**char** * value_read)*

Function to read character string, perform validity checks and return valid time value as double

***double toFloat_prob**(**char** * value_read)*

Function to read char string, perform validity checks and return valid probability values as double

***int charCheck**(**char** *value_read)*

Function to read in string, perform validity check on the character type value and return valid character

- Functions to create local instances of components

loc_gen **MakeLocalGen** (**int** GenID, **double** IntTime, **int** DestID)*

Function to create local instances of generator component

loc_QS **MakeLocalQS** (**int** QSid, **double** ServTime, **int** DestID)*

Function to create local instances of QS component

loc_exit **MakeLocalExit** (**int** ExitID)*

Function to create local instances of Exit component

loc_Fork **MakeLocalfork** (**int** Fid, **int** numComponents, **int*** destID, **double*** probabilities)*

Function to create local instances of forks

- Functions to check that the components created and associated attributes are valid

***int Generator_check**(loc_gen Gen, **struct** config_master_list* comp_list, **int** numofComponents)*

Function to check validity of previously created local generator component

***int QS_check**(loc_QS QS, **struct** config_master_list* comp_list, **int** numofComponents)*

Function to check validity of previously created local QS component

int fork_check*(loc_Fork fork,* ***struct*** *config_master_list* comp_list,* ***int*** *numofComponents)*
Function to check validity of previously created local fork component

- Function to release memory allocated for components generated in configuration program
  int freeComp*(****struct*** *config_master_list configComponents[],* ***int*** *numofComponents)*

- Function to print statistics about simulation
  int print_stat*(****char*** **outfilename, STAT *p,* ***int*** *numQS)*

- Structure that holds all data of components
  struct *config_master_list*

- Structures to create local instances of components:
  struct *local_generator*
  struct *local_QS*
  struct *local_Fork*
  struct *local_Exit*

3. *configuration_functions.h*

   Contains prototypes for structures used to create local instances of

Simulation Library

List of Structures

1. A **global array** to store components in the network, which contains the ID and infomation of each component.

```
struct {
  int ComponentType;
  void* Comp;     //infomation of component
} Component[MAXCOMPONENTS];
```

2. A **linked list** structure used in customer, which tracks the Queue Stations that the customer has entered.

```
typedef struct LinkedList List;
```

3. **Customer Structure**, which contains the time when the customer is created and exits the system. The wait time that customer in each queue and all queue is also included. Moreover, it contains the linked list mentioned above that records the path of the customer.

```
typedef struct customer Customer;
```

4. A single **event data structure** is used to handle all three event types. It contains the type of each event and the information of customer of the event as well as the ID of components that the event is going to occur.

```
typedef struct EventData EventData;
```

5. Structures for different components:

   – **Structure of Generator component**, which contains the interarrival time and ID of detination component of the Generator.

   ```
   typedef struct Generator Gen;
   ```

   – **Structure of Exit component**, which contains the number of customers that have exited.

   ```
   typedef struct Exit Exit;
   ```

   – **Structure of Queue Station component**, which contains the service time andd detination component of the Queue Station, as well as a FIFO queue to keep customers waiting.

   ```
   typedef struct Queue_Station QS;
   ```

   – **Structure of Fork component**, consists of the number of output port and two arrays; one array contains the ID of destination components and the other contains the probabilities of which port the customer is routed.

   ```
   typedef struct Fork Fork;
   ```

List of Functions

1. **Functions to create components**

   ```
   int MakeExit(int ExitID);
   int MakeGenerator (int GenID, double IntTime, int DestID);
   int MakeQStation (int QSID, double service_time, int DestID);
   int MakeFork(int ForkID, int K, double probabilities[], int DestID[]);
   ```

   If the component is created successfully, the function returns 1; otherwise returns 0.

2. **Functions to handle statistics**

   ```
   STAT* GetStat();
   void OutStat(STAT* p);
   ```

   – The `GetStat` function is for computation of output statistics, it returns a pointer of a structure holds the data. The structure is defined in the shared head file, whicn will be introduces in the next part.

   – The `OutStat` function prints out the data. This is not necessary is you just want to write the statistics into output file.

3. **Functions to handle events**

   ```
   void EventHandler (void *data);
   void Generate (EventData *e);
   void Arrival (EventData *e);
   void Departure (EventData *e);
   ```

A general event handler and three event handler deals with different kinds of events. Note that departure event only happens when a customer is leaving the queue station and the current first of the FIFO queue is arrange to depart from the queue station. The `EventHandler` is from sample simulation code.

4. **Function `ChangeWaitTime`** is for updating the wait time of customers that are in FIFO queue. Every time a customer is leaving the Queue Station, the wait time of customers remains in queue is changed.

```
void ChangeWaitTime(FIFOQ* q, double current_time);
```

5. **Math functions**

```
double urand();
double randexp( double U );
```

   – Function `urand` generate a random number uniformly distributed over the interval [0,1).

   – Function `randexp` generate random numbers from an exponential distribution with mean U.

6. **Function `runsim_config`** simply calls the `Runsim()` function which is defined in `engine.c`. The purpose of doing this is to avoid including unnecessary head file `sim.h` in the configuration program.

```
void runsim_config(double endtime);
```

7. Other functions defined in `engine.c` are from the sample simulation code.

API/Interface

The head file `interface.h` provides the access of several functions and structures as well as global variables that can be used in the configuration program.

The procedure of how the simulation runs is hidden from the user. To create the queueing network, the user can call the functions to make components. And then, call function `runsim_config()` with simulation time to start the simulation process. To get the output data, call function `GetStat`, which will return a pointer.

The structure containing the statistics is defined in this head file too. This provide the access for users to get the needed information. Besides, users can create their own function to output the statistics with customized formats based on the structure.

Input Checks

The *configuration program* is required to read data from the *configuration file*, and check for errors. As part of this assignment, a wide range of tests were performed to check that the *configuration file* read is representative of a logical queuing network queue.

```
6
0 G 15.0 1
1 Q 20.0 2
2 F 4 0.3 0.2 0.4 0.1 3 4 5 1
3 Q 7.0 1
4 Q 5.0 1
5 X
```

The first test was performed using the sample *configuration file* provided. This worked fine in generating components. Following this, the configuration file was modified to contain erroneous data of different kinds. These modified files were then read by the configuration program to check if appropriate/required error messages were displayed for the user to correct his data and prevent the execution of a simulation on an *erroneous network.* Illogical networks such as networks with no Generators, or Exits are not checked for.

General

1. Number of components mentioned in the first line should match the actual data [***test1config.txt***]

```
7
0 G 15.0 1
1 Q 20.0 2
2 F 4 0.3 0.2 0.4 0.1 3 4 5 1
3 Q 7.0 1
4 Q 5.0 1
5 X
```

   Result: error message

   No of stations mentioned does not match data input

   Program did not proceed further

2. First line must mention number of components [***test2config.txt***]

```
0 G 15.0 1
1 Q 20.0 2
2 F 4 0.3 0.2 0.4 0.1 3 4 5 1
3 Q 7.0 1
4 Q 5.0 1
5 X
```

Result: error message

Invalid Input of component count

Program did not proceed further

3. Component type entered must be sensible (G,X,F,Q) [***test3config.txt***]

```
6
0 G 15.0 1
1 % 20.0 2
2 F 4 0.3 0.2 0.4 0.1 3 4 5 1
3 P 7.0 1
4 Q 5.0 1
5 X
```

Result: error message

Invalid component type input

Invalid Input

Program did not proceed further

4. ID must be a valid number [***test4config.txt***]

```
6
0 G 15.0 1
-1 Q 20.0 2.5
2 F 4 0.3 0.2 0.4 0.1 3 4 5 1
3 Q 7.0 1
4 Q 5.0 1
5 X
```

Result: error message

Value read not positive integer

Invalid Input

Program not executed further

[Please not that, the assumption is that a component ID must be a non-negative integer]

5. Test for random characters [***test5config.txt***]

```
6
& G 15.0 1
1 Q 20.0 2
2 F 4 0.3 0.2 0.4 0.1 3 4 5 1
3 Q 7.0 1
4 Q 5.0 1
5 X
```

Result: error message

Value read not positive integer

Invalid Input

Program not executed further

Generator Component

1. Must connect to valid component [***test6config.txt***]

```
6
0 G 15.0 10
1 Q 20.0 2
2 F 4 0.3 0.2 0.4 0.1 3 4 5 1
3 Q 7.0 1
4 Q 5.0 1
5 X
```

Result: Error message

Generator connected to invalid component ID

Program did not proceed further

2. Interarrival time must be non negative/valid [***test7configA.txt*** and ***test7configB.txt***]

```
6
0 G X 1
1 Q 20.0 2
2 F 4 0.3 0.2 0.4 0.1 3 4 5 1
3 Q 7.0 1
4 Q 5.0 1
5 X
```

Result: Error message

Invalid floating point value

Invalid time input

Program did not proceed further

A check was also performed with a negative time value

```
6
0 G -15.0 1
1 Q 20.0 2
2 F 4 0.3 0.2 0.4 0.1 3 4 5 1
3 Q 7.0 1
4 Q 5.0 1
5 X
```

Result: Error message

Invalid floating point value
Invalid time input

Program did not proceed further

3. Must not have more than 4 data inputs in a line [***test8config.txt***]

```
6
0 G 15.0 1 5
1 Q 20.0 2
2 F 4 0.3 0.2 0.4 0.1 3 4 5 1
3 Q 7.0 1
4 Q 5.0 1
5 X
```

Result: Error Message

Erroneous input. QS/Generator cannot have more than 4 inputs per line

Program did not proceed further

Queue Station Component

1. Must connect to valid component [***test9config.txt***]

```
6
0 G 15.0 1
1 Q 20.0 X
2 F 4 0.3 0.2 0.4 0.1 3 4 5 1
3 Q 7.0 1
4 Q 5.0 1
5 X
```

Results: Error message

Value read not integer
Invalid Destination ID

Program did not proceed further

2. Must not connect to generator [***test10config.txt***]

```
6
0 G 15.0 1
1 Q 20.0 0
2 F 4 0.3 0.2 0.4 0.1 3 4 5 1
3 Q 7.0 1
4 Q 5.0 1
5 X
```

Results: Error message

QS connected to generator
QS connected to invalid component ID

Program did not proceed further

3. Service time must be non-negative/valid [***test11config.txt***]

```
6
0 G 15.0 1
1 Q -20.0 2
2 F 4 0.3 0.2 0.4 0.1 3 4 5 1
3 Q 7.0 1
4 Q 5.0 1
5 X
```

Result: Error message

Invalid time input

Program did not proceed further

4. Must not have more than 4 data inputs in a line [test12config.txt]

```
6
0 G 15.0 1
1 Q 20.0 2 6
2 F 4 0.3 0.2 0.4 0.1 3 4 5 1
3 Q 7.0 1
4 Q 5.0 1
5 X
```

Result: Error message

Erroneous input. QS/Generator cannot have more than 4 inputs per line

Program did not proceed further

Fork Component

1. Third data in the line must be a valid number (float or integer) [***test13configA.txt*** and ***test13configB.txt***]

```
6
0 G 15.0 1
1 Q 20.0 2
2 F X 0.3 0.2 0.4 0.1 3 4 5 1
3 Q 7.0 1
4 Q 5.0 1
5 X
```

Result: Erroneous message

Value read not integer
Invalid fork count

Program did not proceed

testconfigB.txt [similar test to above]

Result: Erroneous message

Value read not positive integer
Invalid fork count

Program did not proceed

2. Probabilities must be between 0 and 1 [***test14config.txt***]

```
6
0 G 15.0 1
1 Q 20.0 2
2 F 4 1.3 0.2 0.4 0.1 3 4 5 1
3 Q 7.0 1
4 Q 5.0 1
5 X
```

Result: Error message

Invalid probability input

Program did not proceed

3. Sum of probabilities must be 1 [***test15config.txt***]

```
6
0 G 15.0 1
1 Q 20.0 2
2 F 4 0.3 0.2 0.4 0.2 3 4 5 1
3 Q 7.0 1
4 Q 5.0 1
5 X
```

Result: Error message

Invalid Probabilities for fork
Fork error

Program did not execute further

4. Number of probabilities must equal number of destinations which must equal number of fork connection [***test16config.txt***]

```
6
0 G 15.0 1
1 Q 20.0 2
2 F 4 0.3 0.2 0.4 0.1 3 4 5 1 6
3 Q 7.0 1
4 Q 5.0 1
5 X
```

Result: Error message

Erroneous input in fork input line. More data than required

Program did not execute further

Exit Component

1. Must not have more than 2 data inputs [***test17config.txt***]

```
6
0 G 15.0 1
1 Q 20.0 2
2 F 4 0.3 0.2 0.4 0.1 3 4 5 1
3 Q 7.0 1
4 Q 5.0 1
5 X L
```

Result: Error Message

Erroneous input! Exit component cannot have more than 2 input arguments

Program did not execute further

Test Run [*testRun.txt*]

This test was performed on a network as defined here with an interarrival time of 5

```
1
0 G 5 1
1 Q 0.2 2
2 F 2 0.5 0.5 3 4
3 Q 3.0 8
4 Q 2.5 8
5 X
```

The simulation time was 10.0.

```
Initial event list:
Event List: 5.000000
Processing Generate event at time 5.000000, Component=0
Schedule Arrival from Generator at time: 5.000000
Schedule Generate at time: 9.004132
Event List: 5.000000 9.004132
Processing Arrival event at time 5.000000, Component=1
1Schedule Departure at time: 5.175049
Event List: 5.175049 9.004132
Processing Departure event at time 5.175049, Component=1
Schedule Arrival from QS 1 at time: 5.175049. FIFO Queue Length: 0.
Event List: 5.175049 9.004132
Processing Arrival event at time 5.175049, Component=2
Schedule Arrival from Fork at time: 5.175049. DestID = 3
Event List: 5.175049 9.004132
Processing Arrival event at time 5.175049, Component=3
1Schedule Departure at time: 11.298325
Event List: 9.004132 11.298325
Processing Generate event at time 9.004132, Component=0
Schedule Arrival from Generator at time: 9.004132
Schedule Generate at time: 17.282737
Event List: 9.004132 11.298325 17.282737
Processing Arrival event at time 9.004132, Component=1
1Schedule Departure at time: 9.028144
Event List: 9.028144 11.298325 17.282737
Processing Departure event at time 9.028144, Component=1
Schedule Arrival from QS 1 at time: 9.028144. FIFO Queue Length: 0.
Event List: 9.028144 11.298325 17.282737
Processing Arrival event at time 9.028144, Component=2
Schedule Arrival from Fork at time: 9.028144. DestID = 3
Event List: 9.028144 11.298325 17.282737
Processing Arrival event at time 9.028144, Component=3
Enqueue. Current queue length: 2.
Event List: 11.298325 17.282737
```

The printed information shows the progress of the simulation. Different types of events being processing or scheduled are printed out, with the component ID where the events occurred. Also, the FIFO queue length of queue station is printed out when a customer is leaving.

M/M/1 Queue system [1]

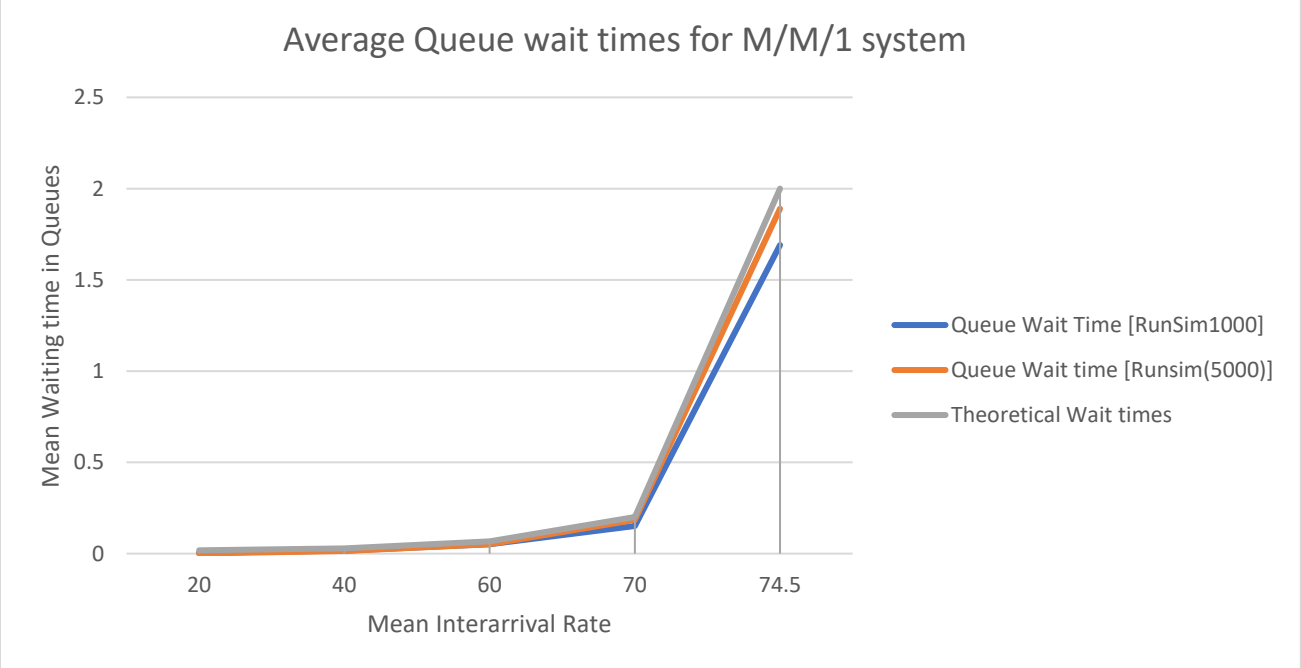
The M/M/1 queue is a straight-forward system to analyze in that the model assumes Poisson arrivals (this means the interarrival times are exponentially distributed as is the setup in the simulation setup in assignment 3). This system involves also, a single station with exponentially distributed processing times, again stacking up with the assumptions of the assignment. Furthermore, M/M/1 queues work on a *first-come-first* serve basis – the same treatment given to the queue stations in the assignment simulation.

For M/M/1 queues, the average time spent in a queue is given by

$$Cycle\ time_{Queue} = \left[\frac{utilization}{(1 - utilization)}\right] = \left[\frac{\frac{Arrival\ rate}{Processing\ rate}}{1 - \left(\frac{Arrival\ Rate}{Processing\ rate}\right)}\right] * Processing\ time$$

In this particular case the service/process rate used is 75 customers per unit time and the arrival rate is varied from 0.01 to 74.5. At *Arrival rate=Process rate*, the expression breaks down. The simulation runs for the M/M/1 system was performed varying the arrival rate and plotting the mean waiting time in queues as shown below. Two sets of simulation plots were made with different run durations (1000 and 5000). Further, the formula above was used to calculate the theoretical queue wait times in an M/M/1 queue. All plots and raw data are shown respectively in the plot and table below.

We note that the simulation results and theoretical results match quite closely. The simulation results are closer to the theoretical results the longer the simulation run.



Sno	Arrival Rate	Mean InterArrival Time	Queue Wait Time [RunSim1000]	Queue Wait time [Runsim(5000)]	Utilization	Theoretical Wait times
1	0.01	100	0	0	0.0001	0.013335111
2	20	0.05	0.004874	0.0004763	0.2667	0.018181818
3	40	0.025	0.015876	0.015214	0.5333	0.028571429
4	60	0.016666667	0.052041	0.051761	0.8	0.066666667
5	70	0.014285714	0.151074	0.184457	0.9333	0.2
6	74.5	0.013422819	1.69081	1.89	0.9933	2

Simulation Run

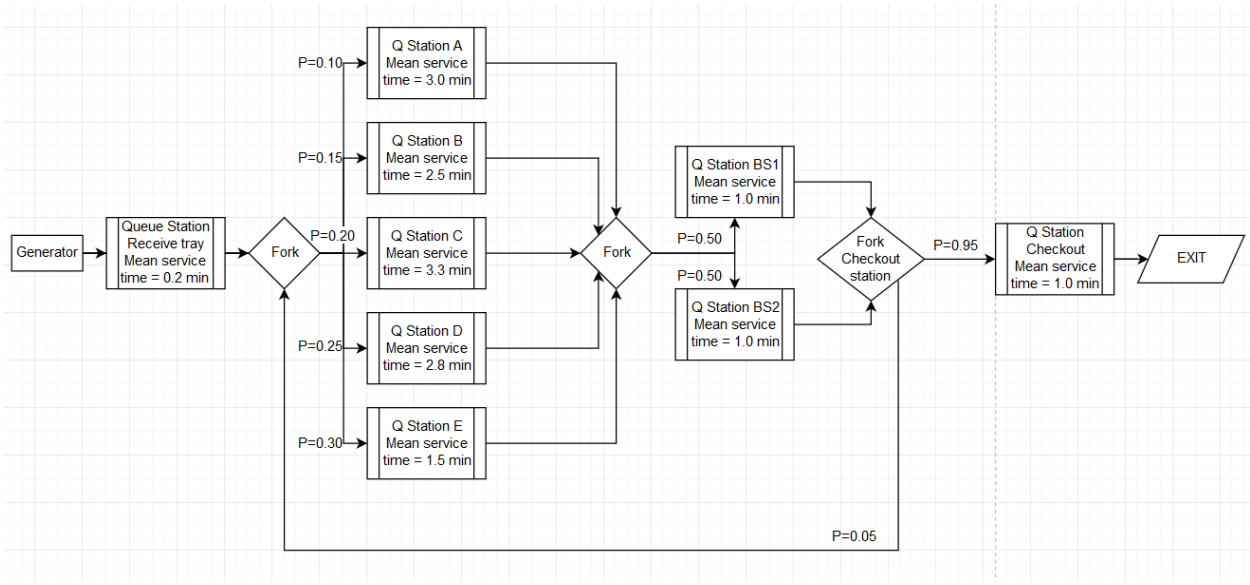
The assignment requires that the simulations using the simulation program be performed on a predefined network that represents a food court. Based on the description provided, a food court network was created, and a representation has been provided below. This representation is for the case with only 1 checkout station.

The simulation is performed on three cases

1. Case 1: Network with only one checkout station
2. Case 2: Network with 2 checkout stations
3. Case 3: Network with 3 checkout stations

In all three cases above, the mean interarrival rate was checked for the following values = {0.02, 0.08, 0.16, 0.20, 1.0, 2.0, 4.0, 10.0}. All values are in minutes, and are chosen around the service times of the queue stations in the network.

Network to be generated



Configuration file for Network

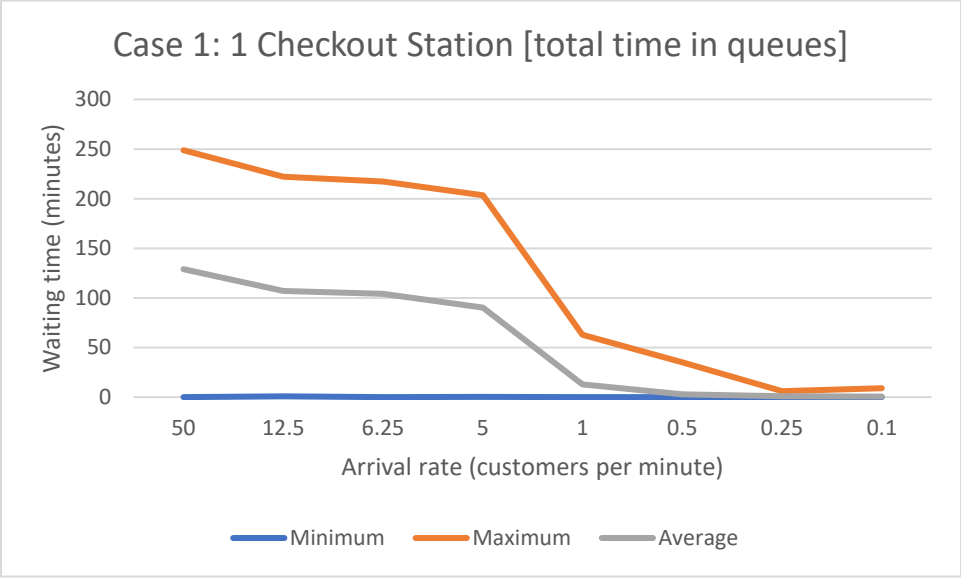
Case 1: One Checkout station

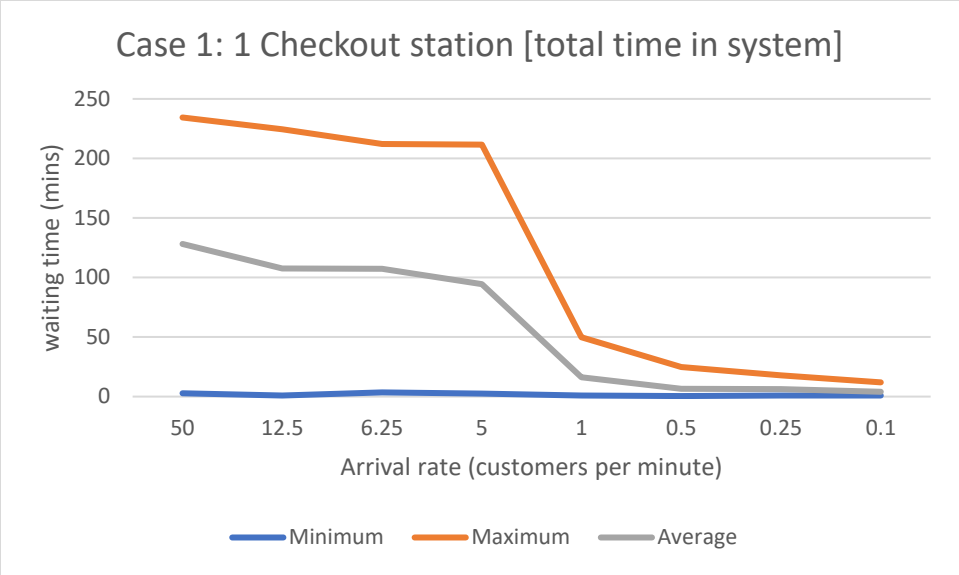
The configuration input file for the first case is as follows. This file was used as the input to run the simulation. <interarrival time> was varied as indicated above.

```
14
0 G <interarrival time> 1
1 Q 0.2 2
2 F 5 0.10 0.15 0.20 0.25 0.30 3 4 5 6 7
3 Q 3.0 8
4 Q 2.5 8
5 Q 3.3 8
6 Q 2.8 8
7 Q 1.5 8
8 F 2 0.50 0.50 9 10
9 Q 1.0 11
10 Q 1.0 11
11 F 2 0.95 0.05 12 2
12 Q 1.0 13
13 X
```

Results for case 1

The plot of the waiting *times in queues* and in the *system* is plotted as shown in the two graphs below.





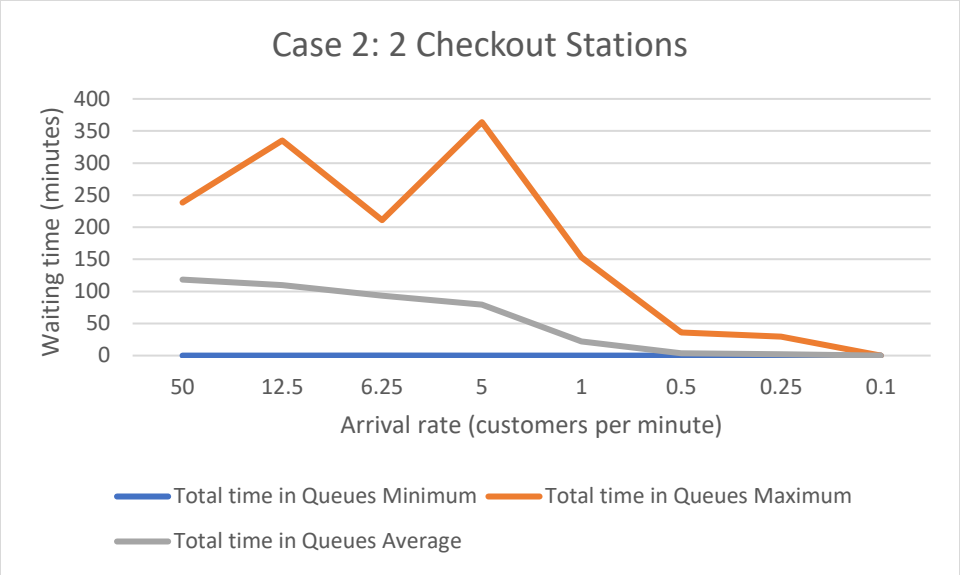
Case 2: 2 Checkout stations

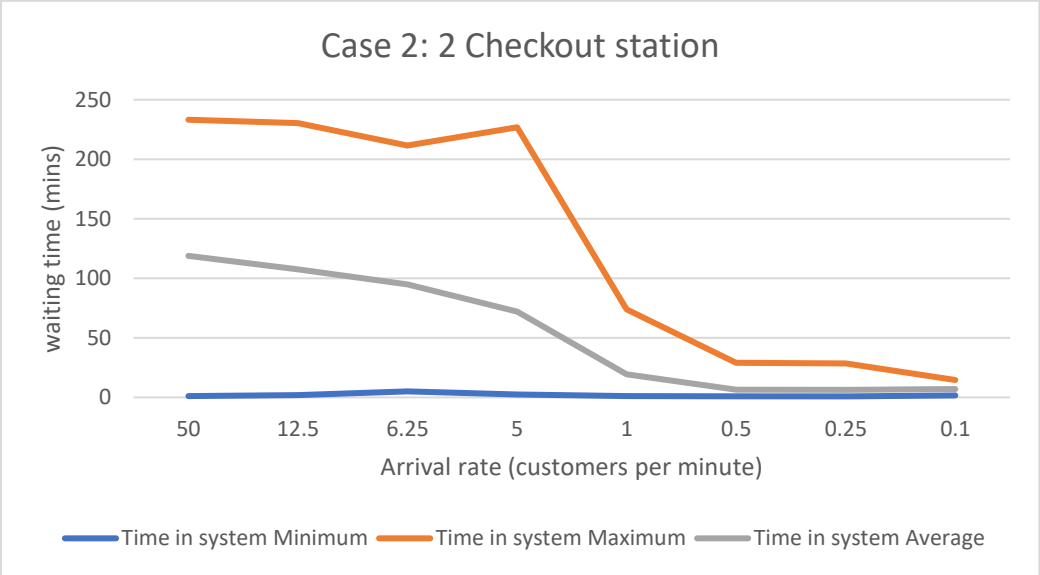
The configuration input file for the second case is as follows. This file was used as the input to run the simulation. <interarrival time> was varied as indicated above.

```
15
0 G <interarrival time> 1
1 Q 0.2 2
2 F 5 0.10 0.15 0.20 0.25 0.30 3 4 5 6 7
3 Q 3.0 8
4 Q 2.5 8
5 Q 3.3 8
6 Q 2.8 8
7 Q 1.5 8
8 F 2 0.50 0.50 9 10
9 Q 1.0 11
10 Q 1.0 11
11 F 3 0.475 0.475 0.05 12 13 2
12 Q 1.0 14
13 Q 1.0 14
14 X
```

Results for case 2

The plots below show the variation of *time in queue,* and *time in system* for case 2.

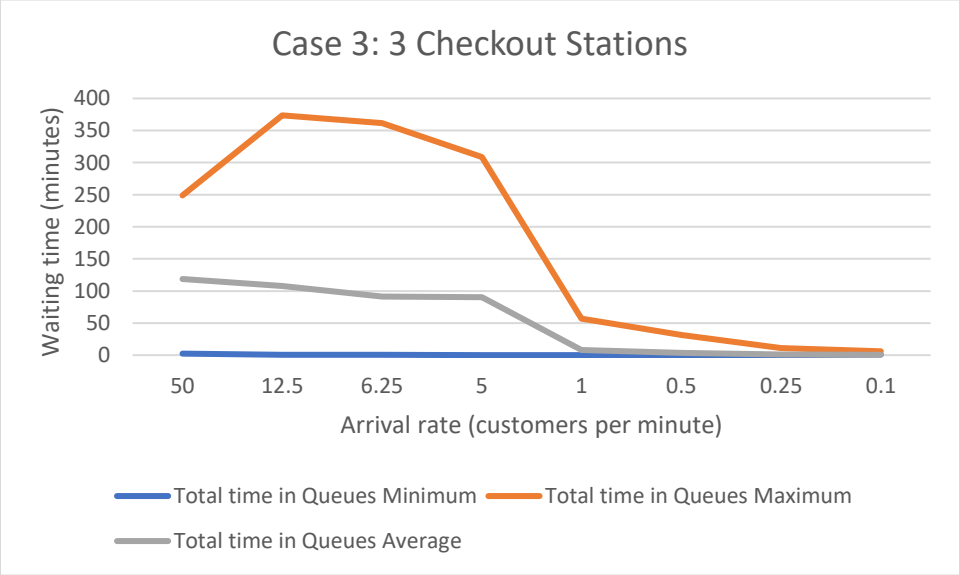

Case 2: 2 Checkout Stations
Waiting time (minutes)
400
350
300
250
200
150
100
50
0
50
12.5
6.25
5
1
0.5
0.25
0.1
Arrival rate (customers per minute)
Total time in Queues Minimum
Total time in Queues Maximum
Total time in Queues Average

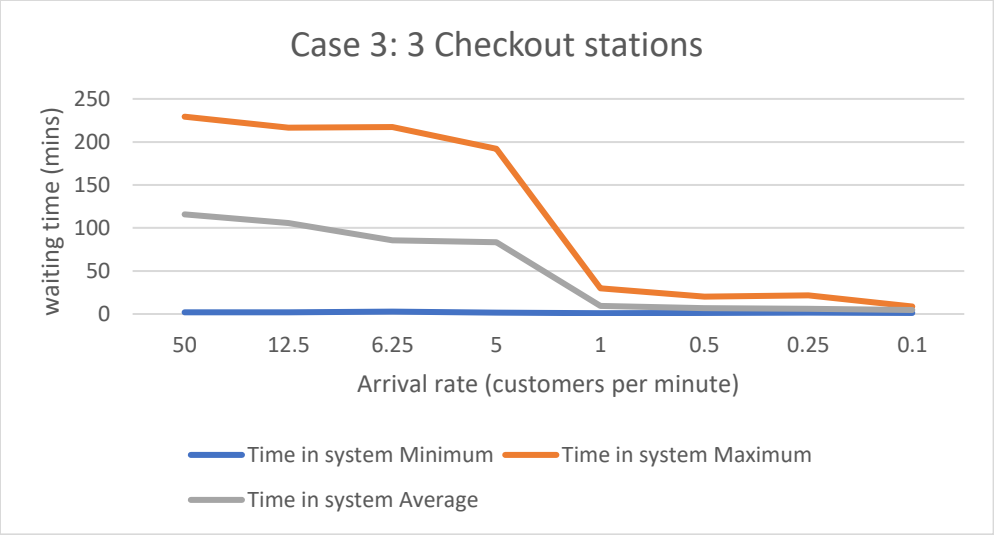

Case 2: 2 Checkout station
waiting time (mins)
250
200
150
100
50
0
50
12.5
6.25
5
1
0.5
0.25
0.1
Arrival rate (customers per minute)
Time in system Minimum
Time in system Maximum
Time in system Average

Case 3: 3 checkout stations

The configuration input file for the third case is as follows. This file was used as the input to run the simulation. <interarrival time> was varied as indicated above.

```
16
0 G <interarrival time> 1
1 Q 0.2 2
2 F 5 0.10 0.15 0.20 0.25 0.30 3 4 5 6 7
3 Q 3.0 8
4 Q 2.5 8
5 Q 3.3 8
6 Q 2.8 8
7 Q 1.5 8
8 F 2 0.50 0.50 9 10
9 Q 1.0 11
10 Q 1.0 11
11 F 4 0.316 0.317 0.317 0.05 12 13 14 2
12 Q 1.0 15
13 Q 1.0 15
14 Q 1.0 15
15 X
```



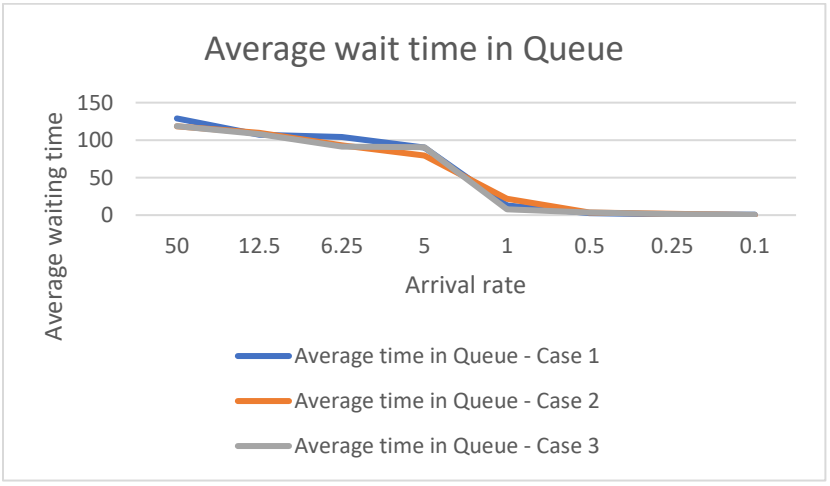


Discussion of results

We note that for cases 1,2 and 3, with decreasing arrival rates, the trend for in-system and in-queue waiting times in downwards. As the number of checkout counters increases from 1 to 3, again we notice a decrease in waiting times.

This is logical, given that the service rates of the stations is constant, an increase in the arrival rates will mean that queues will build up and customers will have to wait longer to be served. On the other hand, as the arrival rate decreases, queues tend to not build up as customers rarely arrive while a current customer is being served (for low arrival rates)

Having only one checkout stations could mean that at high arrival rates, it would serve as a bottleneck and increase waiting times. As the number of checkout stations is increases, one would expect that queue build up due to lack of checkout counters would be mitigated. However, it is noted from the results that an increase in the checkout station results in only a marginal improvement in waiting times. This is possible if a different queue station from the checkout station serves as a bottleneck.



Sample output file example [Case 3.1: Case 3 with highest arrival rate]

For case 3 with arrival rate of 50 (interarrival time of 0.02).

```
        Statistics Output
No. of customers entering system is 12175
No. of customers exiting the system is 383
Minimum time a customer remains in the system is 2.788059
Maximum time a customer remains in system is 226.813245
Average time a customer remains in the system is 121.193576
Minimum total time a customer waits in queues is 0.000000
Maximum total time a customer waits in queues is 499.828044
Average total time a customer waits in queues is 135.048566
Avg time in QS 1: 40.774408
Avg time in QS 3: 49.518271
Avg time in QS 4: 70.908376
Avg time in QS 5: 29.922092
Avg time in QS 6: 73.947623
Avg time in QS 7: 68.171068
Avg time in QS 9: 10.303772
Avg time in QS 10: 21.450097
Avg time in QS 12: 8.614825
Avg time in QS 13: 8.777560
Avg time in QS 14: 8.374343
```

All other results files have been attached with submissions and tagged as 1.1 (Case 1 scenario 1), 1.2 and so on. For further information see appendix. These files contain all statistics, not just the ones plotted as was required in the assignment.

References

[1] [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_71/rtref/itoi.htm.

[2] [Online]. Available: https://www.gnu.org/software/libc/manual/html_node/String-Length.html.

[3] M. S. Wallace Hopp, Factory Physics, 1995.

Appendix

You may find the output file names here for each simulation case along with results needed. The actual output files are attached with the submission folder.

Case 1 Raw data

					Time in system				Total time in Queues		
Sno	Arrival rate	InterArrival time	Outfile		Minimum	Maximum	Average		Minimum	Maximum	Average
1	50	0.02	1.1		2.86	234.38	128.18		0	248.83	128.93
2	12.5	0.08	1.2		0.82363	224.56	107.54		0.7957	222.0338	107.14
3	6.25	0.16	1.3		3.514	212.13	107.27		0	217.139	104.1
4	5	0.2	1.4		2.389	211.59	94.36		0.21	203.47	90.216
5	1	1	1.5		0.969	49.84	16.16		0	62.7	12.77
6	0.5	2	1.6		0.453	24.914	6.548		0	35.07	2.809
7	0.25	4	1.7		1.012	17.89	6.208		0	5.998	0.837
8	0.1	10	1.8		0.9	11.932	4.03		0	8.976	0.553

Case 2 raw data

					Time in system				Total time in Queues		
Sno	Arrival rate	InterArrival time	Output filename		Minimum	Maximum	Average		Minimum	Maximum	Average
1	50	0.02	2.1		1.15	233.2	118.87		0	238.64	118.42
2	12.5	0.08	2.2		2.00439	230.438	107.63		0	335.15	109.76
3	6.25	0.16	2.3		5.081	211.673	94.97		0.115	210.974	93.176
4	5	0.2	2.4		2.558	226.944	72.054		0	363.78	79.63
5	1	1	2.5		1.119	74.078	19.275		0	153.22	22
6	0.5	2	2.6		0.918	29.002	6.547		0	36.084	3.433
7	0.25	4	2.7		0.938	28.408	6.35		0	29.638	1.879
8	0.1	10	2.8		1.629	14.64	6.94		0	0	0

Case 3 Raw Data

					Time in system				Total time in Queues		
Sno	Arrival rate	InterArrival time	Output file name		Minimum	Maximum	Average		Minimum	Maximum	Average
1	50	0.02	3.1		2.01368	229.3234	115.79		2.36517	248.774	118.65
2	12.5	0.08	3.2		2.07756	216.5352	105.57		0.37307	373.6711	107.94
3	6.25	0.16	3.3		2.73963	217.395	85.669		0.58246	361.7185	91.384
4	5	0.2	3.4		1.65961	192.0754	83.423		0.07023	308.5894	90.497
5	1	1	3.5		0.91649	29.77102	9.2362		0.13392	56.66206	7.8514
6	0.5	2	3.6		1.22283	19.98263	6.8372		0.14608	31.38925	3.4051
7	0.25	4	3.7		1.66698	21.68408	5.9364		0.00417	10.96729	1.236
8	0.1	10	3.8		1.35683	8.689806	4.6457		1.28339	5.977736	0.5558