Event-driven Simulation of a Link

Assignment Description

In this assignment, you will write an event-driven simulator to simulate a simple link. Packets arrive at the link at various times. If the link is already transmitting a packet while another packet arrives, the new packet is stored in the link's buffer. Henceforth, the transmission time for a packet in the link will be referred to as the 'service time' of the packet.

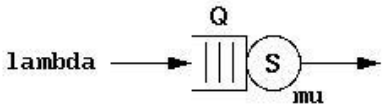

Figure 1: A Simple Link.

There is a single stream of packets arriving to the above link with rate lambda. Packets are queued at the buffer (Q) and are served in first-in-first-out (i.e., first-come-first-served) order. As the link S becomes idle, it removes the packet at the head of the queue and begins transmitting the packet. The packets are served at a rate mu. The unit for both lambda and mu is (1/sec).

Our system can run in only one of three modes.

Deterministic : In this mode, all inter-arrival times are equal to exactly 1/lambda seconds and all service times are equal to exactly 1/mu seconds. (This mode is mainly for debugging purposes.)

Exponential : In this mode, inter-arrival times are exponentially distributed with a mean of 1/lambda seconds and service times are exponentially distributed with a mean of 1/mu seconds.

Trace-driven : In this mode, we will drive the simulation using a trace specification file. Each line in the trace file specifies the inter-arrival time (in milliseconds) of a packet and its service time (in milliseconds).

Note that unless otherwise specified, the queue size (or buffer size) at the server is assumed to be infinite. Clearly, for infinite queue sizes, there are no packet drops. Simulation of this system is done using an event-driven simulation.

Compiling

Please use a Makefile so that when the grader enters:

make

an executable named mm1 is created.

Commandline

The command line syntax (also known as "usage information") for mm1 is as follows:

mm1 [-lambda lambda] [-mu mu] [-det num] [-exp num] [-s seed] [-t tsfile]

Square bracketed items are optional. You must follow the UNIX convention that commandline options can come in any order. (Note: a commandline option is a commandline argument that begins with a - character in a commandline syntax specification.) Unless otherwise specified, output of your program must go to stdout and error messages must go to stderr.

The lambda and mu parameters have obvious meanings (according to the description above).

The -det option specifies that you should run the simulation in the deterministic mode and the total number of packets to arrive is num.

The -ext option specifies that you should run the simulation in the exponential mode and the total number of packets to arrive is num. In this case, you may provide a seed for initializing a random number generator using the -s option.

The -t option specifies that you should run the simulation in the trace-driven mode and tsfile specifies a trace specification file.

The default simulation mode is the deterministic mode. This means that if none of -det, -ext, or -t options are specified, you must run the simulation in the deterministic mode. (In this case, please ignore the -s commandline options if it is given by the user.) Furthermore, specifying more than one of the -det, -ext, or -t options is not allowed. If the -t options is specified, please ignore the -lambda, -mu, and -s commandline options if they are specified.

The default value (i.e., if it's not specified in a commandline option) for lambda is 1.0 (packet per second). The default value for mu is 0.7 (packets per second).
The default value for num is 20 (packets).
The default value for seed is 0.
The smallest possible value for num is 1 and the largest possible value for num or size is 2147483647 (or 0x7fffffff in hexadecimal). Finally, lambda and mu must be positive real numbers.

Running Your Code and Program Output

The simulation should go as follows. At simulation time 0, the first packet should be scheduled to arrive at the time it is suppose to arrive (i.e., either according to the value of lambda or line 2 in a trace specification file, depending on the simulation mode).

As a packet arrives, or as the server becomes free, you need to follow the operational rules of the link.

You are required to produce a detailed trace for every important event occurred during the simulation. Each line in the trace must correspond to one of the following situations (we use packet #1 as an example):

- For each packet, there must be exactly 3 output lines that correspond to this packet. They are (# is a packet number and made-up numeric values are used):

  p# arrives and enters Q, inter-arrival time = 0.503s
  p# leaves Q and begins service at S, time in Q = 0.247s, requesting 2.850s of service p# departs from S, service time = 2.859s, time in system = 3.109s

Please note that the value printed for "inter-arrival time" must equal to the timestamp of the "p# arrives" event minus the timestamp of the "arrives" event for the previous packet. The value printed for "time in Q" must equal to the timestamp of the "p# leaves Q" event minus the timestamp of the "p# enters Q" event. (If the server is not busy when the packet arrives, then "time in Q" must be equal to 0.000s.) The value printed for "service time" must equal to the timestamp of the "p# departs from S" event minus the timestamp of the "p# begins service at S" event (and it should equal to the service time of this packet). The value printed for "time in system" must equal to the timestamp of the "p# departs from S" event minus the timestamp of the "p# arrives" event;

- When you are ready to start your simulation, you must print:

  simulation begins

- When you are ready to end your simulation, you must print:

  simulation ends

All the numeric values above are made up. You must replace them with the actual packet number, actual inter-arrival time, measured time spent in Q, measured service time, and measured time in the system.

Below is an example what your program output must look like (please note that the values used here are just a bunch of unrelated random numbers for illustration purposes):

```
Simulation Parameters:
    number to arrive = 20
    lambda = 2            (<== print this line only if -t is not specified)
    mu = 0.35             (<== print this line only if -t is not specified)
    size = 5              (<== print this line only if -z is specified)
    mode = det/exp        (<== print this line only if -det or -exp is specified)
    seed = 1              (<== print this line only if -exp is specified)
    tsfile = FILENAME     (<== print this line only if -t is specified)

00000000.000s: simulation  begins
00000003.112s:  p1 arrives and enters Q, inter-arrival time = 3.112s,
                p1 leaves Q and begins service at S, time in Q = 0.000s, requesting 2.850s of service
00000004.271s:  p2 arrives and enters Q, inter-arrival time = 1.159s
00000004.986s:  p3 arrives and enters Q, inter-arrival time = 0.715s
00000005.962s:  p1 departs from S, service time = 2.850s, time in system = 2.850s
                p2 leaves Q and begins service at S, time in Q = 1.691s, requesting 2.134s of service
00000006.753s:  p4 arrives and enters Q, inter-arrival time = 0.789s
00000008.096s:  p2 departs from S, service time = 2.134s, time in system = 3.825s
                p3 leaves Q and begins service at S, time in Q = 3.110s, requesting 1.894s of service
...
????????.???s:  p20 departs from S, service time = ???.???s, time in system = ???.???s
                simulation  ends




Statistics:

    average packet inter-arrival time = <real-value>
    average packet service time = <real-value>

    average number of packets in Q = <real-value>
    average number of packets at S = <real-value>

    average time a packet spent in system = <real-value> standard
    deviation for time spent in system = <real-value>
```

The first column contains timestamps and they correspond to event times, measured relative to the start of the simulation. Please use 8 digits (with leading zeroes) to the left of the decimal point and 3 digits after the decimal point for all the timestamps in this column. All time intervals must be printed in seconds with 3 digits after the decimal point. In the printout, after simulation parameters, all values reported must be measured values.

The average number of packets at a facility can be obtained by adding up all the time spent at that facility (for all packets) divided by the total simulation time. The time spent in system for a packet is the difference between the time the packet departed from the server and the time that packet arrived.

All real values in the statistics must be printed with at least 6 significant digits. (If you are using printf(), you can use "%.6g".) A timestamp in the beginning of a line of trace output must be in milliseconds with 8 digits (zero- padded) before the decimal point and 3 digits (zero-padded) after the decimal point.

Please use sample means when you calculated the averages. If *n* is the number of sample, this mean that you should divide things by *n* (and not *n-1*).

The unit for time related *statistics* must be in seconds (and not milliseconds).

Let X be something you measure. The standard deviation of X is the square root of the variance of X. The variance of X is the average of the square of X minus the square of the average of X. Let E(X) denote the average of X, you can write:

Var(X) = E(X2) - [E(X)]2

Finally, when Q is empty and no future packet can arrive into it, you must end the simulation.

Event-driven Simulation

In an event-driven simulator, the simulator maintains a virtual clock (i.e., not a wall clock or a real-time clock) and an event queue. The time on the virtual clock is the "simulation time" and it is zero when the simulate starts. The event queue is a queue of timestamped events, sorted according to the event timestamps in ascending order (i.e., an event with the smallest timestamp is at the head of the queue). Let q denote the event queue and clk denote the virtual clock value, the pseudo-code of the simulator is depicted below:

```
while (!q->empty) do:
    e = q->dequeue clk
    = e->timestamp
    e->process()
end-while
```

The last step within the while-loop, i.e., "e->process()", refers to whatever code you need to execute to process the event e. When you process an event, you may end up adding new events to the event queue. Please note that event processing does not take up any simulation time, no matter how long it takes in real time process the event.

Specific Event Types and Processing for This Assignment

For this assignment, there are only two types of events, an packet arrival event and a packet departure event. Here are the processing that needs to be done for these events:

Event	Event Processing
packet arrival	1) append packet to Q 2) if server S is not busy, schedule a departure event for the first packet in Q 3) if there are more packets to arrive, schedule the next packet arrival event
packet departure	1) if Q not empty, schedule a departure event for the first packet in Q

Random Numbers Generation

In the exponential mode, the inter-arrival times and service times are distributed according to exponential distributions. Please use the ExponentialInterval(rate) function below to generate an exponentially distributed random interval with mean 1/rate (where rate is either lambda or mu):

```
#include <time.h>
#include <stdlib.h>
#include <math.h>

#define round(X) (((X)>= 0)?(int)((X)+0.5):(int)((X)-0.5))

void InitRandom(long l_seed)
     /*
      *initialize the random number generator
      *if l_seed is 0, will seed the random number generator using the current clock
      *if l_seed is NOT 0, your simulation should be repeatable if you use the same l_seed
      *call this function at the beginning of main() and only once
     */
{
    if (l_seed == 0L) {
        struct timespec ts;
        clock_gettime(CLOCK_REALTIME, &ts);

        srand48(ts.tv_nsec);
    } else {
        srand48(l_seed);
    }
}

int ExponentialInterval(double rate)
     /*
      *returns an exponentially distributed random interval (in milliseconds) with mean 1/rate
      *rate is either lambda or mu (in packets/second)
      *according to the spec: w = ln(1-r) / (-rate)
     */
{
    double dval;
    do {
        dval = drand48();
        /* if dval is too small or too large, try again */
    } while (dval < 0.000001 || dval > 0.999999);

    dval = ((double)1.0) - dval;

    dval = -log(dval);

    dval = ((dval / rate) * ((double)1000));

    return round(dval);
}
```

In order for everyone's program to generate the same packet inter-arrival times and service times, you are required to do the following. Basically, to obtain the inter-arrival time of packet cn, it must be the (2n-1)th time you call ExponentialInterval(). To obtain the service time of packet cn, it must be the (2n)th time you call ExponentialInterval(). For example:

- The 1st time you call ExponentialInterval(), you must pass lambda as the argument and the returned value must be interpreted as the inter-arrival time of packet p1.
- The 2nd time you call ExponentialInterval(), you must pass mu as the argument and the returned value must be interpreted as the service time of packet p1.
- The 3rd time you call ExponentialInterval(), you must pass lambda as the argument and the returned value must be interpreted as the inter-arrival time of packet p2.
- The 4th time you call ExponentialInterval(), you must pass mu as the argument and the
- returned value must be interpreted as the service time of packet p2.
  and so on

Trace Specification File Format

The trace specification file is an ASCII file containing n+1 lines (each line is terminated with a "\n") where n is the total number of packets to arrive. Line 1 of the file contains a positive integer which corresponds to the value of n. Line k of the file contains the inter-arrival time in milliseconds (a positive integer) and service time in milliseconds (a positive integer) for packet k-1. The two fields are separated by space or tab characters (or any combination of any number of these characters). There must be no leading or trailing space or tab characters in a line. A sample tsfile for n=3 packets is provided. It's content is listed below:

```
3
2716  16253
7721  35149
972   2614
```

In the above example, packet 1 arrives 2716ms after simulation starts and its service time is 9253ms; the inter-arrival time between packet 2 and 1 is 7721ms and its service time is 35149ms; the inter-arrival time between packet 3 and 2 is 972ms and its service time is 2614ms.

This file is expected to be error-free. (This means that if you detect a real error in this file, you must simply print an error message and call exit(). There is no need to print statistics or perform error recovery.)

Sample Output

If the user enters:
./mm1 -t tsfile.txt
where tsfile.txt is the sample given above, other than the numerical values in the "Statistics" section, your output must look exactly like the following:

```
Simulation Parameters:
    number to arrive = 3
    tsfile = tsfile.txt

00000000.000s: simulation  begins
 00000002.716s: p1 arrives  and enters Q, inter-arrival time = 2.716s,
                p1 leaves Q and begins service at  S, time in Q = 0.000s, requesting 16.253s of service
 00000010.437s: p2 arrives and enters Q, inter-arrival time = 7.721s
 00000011.409s: p3 arrives and enters Q, inter-arrival time = 0.972s
 00000018.969s: p1 departs from S, service time =  16.253s, time in system = 16.253s
                p2 leaves Q and begins service at  S, time in Q = 8.532s, requesting 35.149s of service
 00000054.118s: p2 departs from S, service time =  35.149s, time in system = 43.681s
                p3 leaves Q and begins service at  S, time in Q = 42.709s, requesting 2.614s of service
 00000056.732s: p3 departs from S, service time =  2.614s, time in system = 45.323s
               simulation ends

Statistics:

    average packet inter-arrival time = 3.803s
    average packet service time = 18.005333s


    average number of packets in Q = 0.946838
    average number of packets at S = .952126

    average time a packet spent in system = 35.991000s
    standard deviation for time spent in system = 12.055024
```