MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Queues Simulator

Chiș George
Grupa 2021

The following project is a simulation application aiming to analyse queuing based systems for determining and minimizing clients' waiting time.

Queues are commonly used to model real world domains. The main objective of a queue is to provide a place for a "client" to wait before receiving a "service". The management of queue-based system is interested in minimizing the time amount their "clients" are waiting in queues before they are served. One way to minimize the waiting time is to add more servers, i.e. more queues in the system (each queue is considered as having an associated processor) but this approach increases the costs of the service supplier. When a new server is added the waiting customers will be evenly distributed to all current available queues.

The application should simulate (by defining a simulation time tSimulation) a series of N clients arriving for service, entering Q queues, waiting, being served and finally leaving the queues. All clients are generated when the simulation is started, and are characterized by three parameters: ID(a number between 1 and N), tArrival (simulation time when they are ready to go to the queue; i.e. time when the client finished shopping) and tService (time interval or duration needed to serve the client by the cashier; i.e. waiting time when the client is in front of the queue). The application tracks the total time spend by every customer in the queues and computes the average waiting time. Each client is added to the queue with minimum waiting time when its tArrival time is greater
than or equal to the simulation time ($tSimulation \geq tArrival$).

The following data should be considered as input data read from a text file for the application:
- Number of clients (N);
- Number of queues (Q);
- Simulation interval (tSimulation_max);
- Minimum and maximum arrival time ($tArrival\_min \leq tArrival \leq tArrival\_max$);
- Minimum and maximum service time ($tService\_min \leq tService \leq tService\_max$);

The output of the application is a text file containing a log of the execution of the application and the average waiting time of the clients, as shown in the following example.

Code content

The application contains 4 classes :

- **Client.java** : this one handles all the attributes and functionality of the clients (each object instantiated represents a virtual client). The class does not have so much functionality , as it holds the information (attributes) of each client. The mathods of the class are mainly setters and getters , a constructor and overrided toString() methods for printing the information.

- **Queue.java** : the queues class designs the functionality of each queue in which clients will be temporary “stored” (more like waiting). It has as a private attribute a LinkedList of clients to store the waiting clients in the queue. A LinkedList is probably not the most useful way to store this clients, as only one client can wait in a queue at a time, so the LinkedList is not used to its full extent , but it helped me using its built-in methods. Then , it has a constructor, setters and getters for the private attributes , a method that adds clients to the queue. This class implements the Runnable interface and it starts a thread each time an object is instantiated.

- **ManageWaitingQueues.java** : this is where most of the functionality of the program is being handled. This class will be instantiated once in the Main class , taking its attributes from the input text file. The class stores all the information that the simulation requires : N (no. of clients) , Q (no. of queues) , tSimulationMax, tArrivalMin, tArrivalMax, tServiceMin, tServiceMax .

Then it generates clients with random attributes situated in the given bounds , and it populates a LinkedList of clients , and another of queues , according to the data taken from the input text file.

The waitingClients LinkedList stores all the clients that are not yet in a queue (the simulation time is currently smaller than their arrival time) . When tSimulation reaches the smallest tArrival of all the clients, this client will be added to the first queue, and it will stay there for tService “seconds” . If the simulation reached another clients’ tArrival , if there is an unoccupied (“closed”) queue , it will enter the queue and wait there for his tService time , and so on with all the clients. The simulation runs until the tSimulationMax is reached or until there are no more clients in the waiting list.

All this functionality is contained in the run() method of the class , which starts a thread (the class implements the Runnable interface).

The results are printed in the output file.

- **Main.java** : the main class takes the input information from the text file , instantiates the ManageWaitingQueues class that starts a thread and the simulation is run on that thread.

There is a set of input and output files attached to the folder that verify the functionality of the application. The average waiting time of a client is also tracked and displayed at the end of the output file.

Here is a screenshot of a example of a input text file (low number of clients):

```
3
2
100
1,5
2,4
```

And the resulting output file:

```

Time 0
Waiting clients: (1,2,3);(2,4,2);(3,1,4);
Queue 1: closed
Queue 2: closed

Time 1
Waiting clients: (1,2,3);(2,4,2);
Queue 1: (3,1,4);
Queue 2: closed

Time 2
Waiting clients: (2,4,2);
Queue 1: (3,1,3);
Queue 2: (1,2,3);

Time 3
Waiting clients: (2,4,2);
Queue 1: (3,1,2);
Queue 2: (1,2,2);

Time 4
Waiting clients: (2,4,2);
Queue 1: (3,1,1);
Queue 2: (1,2,1);

Time 5
Waiting clients: (2,4,2);
Queue 1: closed
Queue 2: closed

Time 6
Waiting clients:
Queue 1: (2,4,2);
Queue 2: closed

Time 7
Waiting clients:
Queue 1: (2,4,1);
Queue 2: closed
Average waiting time: 3.0
```