Assignment 2

Computer Science 531

Due: 11:55 PM, Friday October 30, 2015

Instructor: Majid Ghaderi

Overview

The purpose of this assignment is to implement the basic components of a discrete-event simulation program. While this assignment specifically considers a single server queue, you should design various components of your program, such as the event list and random variate generators, in such a way that they can be used to simulate more complicated systems in the future assignments.

1 System Description

A popular food truck called "Polar Burger" has only one chef who is in charge of both cooking the burgers and making sandwiches. Customers wanting to order a sandwich must wait in a line right beside the food truck that stretches along the street. Customers are served following the first-come first-served order. Due to the popularity of Polar Burger, the wait time for getting a sandwich can be long. The weather in the City of Calgary, where the food truck operates, is very cold. To avoid getting frostbites, if a customer cannot place his/her order within a specific time called the customer's *patience time*, the customer will leave the line to go somewhere else for food. Different customers have different patience times.

The owner of the food truck is considering hiring another person to help the chef making sandwiches faster. However, before making a decision, he would like to carry out a study to find out what percentage of customers are actually lost due to long wait times. If the percentage is small then it may not be worth hiring another person.

Of particular interest are the following performance parameters:

1. The *mean number of customers in the system* denoted by N. The number of customers in the system includes the customers waiting in line and the customer who is getting service.
2. The *mean wait time of patient customers* denoted by W_p. Patient customers are those who leave the system after getting service.
3. The *mean wait time of impatient customers* denoted by W_i. Impatient customers are those who leave the system before getting service.
4. The *percentage of impatient customers* denoted by K. The percentage of impatient customers is given by the number of customers who leave the system before getting service (due to impatience) divided by the total number of customers who enter the system.
5. The *utilization of the chef* denoted by U. The utilization is given by the percentage of time that the chef is busy.

2 Modeling Assumptions

We assume that the patience time of a customer is known at the time the customer arrives at the system. When the chef starts serving a customer, the customer waits until he/she gets a sandwich before leaving the system regardless of the customer's patience time.

For simulation purposes, we further assume that:

- The customer arrivals follow a Poisson process with the rate λ customers per minute.
- The service times of customers are IID and follow an exponential distribution with the mean of $1/\mu$ minutes.
- The patience times of customers are IID and follow an exponential distribution with the mean of $1/\gamma$ minutes.
- At time 0, the system is empty and the first customer arrives after an interarrival time.

3 Software Interfaces

Skeleton code for a Java class called `PolarBurger` is provided to you. You are asked to write code to complete the implementation of this class. You can define other variables, methods and classes if required. The following public methods in this class need to be implemented:

- `PolarBurger(double lambda, double mu, double gamma)`
  This is the constructor to initialize the simulation. The parameters `lambda`, `mu` and `gamma` are as defined in Section 2.
- `void run(int time, int seed)`
  Starts the simulation. The simulation runs for the specified length of time given by the parameter `time` (in minutes). The parameter `seed` specifies the seed for your random number generator. You should put the code for simulation initialization and main loop in this method. It should be possible to run the simulation for different parameters `time` and `seed` by repeatedly calling this method.
- `Report getReport()`
  Generates and returns a report on the performance measures of interest as described in Section 1. This method returns an object of type `Report`. The source code for the class `Report` is provided to you on the assignment page on D2L.

Your implementation should handle all possible exceptions/errors by catching them and printing appropriate messages to the standard output. A simple tester class, named `Tester`, is also provided so that you can see how we are going to test your code.