README

COP290 Assignment 6

Vishal Bindal
2018CS50425

Instructions to run

Go to the parent directory containing Makefile and src, and run the following commands.

```
make
cd bin
./qSim #customers #tellers simulationTime averageServiceTime
```

e.g. for last command is

```
qSim 100 4 60 2.3
```

Note: 10 test cases and their output are written in TestCases.txt

Output

- The summary regarding the statistics of the 2 simulations (single queue vs one queue per teller) are printed on the console
- The plot between average time spent in bank vs no of tellers is saved as TimeSpent.png in the ./output subdirectory (relative to parent directory). **The plot is for 50 tellers, starting from #tellers = 2, to #tellers = 51.**

Description

Structures implemented

- Node and Queue structures have been implemented. Node contains a void* value field which can point to an event or customer object (based on the purpose the queue is used for) In Queue, function pointers have been used for implementing the functions addAtTail (to add node at tail), addAppropriate (to add node appropriately at a position in the queue acc to a parameter, say arrival time), and pop (to pop node from the head)
- Structures have been made for Customer and Teller. Customer struct stores id, arrivalTime, waitingTime and serviceTime. Teller stores id, and current customer id.
    - Functions related to Customer and Teller:
    - createCustomer:

        - initialise a customer based on id and arrival time
    - createTeller:
        - initialise a teller based on id
- A structure is made for TellerQueue, which stores a pointer to a Queue array (for implementing single-queue or multi-queue simulations accordingly). It also has function pointers for implementing the functions add (to add a customer), pop (to get a customer from the Teller Queue).
    - Functions related to TellerQueue:
    - getShortestTellerLine
        - It is used to find the index of the appropriate teller line to which a customer should be added after he/she has arrived
        - The index of the queue having the shortest length is returned. If there are multiple queues with the same shortest length, then they are added to an array shortestLines, and a random queue is picked from these.
    - addCustomer
        - The queue to which the newly arrived customer should be added is found using getShortestTellerLine.
        - The customer object is added to the queue found above
    - popCustomer
        - This is used when a teller is free and a customer has to be popped from a teller line
        - In case of single queue system, the customer is popped from the head of the single teller line
        - In case of multiple lines, the line corresponding to the teller is first checked. If it is non-empty, customer is popped from its head. Else, the customer is taken from a randomly selected teller line.
    - createTellerQueue
        - for initialising the tellerQueue object based on the queueing model - single line or multiple lines
- A structure Event has been made. It has a void* person field, which can point to a customer object or a teller object. It has a function pointer void (*action), which points to the function to be executed when the event is called. It also has function pointers addToQueue and popFromQueue for adding and removing the event from the event queue.
    - Functions related to event:
    - addEventToQueue:
        - for adding event object to event queue
    - popEventToQueue:
        - for popping event object from event queue
    - createEvent
        - for initialising event object
    - addEventNode

        - A function for adding a node containing an event object to the queue. It goes into the addAppropriate function pointer of the event queue.
        - It adds the node to the event queue on the basis of eventTime field of the event. (The event having lesser event time is closer to the head of the queue)
    - createEventQueue
        - for initialising the event queue and setting addAppropriate function pointer of the queue to addEventNode.
    - The action functions for 3 types of events are documented below

Action functions for various events

- There are 3 kinds of actions that can be done for an event. 3 functions have been made. The function pointer in struct Event points to one of these accordingly
- actionCustomerArrival
    - This is for a new customer who has arrived at the bank.
    - The clock is advanced to the event time
    - The customer is added to the teller queue
- actionTeller
    - This is when the teller has either finished serving a customer, or just finished his/her idle time
    - The clock is advanced to the event time
    - If tellerQueue.pop returns a customer (i.e. if the teller queue is not empty), then 2 new events are created - one a teller event corresponding to the next time the teller will be free, and the other a customer event corresponding to when the customer will finish service and exit from the bank. The service time is generated randomly according to the formula given. Both these events are added to the event queue
    - If no customer is present, then a random idle time is generated in the range (1,150) and a teller event is created, which is added to the event queue
- actionCustomerExit
    - This is when the customer has finished services and is exiting from the bank
    - The statistics are updated - the customer's total time spent, max waiting time, and no of customers served
    - The objects associated with the event are freed from memory

Function pointers used:

- In struct Queue:
    - void (*addAtTail)(struct Queue*, Node*)
    - void (*addAppropriate)(struct Queue*, Node*)
    - Node* (*pop)(struct Queue*)

- In struct Event:
    - void (*addToQueue)(struct Event*)
    - struct Event* (*popFromQueue)(Queue*)
    - void (*action)(struct Event*)

Workflow

- First an array arrivalTimes of length #customers is created. It is filled with random arrival times using the formula given. These arrival times are used for both queueing cases - one queue per teller and single queue.
- run() is called, first for mode 1 (one line per teller) and then for mode 2 (single line)
- The clock (called masterClock) is set to zero, and all the statistics (totalTellerServiceTime, totalTellerIdleTime, maxCustomerTime, etc) to be collected are initialised
- Event queue is initialised. Tellerqueue is initialised based on queueing model (common line or multiple lines)
- Customer objects are initialised, based on their arrival times calculated above. Event objects are created for all customers and added to the event queue.
- Teller objects (acc to #tellers input parameter) are initialised. Event objects are initialised for each teller with an idle time in the range (1,600). All these events are added to the event queue
- The simulation is played out till (1) the event queue becomes empty, or (2) all customers are served. In each iteration, an event object is popped from the event queue, its action method called, and the memory related to the event freed from memory.
- In the end, the required statistics are calculated and displayed on the console.
- After running the simulation for both modes, it is run again 50 times in mode 2 (single queue system) for collecting data for gnuPlot. In each simulation, the average time spent by a customer in the bank is collected. Finally, the plot is made and saved in ./output

Note

- **The simulation is run till all the customers get served**, so the total no of customers served is always #customers in the input parameter. Some customers have arrival times greater than simulationTime input parameter, so if we had run the simulation till simulationTime, then some customers would not have been able to even arrive
- **The plot is plotted from no of tellers = 2 to 51**
- **The arrival times are kept same in both simulations, but the service times and idle times may be different** since they are calculated every time a teller event occurs
- **Function pointers are used in 6 places, as mentioned above in ‘Function pointers used**’

Interactive / summary mode

- For purpose of debugging, I created two global variables ‘interactive’ and ‘actionSummaries’
- if actionSummaries is set to 1, then all events are printed on the console (arrival, service, idle, etc)
- if interactive is also set to 1, then the teller queues and customer being served are printed at each stage. Press enter at each stage to advance the clock

Results - Performance comparison of both queueing systems

- From the plot, it is observed that the average time spent by a customer decreases (exponentially) as the no of tellers increase

- Comparison of average time spent

S.no.	Test case	Average time spent by customer in bank **(one line per teller)** (seconds)	Average time spent by customer in bank **(common line)** (seconds)
1	./qSim 100 5 30 2.3	760.577942	598.115112
2	./qSim 100 5 60 2.3	195.232239	181.904602
3	./qSim 100 5 80 2.3	171.065643	145.085724
4	./qSim 500 5 60 2.3	5321.106934	5443.620117
5	./qSim 1000 5 60 2.3	12096.048828	12379.695312
6	./qSim 2000 5 60 2.3	26010.552734	25427.103516
7	./qSim 1000 20 60 2.3	2002.885620	1945.021973
8	./qSim 1000 20 60 5	5990.123047	6318.788086
9	./qSim 1000 20 60 0.5	43.597187	36.964069
10	./qSim 1000 10 60 2.3	5526.688965	5582.965332

The average time spent is lower for the common line model in most cases. In some cases where we increase #customers (Case 4 and 5), #tellers (case 10) or averageServiceTime (case 8), average time spent for common line case is more. But on increasing #customers, #tellers or averageServiceTime further, the average time spent again decreases. So we can say there's a small region where average time spent is more for the common line model, but lower in other regions.

- Comparison of maximum waiting time

S.no.	Test case	Maximum waiting time **(one line per teller)** (seconds)	Maximum waiting time **(common line)** (seconds)
1	./qSim 100 5 30 2.3	1313.391235	1005.258057
2	./qSim 100 5 60 2.3	253.993408	146.013184
3	./qSim 100 5 80 2.3	292.757263	88.452637
4	./qSim 500 5 60 2.3	10217.635742	10332.500000
5	./qSim 1000 5 60 2.3	23954.308594	24176.882812
6	./qSim 2000 5 60 2.3	52052.707031	50809.050781
7	./qSim 1000 20 60 2.3	3676.171631	3448.184326
8	./qSim 1000 20 60 5	11439.498047	11799.862305
9	./qSim 1000 20 60 0.5	405.117126	67.031982
10	./qSim 1000 10 60 2.3	10538.971680	10566.932617

The trend for maximum waiting time is similar to that of average time spent in bank, i.e. it is lower for the common line model in most cases. There are some cases where the multi-line model performs better, similar to the previous analysis.

All test cases and their output is mentioned in TestCases.txt