The University of Texas at Dallas
CS/CE 6352, Performance of Computer Systems and Networks
Spring 2014

Programming Assignment 1
due Wednesday, March 26, 2014

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

Consider a computer system consisting of m processors and with a total capacity of K jobs (including those jobs being served by the processors). Assume $K \geq m$. Administrative job tasks arrive to the system according to a Poisson process with rate λ_1 jobs/second. If the system is at its full capacity of K jobs, arriving administrative job tasks are blocked from entering the system. User job tasks arrive to the system according to a Poisson process with rate λ_2 jobs/second. In order to maintain space in the system for future administrative job tasks, a threshold limit of l is set for user job tasks. If the total number of jobs in the system is greater than or equal to l, then arriving user job tasks will be blocked from entering the system. Assume that $0 \leq l \leq K$. Once jobs enter the system, they are all treated equally, and each job requires a processing time that is exponentially distributed with an average processing time of $\frac{1}{\mu}$ seconds.

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 system size if the system is not at its full capacity and the arrival is not blocked.
- If there is an idle server, 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 system size.
- 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 the 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 $\frac{\lambda_1}{m\mu}$. 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_1 can be determined from the values of μ, ρ, and m, i.e., you will run the simulation for values of $\lambda_1 = 0.1 \cdot m \cdot \mu, 0.2 \cdot m \cdot \mu$, 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, l, λ_2, 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 $\mu = 3$ jobs/second and $\lambda_2 = 4$ jobs/second. Plot the average number of jobs in the system versus ρ for the case in which $m = 2$, $K = 4$, and $l = 1$. (You may have your program output numerical values, and then create the plots using any standard plotting/graphing/spreadsheet program.) Include in the same graph plots of the theoretical values of the expected number of jobs in the system versus ρ. You may calculate the theoretical values either by hand or by programming the appropriate equations into a computer program.
2. In the same graph above, plot the average number of jobs in the system versus ρ with $l = 3$. The remaining parameters should remain the same.
3. Plot the simulation and theoretical values for the expected time that a job spends in the system versus ρ for each of the cases above ($l = 1$ and $l = 3$).
4. Plot the simulation and theoretical values for the probability that an administrative job is blocked versus ρ for each of the cases above ($l = 1$ and $l = 3$).
5. Plot the simulation and theoretical values for the probability that a user job is blocked versus ρ for each of the cases above ($l = 1$ and $l = 3$).

Submission

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

1. Source code files. Be sure to include sufficient comments.
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.