COP290 Assignment 6

Navneel Singhal

March 2020

Assignment description

In this assignment, we were asked to write a program to implement a simulation of a bank teller system, with two different queueing models, as mentioned in the specifications.

Summary of program

We represent events, customers and tellers as separate structures. An event has a function pointer to an action that is performed when that event is encountered. We separate events into 4 types - arrival of a customer, start of service of a customer, departure of a customer, and waking up of a teller. All of these have the same behaviour for both queueing regimes, except for arrivals, where the choice of a teller becomes the differentiating part.
When we encounter an arrival event, we insert a service start event for the corresponding customer after choosing a suitable queue. Note that initially all the tellers are waiting to accept the customers and don't go into periodic sleep. If there are multiple queues, we do the same as the mentioned in the specifications. However, if the model is a single-queue model, we implement it as a single queue with one of the tellers (which is assigned the head teller), and distribute the first few events to the remaining tellers too.
When we encounter a service event, we remove the customer from the queue of the teller, and put the teller in a working state (and push a departure event into the events queue).
When we encounter a departure event or a wake up event, we do the same things except that we update statistics for the customer and the teller in the departure event too. The remaining part is as follows. We check if there is an element in the queue for this teller. If there is, we do nothing (because the service event of that element would have pushed a departure event of that element already, which should be the next event). However, if there is none, we try to find a queue from which we can take out elements. For that, we randomly pick any non-empty queue of a teller (if it exists; if it doesn't, put the teller to sleep), and then take the top element from there, and update the times of all the customers in that queue, and reinsert their service events into the queue. We also insert the service event of the customer we picked into the queue, and the customer into the queue of the teller we are dealing with.

How to run the program

Firstly run the make command to generate the executable.
We support two modes of the program – one of them is as given in specifications, while the other is used for comparing the single queue and multi-queue models at different numbers of tellers.
The program can be run in the first mode by using the same command-line argument format as mentioned in the specification. For running in the second mode, we append the command by the word PLOT (as done in the file **sample.sh**, which is actually used for showing the behaviour of the bank under both queueing regimes).
Finally, if needed, we can run make clean to get rid of the extra files.

Testing, simulation and analysis of results

The testing was carried out on three different customer numbers (50, 200, 1000 – corresponding to small, medium and large respectively), and four pairs of (simulation time, average service time), where simulation time $\in \{60, 10\}$ and average service time $\in \{10, 2\}$. This makes for a total of 12 test cases that the program was run on. The relevant command line arguments are in **sample.sh**, and the plot is named in accordance with the command line arguments used for generating that plot, in the directory samples.
It was observed that for a small number of tellers, on each of these cases, a multiple queue system was a lot more efficient than a single queue system, however, on increasing the number of tellers sufficiently, the times taken by both the models were almost the same.