***Report***

*Zongyi Li, Zhiquan Zhang*

1. **Introduction and Goal**

This assignment is about a simulation for a queueing network. For example, a fast food restaurant can be regarded as a queueing network, s.t, people will wait in a line first to get a tray and then go to a service station for the purpose of getting some kinds of food. After getting the first kind of food, people can randomly go to other service station to get what they need. The whole system can be seen as a queueing network. And this assignment is to develop a model like such systems to help analyze the useful information, which can be helpful for the improvement of the service.

Our target is to 1. use C language to develop a library, which contains many functions that can help build a queueing network. 2. Use C language to develop a configuration file, which contains the structure information of the system like the number and type of different components. And finally, we can let our program check the configuration file and run those files of correct format.

1. **Library and Configuration**

In this project, we didn’t use some other APIs or existing data structures/ libraries to implement the library. We define all the data structures in application and engine files.

**Implementation of the library:**

The library can be divided into two parts, one is engine and another is application.

In files engine.h and engine.c, we simply used the file provided by instructor, the functions included are as below.

|  |  |
| --- | --- |
| Function | Functionality |
| RunSim() | Used to remove the event with the highest timestamp in the priority queue, and input it into EventHandler() function in application files. |
| Schedule() | Used to insert the new event from application files into the priority queue. |
| CurrentTime() | Used to return the system current time, the default value is 0.0. |
| \*Remove() | Used to remove the element from priority queue. |
| PrintList() | Print the list of event to the screen to monitor the process |

In files application.h and application.c, the main struct we define are as below.

|  |  |  |
| --- | --- | --- |
| Struct | Properties | Description |
| Customer | CreationTime | Record the time customer is generated |
| ArrTime | Record the time customer arrive at station, and being added into a queue |
| totalWait | Total waiting time in the system |
| CustomerList | amount | The total number of customers in the list, which is used to record the information of all the customers |
| \*head | The head of the list |
| EventData | EventType | Type of event(GENERATE, ARRIVAL, DEPARTURE) |
| \*Cust | Arriving or departing customers; unused for GENERATE events |
| CompID | ID of component where customer created, arrives or departs |
| FIFOQueue | \*first | Pointer to first customer in queue |
| \*last | Pointer to last customer in queue |
| Generator | IntArrTime | Mean interarrival time for generated components |
| DestComp | ID of next component customers are sent to |
| Queue\_Station | \*Q | A FIFO queue in station |
| AvgWaitTime | Average total waiting time for all customers in the station |
| totalWait | Total waiting time in the station |
| inLine | Number of customers in line |
| DestComp | ID of next component customers are sent to |
| Fork | num | number of ports it has |
| \*probability | probabilities for each port |
| \*DestComp | D of next component customers are sent to |
| Exit | Count | The number of customers that exited at this component |
| totalTime | System total service time and wait time for all customers |
| maxTime | The maximum time a customer stayed in the system |
| minTime | The minimum time a customer stayed in the system |
| Component[] | ComponentType | GENERATOR, QUEUE\_STATION, FORK, EXIT |
| \*Comp | Pointer to information on component (Generator, Exit struct, etc.) |

In application.c, we also defined global variables to record the customer information.

|  |  |
| --- | --- |
| Global variable | Description |
| \*CustList | Used a linked list to record all customer information |
| totalExits | Total number of customers that exited from the system |
| numComponents | Number of components in the system |

In implementation. Firstly we initialize the Customer List, and make all the components according to the input file. Initialize all the components in each function. While making the Generator component, we schedule the first event and put it into the engine. Then we iteratively remove the event from the priority queue and use EventHandler to handle each different type of event.

If the event type is GENERATOR, we make a new customer, and add it to the Customer List. Then we schedule ARRIVAL event at the component it connected. Then we schedule next GENERATION event with timestamp generated from an exponential distribution with mean U, where U is the mean interarrival time of generation.

If the event type is ARRIVAL, then we need to check which component it arrives. If the customer arrives at Queue Station component, if the queue in this station is empty, we schedule DEPARTURE event for this customer with timestamp at current time. The component ID is still the same as this component; If the queue is not empty, we add the customer in to queue, and record its arrive time. If the customer arrives at Fork component, we firstly random select a port with their probabilities. Then we schedule an ARRIVAL event at the port we choose. If the customer arrives at Exit component, we simply record its leaving time, and calculate its total time in the system.

If the event type is DEPARTURE, we can only have a DEPARTURE event at Queue Station. We firstly schedule ARRIVAL event for the customer who is the first in queue at the component it connected, with timestamp generated from an exponential distribution with mean V, where V is the mean service time of this station, and remove this customer from the queue. Then if the queue in this station is not empty, we schedule DEPARTURE event for the next customer (who is now the first of the queue), with timestamp at current time. The component ID is still the same as this component.

After running RunSim() in the time we set, we can calculate different statistics for the whole system, and output them in files.

We wrote the random number generation functions in random.h and random.c and include them in application.c. There’re three functions in these files.

|  |  |
| --- | --- |
| Functions | Functionality |
| rand\_init() | Initialize random function to set time seed |
| urand () | Generate a random number uniformly distributed over the interval [0,1) |
| randexp() | Returns -U\*(log(1-urand())), where log() is the C function defined in <math.h>, and U is the mean |

**Implementation of the configuration:**

The configuration file only has a struct that help save the information of each read component.

|  |  |  |
| --- | --- | --- |
| Struct | Properties | Description |
| Component\_read | ComponentType | The type of the component that’s read |
| Comp | The concrete component’s pointer, which can be from generator, fork, queue or exit. |

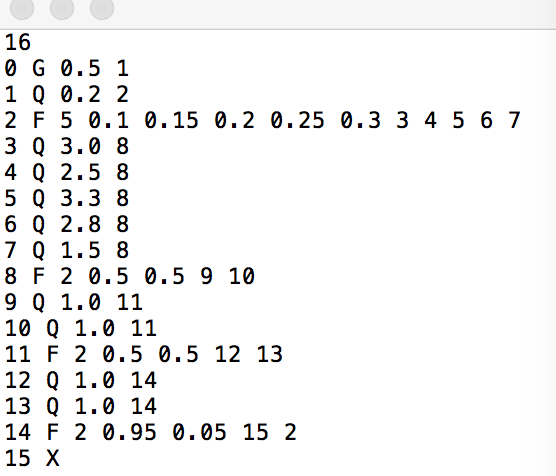
The below functions can be called directly by users to help set up a system.

|  |  |
| --- | --- |
| Functions | Functionality |
| read\_file() | It can read the component information and save them in the struct. It can also check the file’s format and content’ legality. |
| SetupSystem() | It can set up a system according to the content in the struct array that saves the information of a system. |

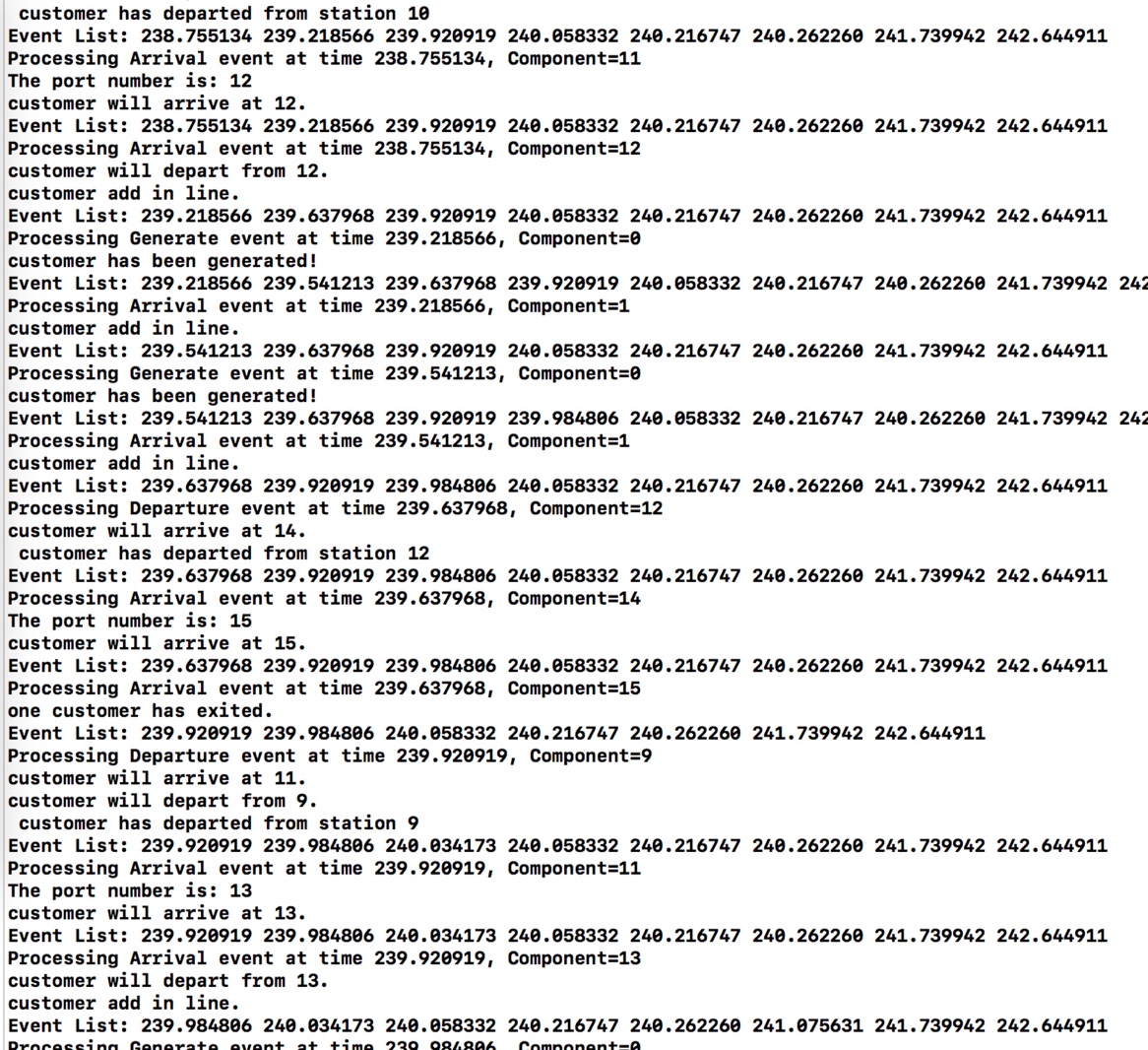
In config.c file, there are some other functions that can check for the format or content errors for the configuration files but they are all hidden for users. For users, they can just use read\_file() function to get and check a file’s information and then call SetupSystem() function to set up a complete system. After establishing a system, users can directly run the system.

1. **Test procedure and evidence of correctness of the code**

First of all, based on the format of configuration file, writing a configuration program as the image below.

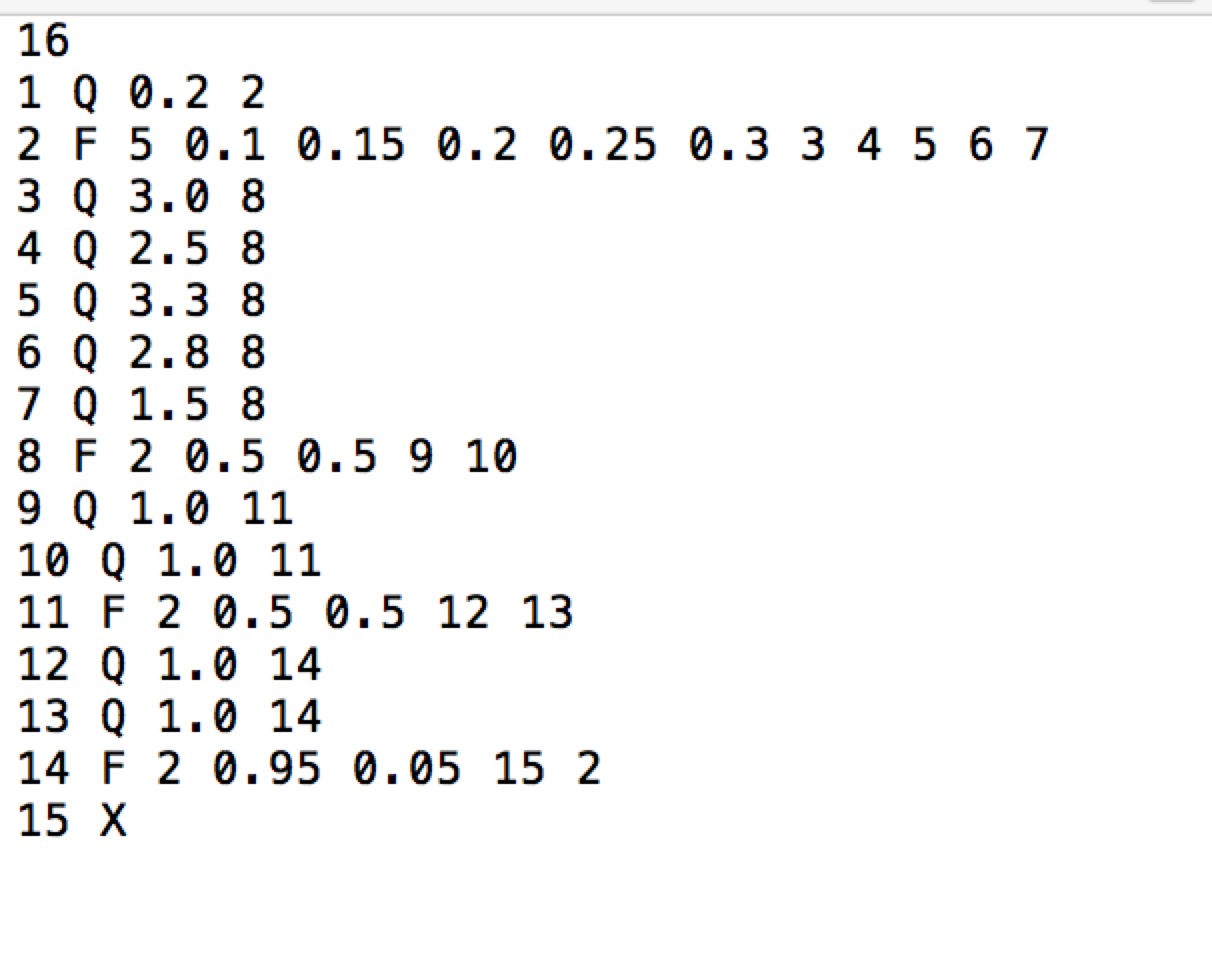


Obviously, this file has the correct format of input. And it could be executed correctly. And the system can run the simulation.

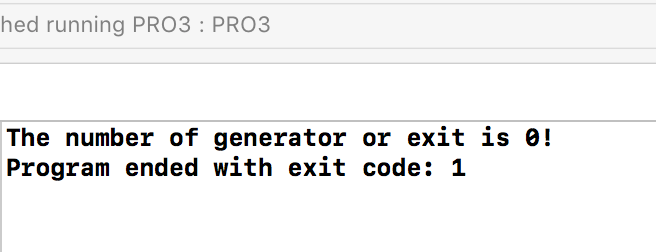


The events can happen in a time order and they will be scheduled by the system. So the validation of simulating a correct format of configuration file has been done. Then I want to verify the validation of the error checking.

Below is an image that contains the wrong format of configuration file, it doesn’t contain the component Generator.



If we run the simulation, we can see that the program reminds that the number of generator or exit is 0, so it can not be executed.



1. **Implementation of simulation software.**

According to the requirement, we simulate the food court system in our program.

First of all, let’s analyze the time data, varied based on the variation of number of check out station. It’s not far to see that roughly with the increment of the number of check out station, the average waiting time decreased most apparently. As for the maximum and minimum, they don’t change greatly.

As for the time in system, it’s also obvious that the total time in a system of those who exited the system also decreased roughly with the increasement of the check out station.

Then let’s observe the variation of the rate. It’s very obvious that with the decreasing of rate, the number of customers entering the system drop down and approximates the number of customers exited the system.

When talking about the waiting time’s variation, it’s not far to see that with the decreasing of the rate, the waiting time for customers also drop down roughly.

As for the time in a system, it’s also obvious that they all drop down with the decreasing of rate.

1. **M/M/1 result.**
2. **In comparision with literature**