Group Project 2

In this problem, you will implement an event-driven simulation of a queueing system to analyze the performance of the computer architecture. In an event-driven simulation, the ALU or CPU status is updated only when an event (e.g., an arrival or a departure of programs) occurs, rather than being updated at periodic time intervals. When an event occurs, several steps must be taken to update the 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 programs in the queue. Finally, new events are generated based on the current event. Once the computer system state is updated, the simulation moves on to the next event in chronological order.

<Simulation 1>

Consider a computer consisting of m processors (CPUs) and with a total capacity of K programs (including those jobs being served by the processors). Note that the capacity is partially determined by the memory of the computer. Assume $K \geq m$. Programs arrive to the computer according to a Poisson process with rate λ jobs/second. If the computer is at its full capacity of K programs, arriving programs are blocked from entering the computer. Each program 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 of the program: 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 processor and that processor is in operation, then generate a departure event for a 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. 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 programs waiting in the queue (memory), and if a server is available and in operation, then one program will enter service when the departure occurs. Generate a departure event for the program entering processor.

Once the event has 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.

<Collecting Performance Measures>

When implementing the simulation, you will also need to maintain additional information in order to calculate performance measures for the computer. In particular, you should determine the average number of programs in the computer, the average time a program 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., ρ = 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 λ = 0.1 ·m· μ, 0.2 ·m· μ, etc. For each data point, you should run the simulation for at least 200000 departures.

< Experiments>

1. Let μ = 3 jobs/second. Plot the average number of programs in the computer 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.)
2. Plot the simulation values for the expected time that a program spends in the system versus ρ for each of the cases above.
3. Plot the simulation values for the probability that a program is blocked versus ρ for each of the cases above.

<Notes>

An example simulation for an M/M/1 queueing system, used to simulate the computer with single CPU and infinite memory capacity, is available in the MM1.zip. 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.

<Simulation 2>

In our computer system, we need to consider two basic types of programs: system programs and user programs. Consider a computer consisting of *m* processors (CPUs) and with a total capacity of *K* programs (including those jobs being served by the processors). Note that the capacity is partially determined by the memory of the computer. Assume $K \geq m$. System programs arrive to the computer according to a Poisson process with rate λ_1 jobs/second. If the computer is at its full capacity of *K* programs, arriving system programs are blocked from entering the computer. User programs 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 system programs, a threshold limit of l is set for user programs. If the total number of programs in the system is greater than or equal to l, then arriving user programs 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 program requires a processing time that is exponentially distributed with an average processing time of $\frac{1}{\mu}$ seconds.

The event-driven simulation is the same as the single type with only minor change: when you generate the new arrival event, you should generate the program type, which is the same as the current event. Do not forget to modify the event list to include the program types (system or user).

<Collecting Performance Measures>

When implementing the simulation, you will also need to maintain additional information in order to calculate performance measures for the computer. In particular, you should determine the average number of programs in the computer, the average time a program 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., ρ = 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 λ_1 = 0.1 ·m· μ, 0.2 ·m· μ, etc. For each data point, you should run the simulation for at least 200000 departures.

<Experiments>

1. Let μ = 3 jobs/second and $\lambda_2 = 4$ jobs/second. Plot the average number of programs in the computer 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.)
2. In the same graph above, plot the average number of programs in the computer versus ρ with $l = 3$. The remaining parameters should remain the same.
3. Plot the simulation values for the expected time that a program spends in the system versus ρ for each of the cases above ($l = 1$ and $l = 3$).
4. Plot the simulation values for the probability that a program is blocked versus ρ for each of the cases above ($l = 1$ and $l = 3$).