The University of Texas at Dallas
CS 6352, Performance of Computer Systems and Networks
Fall 2023

Programming Assignment 1
due Thursday, October 26, 2023

In this problem, you will implement an event-driven simulation of a queueing system. In an event-driven simulation, the system state is updated only when an event (e.g., an arrival or a departure) occurs, rather than being updated at periodic time intervals. When an event occurs, several steps must be taken to update the system state. The first step is to update the system time to the time at which the event occurred. The next step is to update any other state parameters, such as the number of customers in the queue. Finally, new events are generated based on the current event. Once the system state is updated, the simulation moves on to the next event in chronological order.

Queueing System Description

Two machines generate components that are sent to a processing center for packaging. The first machine generates components according to a Poisson process with rate γ components/minute. The second machine generates components according to a Poisson process with rate λ components/minute. If there are two or more components in the processing center, the first machine stops generating components. If there are K or more components in the processing center ($K \geq 2$), the second machine continues to generate components, but these components are discarded (they do not enter the queueing system). The processing center employs m workers who package the components. The time it takes each worker to package a component is exponentially distributed with an average packing time of $\frac{1}{\mu}$ minutes.

Event-Driven Simulation

In the simulation of a Markovian queueing system, we need to consider two basic types of events: arrivals and departures. When an arrival event occurs, we need to perform the following tasks:

- Update the system time to reflect the time of the current arrival.
- Increment the number of customers in the system if the system is not at its full capacity and the arrival is not blocked.
- If there is an idle server that can take the arriving customer, then generate a departure event for the new arrival. The departure time will be the current system time plus an exponentially distributed length of time with parameter μ.
- Generate the next arrival event, if applicable. The time of the next arrival will be the current system time plus an exponentially distributed length of time.

When a departure event occurs, we must perform the following steps:

- Update the system time.
- Decrement the number of customers in the system.
- If there are customers waiting in the queue, and if a server is available, then one customer will enter service when the departure occurs. Generate a departure event for this customer entering service.
- If applicable, generate an arrival event. (This step may not be necesary depending on how you implement the simulation).

Once the tasks associated with an event have been completed, the simulation should go on to the next event. In order to manage events, we can maintain an event list. An event list consists of a linked list whose elements are data structures which indicate the type of event and the time at which the event occurs. By sorting the event list in chronological order, the next event can be selected from the head of the event list. When a new event is generated, it is placed in the event list in the correct chronological sequence. Note that the event list is simply a data structure for keeping track of the timing of events in the simulation. The event list does not represent the state of the queueing system.

Collecting Performance Measures

When implementing the simulation, you will also need to maintain additional information in order to calculate performance measures for the system. In particular, you should determine the average number of jobs in the system, the average time a job spends in the system, and the blocking probability versus ρ, where ρ is defined as $\frac{\lambda}{m\mu}$. For each plot, the parameter ρ should range between 0.1 and 1.0, and each plot should contain at least ten data points, (i.e., $\rho = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0$). The values of λ can be determined from the values of μ, ρ, and m, i.e., you will run the simulation for values of $\lambda = 0.1 \cdot m \cdot \mu, 0.2 \cdot m \cdot \mu$, etc. For each data point, you should run the simulation for at least 100000 departures.

Experiments

For the following experiments, *do not* hard-code values for K, m, and μ into your program. Instead, the values should be entered as user inputs to the program. Additional cases may be tested during the grading of your program.

1. Let $\mu = 4$ components/minute and $\gamma = 5$ components/minute. Plot the average number of components in the system versus ρ for the case in which $m = 2$ and $K = 4$. (You may have your program output numerical values, and then create the plots using any standard plotting/graphing/spreadsheet program, such as MS Excel or Matlab.) Include in the same graph plots of the theoretical values of the expected number of customers in the system versus ρ. You may calculate the theoretical values by programming the appropriate equations into a computer program or by calculating the values by hand.
2. Plot the average time spent in the system versus ρ with $\mu = 4$ components/minute, $\gamma = 5$ components/minute, $m = 2$ and $K = 4$. Plot theoretical values for the expected time spent in the system on the same graph.
3. Plot the fraction of components that are discarded (blocked) versus ρ with $\mu = 4$ components/minute, $\gamma = 5$ components/minute, $m = 2$ and $K = 4$. Plot theoretical values for the blocking probability on the same graph.
4. Plot the total utilization of the system (all servers combined) versus ρ with the same parameters as above. Plot theoretical values for utilization on the same graph.

Submission

The assignment is to be submitted through eLearning. To be submitted:

1. Source code files. Be sure to include sufficient comments. **Include your name and netid in a comment at the top of every source code file that you submit.**
2. "Readme" file specifying OS (UNIX, Windows, etc.), compilier/platform, instructions for compiling and running the program.
3. File(s) containing output plots (PostScript, PDF, MS Word, or MS Excel). Be sure to label plots appropriately.

For C++, source code files should end with .cpp or .h extensions. For C, source code files should end with .c or .h extensions. For Java, source code files should end with a .java extension. Place all files in a single-level folder or directory named with your netid, e.g., xyz061000, and zip this directory into a single zip file, e.g., xyz061000.zip. Upload this zipped file to eLearning.

Notes

An example simulation for an M/M/1 system is available at:

http://www.utdallas.edu/%7Ejjue/cs6352/sim/

You may use this code as a template for your simulation, or you may write your own code. **Under no circumstances may you use or view code from any other source. Also, do not show any part of your code to other students.** Programs will be checked using copy-detection software. If your code is found to be similar to another person's code, you will be subject to disciplinary action according to University policies. If you are unable to complete your project on time, submit whatever you have done. Partial credit will be given.