Comprehensive Final Exam for Simulation (CAP 4800) Summer 2013

>>> SOLUTIONS <<<

Welcome to the comprehensive Final Exam for *Simulation*. Read each problem carefully. There are 10 required problems where each problem is worth 10 points. There is also an additional extra credit problem worth 10 points. You may have with you a calculator, pencils and/or pens, erasers, blank paper, and one 8.5 x 11 inch "formula sheet". On this formula sheet you may have anything you want (definitions, formulas, homework answers, old exam answers, etc.) as **handwritten by you in pencil or ink** on both sides of the sheet. Photocopies, scans, or computer generated and/or printed text are not allowed on this sheet. Note to tablet (iPad, etc.) users – you may **not** print-out your handwritten text for the formula sheet. Please answer the problems on the available sheets. If you need extra space, use the back side of the page of the problem that you are answering. You have 120 minutes for this exam.

Problem #1

Each sub-problem is worth 2 points.

Answer the following questions regarding the basics of modeling.

a) What is a model? Give a short formal definition.

"A model is a representation (physical, logical, or functional) that mimics another object under study." (Molloy 1989).

b) Why do we build models and not just experiment on actual systems?

Models are often cheaper, easier, faster, and/or safer to build and experiment on than the actual system.

c) What is performance? Give a short formal definition.

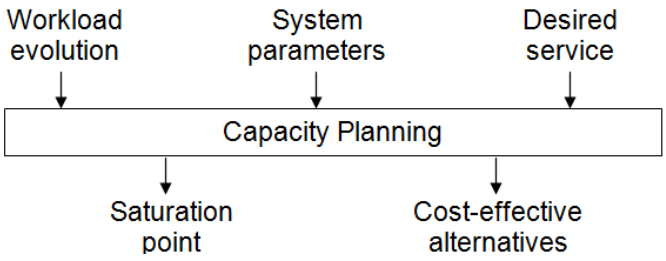
Performance is the quantitative measure of a system.

d) What is a bottleneck? Give a short formal definition.

A bottleneck is the component that limits the performance of a system.

e) What is capacity planning? Discuss the relationship of capacity planning to SLA. **Hint:** A figure may be helpful.

Capacity planning is a process whereby the capacity of a system (processing, network, storage, etc.) is determine to meet a given performance need. The "Saturation point" is the SLA limit for some performance metric (say, response time).



Problem #2 Sub-problems (a) thru (c) worth 2 points each, (d) worth 4 points.

Answer the following questions regarding performance and design of experiments.

a) If system A can perform 100 operations per second and system B can perform 250 operations per second, what is the speed-up of system B relative to system A? What is the speed-up of system A relative to system B?

The speed-up of B relative to A is 250/100 = 2.5. The speed-up of A relative B is 100/250 = 0.40.

b) In the context of experiments, define response variable, factor, factor level, and replication.

Response variable = performance measure of interest (the outcome of the experiment)
Factor = variable that affect the response variable
Factor level = values that a factor can take on
Replication = the repetition or trial of an experiment

c) The following shows the response value for two factors A and B each with two factor values (A1, A2, B1, and B2). Based on the results in the below, do factors A and B interact? Explain why or why not?

	A1	A2
B1	10	15
B2	22	22

It appears that factors A and B do indeed interact. When factor B is at level 1 then when factor A changes from level 1 to level 2 the response increases by 5. However, when factor B is at level 2 the change in factor A level has no effect on the response value, which is different from when B is at level 1 (and hence factor B interacts with factor A).

d) Consider a system with three factors, A, B, and C. Assume that A has 2 levels, B has 2 levels, and C has 3 levels. Design (that is, show the factor levels) a full-factorial and simple experiment for this system.

Full factorial: (for 2x2x3 = 12 experiments)

A1-B1-C1
A1-B1-C2
A1-B1-C3
A1-B2-C1
A1-B2-C2
A1-B2-C3
A2-B1-C1
A2-B1-C2
A2-B1-C3
A2-B2-C1
A2-B2-C2
A2-B2-C3

Simple: (for 1+1+1+2 = 5 experiments)

A1-B1-C1
A1-B1-C2
A1-B1-C3
A1-B2-C1
A2-B2-C1 (Note: Other sets are possible – each factor must take on all of its levels)

Problem #3 Sub-problems (a) and (b) worth 3 points each, (c) worth 4 points.

Answer the following questions regarding probability theory and generating random numbers.

a) Assume the requests arrive to a web server as a Poisson process with λ = 5. What is the mean time between requests? What is the standard deviation of the time between requests? The pdf for the Poisson process is:

$$f(k) = \frac{(\lambda t)^k}{k!} e^{-\lambda t} \qquad k = 0, 1, 2, \ldots \text{ and } t \geq 0$$

The time between events in a Poisson process is exponentially distributed. For the exponential distribution we know that $1/\lambda = 0.2$ seconds is the mean time between events and that for this distribution only, the mean and standard deviation are the same. Thus, the standard deviation is 0.2 seconds as well.

b) Assume that the interarrival times of requests to a web server follow an exponential distribution and assume that the rate of arrivals is 3 requests per second. What is the probability that the next request will arrive between 1 and 2 seconds from the time of arrival of the previous request? Show your work. The pdf and CDF for an exponential distribution are $f(t) = \lambda e^{-\lambda t}$ for $t \geq 0$ and $F(t) = 1 - e^{-\lambda t}$ for $t \geq 0$.

$$Pr[\text{next arrival between 1 and 3 seconds}] = F(2) - F(1) = \left(1 - e^{-3 \cdot 2}\right) - \left(1 - e^{-3 \cdot 1}\right) = 0.047$$

c) Consider a continuous uniform distribution with the following pdf:

$$f(x) = \begin{cases} \frac{1}{3} & 2 \leq x \leq 5 \\ 0 & \text{otherwise} \end{cases}$$

Write a C function that returns a random value distributed as $f(x)$.

First we convert to F(x) so that we can invert…

$$F(x) = \int_0^x f(y)dy = \int_0^x \frac{1}{3} dy = \frac{y}{3}\Big|_2^x = \frac{x}{3} - \frac{2}{3} = \frac{x-2}{3}$$

Then we can invert F(x) as follows…

$$U = \frac{x-2}{3}$$

$$x = 2 + 3 \cdot U$$

The C function is then:

```
double uniform2to5()
{
  double z;          // Unif(0, 1) random value

  // Pull a unif(0, 1) random value
  z = randUnif();

  // Compute random value using inversion
  value = 2 + 3*z;

  return(value);
}
```

Problem #4 Each sub-problem is worth 2 pts.

Answer the following questions about queueing.

a) What is Kendall notation. Describe it.

A queue is described as A/S/c/k/m where A is the arrival distribution, S is the service distribution, c is the number of servers, k is the capacity of the system in number of customers, and m is the number of customers in the universe. A and S can be "M" for Markov, "D" for deterministic, "G" for general, and others.

b) Given a queueing system with an arrival rate of 3 jobs per minute and a measured mean wait (response time) of 10 minutes, what is the mean number of jobs in the system?

This can be solved using Little's Law, $L = \lambda W$ for L is the mean number in the system, λ is the arrival rate, and W is the mean wait in the system. So, 3 x 10 = 30 jobs in the system.

c) Given an M/M/1 queue with an arrival rate of 10 jobs per second and a service rate of 5 jobs per second what is the mean number of customers in the system? What is the mean customer delay?

Pay attention here! The arrival rate is greater than the service rate so the system is unstable (the queue grows without bound). The mean number of customers in the system is infinity and so is the mean customer delay.

d) What does the P-K (Pollaczek–Khinchine) formula solve for? What are the inputs to the formula?

It solves for the mean number of customers (or mean wait) of an M/G/1 queue where the arrival rate, service rate, and second moment of the service time distribution are known.

e) What is the difference between the Erlang-B and Erlang-C equations? In other words, what does Erlang-B solve for compared to Erlang-C?

Erlang-B solves for a system with M servers and no buffer – it solves for the probability that an arriving call (or job) is blocked (i.e., all servers are busy). Erlang-C solves for a system with M servers and an infinite buffer – it solves for the probability that an arriving call (or job) is queued (again, all servers are busy).

Problem #5 Each bug identified and fixed is 1 point

In the appendix to this exam is our `mm1.c` simulation program for modeling an M/M/1 queue with some bugs. The bugs are all major in that they will affect the execution and output (but, the program still compiles). Identify the bugs and fix them. You can do your work in the appendix.

See appendix for solution.

Problem #6 Each sub-problem is worth 2 pts.

Answer the following questions about CSIM.

a) What is a CSIM process? What are process states?

A CSIM process is a C function that runs independently (analogous to a thread) of other CSIM processes. A process can be used to model elements of the workload, clients and servers, or any other active components of the system. Possible states are executing, waiting to execute, and holding.

b) How does time progress in a CSIM simulation?

With a hold() statement

c) Name the key CSIM objects. Describe in one sentence what a CSIM Facility is.

Key objects are Facility, Storage, Event, Mailbox, Table and QTable, Process class, and streams of random numbers. A CSIM Facility is used to model resources which are seized (used) by processes

d) Insert code in the below process to record response time data into a table named `DelayTable.`

```
void queue1(double service_time)
{
  double org_time;

  create("queue1");

  org_time = clock;

  // Reserve, hold, and release server
  reserve(Server);
  hold(service_time);
  release(Server);

  record((clock – org_time), DelayTable);
}
```

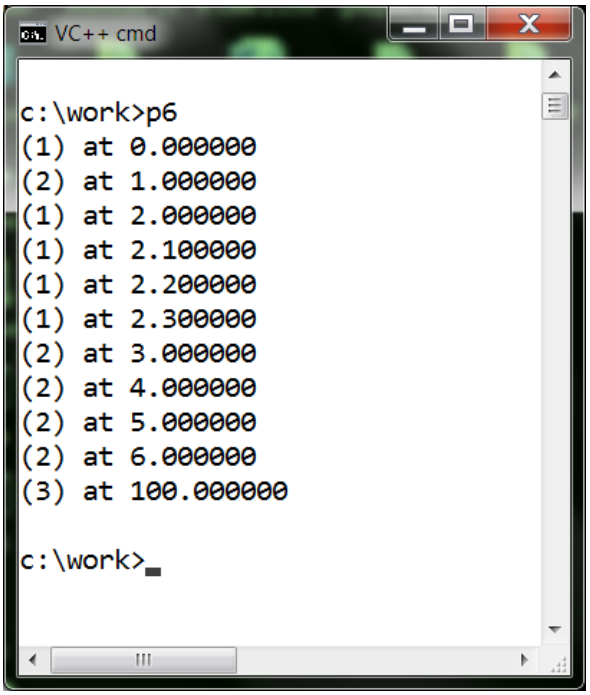
e) Write the code to print the maximum, minimum, and average response time for response time delays recorded in `DelayTable` for the above code snippet.

```
printf(“Maximum delay = %f \n”, table_max(DelayTable));
printf(“Minimum delay = %f \n”, table_min(DelayTable));
printf(“Average delay = %f \n”, table_mean(DelayTable));
```

Problem #7 Each line of output is worth 1 pt.

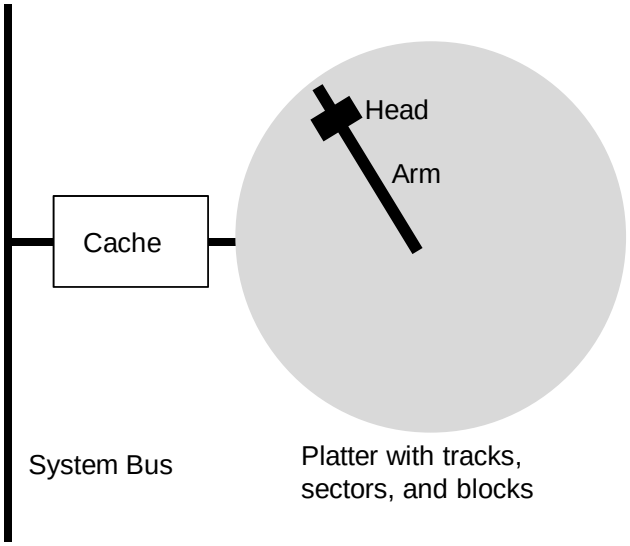
What is the output from the following CSIM model?

```
#include <stdio.h>
#include "csim.h"

FACILITY Server;

void generate(void);
void queue1(void);

void sim(void)
{
  create("sim");
  Server = facility("Server");
  generate();
  hold(100.0);
  printf("(3) at %f \n", clock);
}

void generate()
{
  create("generate");
  queue1();
  hold(2.0);
  queue1();
  hold(0.1);
  queue1();
  hold(0.1);
  queue1();
  hold(0.1);
  queue1();
}

void queue1()
{
  create("queue1");
  printf("(1) at %f \n", clock);
  reserve(Server);
  hold(1.0);
  release(Server);
  printf("(2) at %f \n", clock);
}
```



Problem #8 The description is 5 pts, the code is 5pts.

In class (and in the posted handouts – the phase 1 model) we studied how to model the performance of a disk drive. We considered a "phase 0", "phase 1", and "phase 2" model. Describe the phase 1 model. Write the CSIM code for the function (process) that models the disk drive and measures its response time. **Hint:** You may wish to start with a diagram and description of the system.

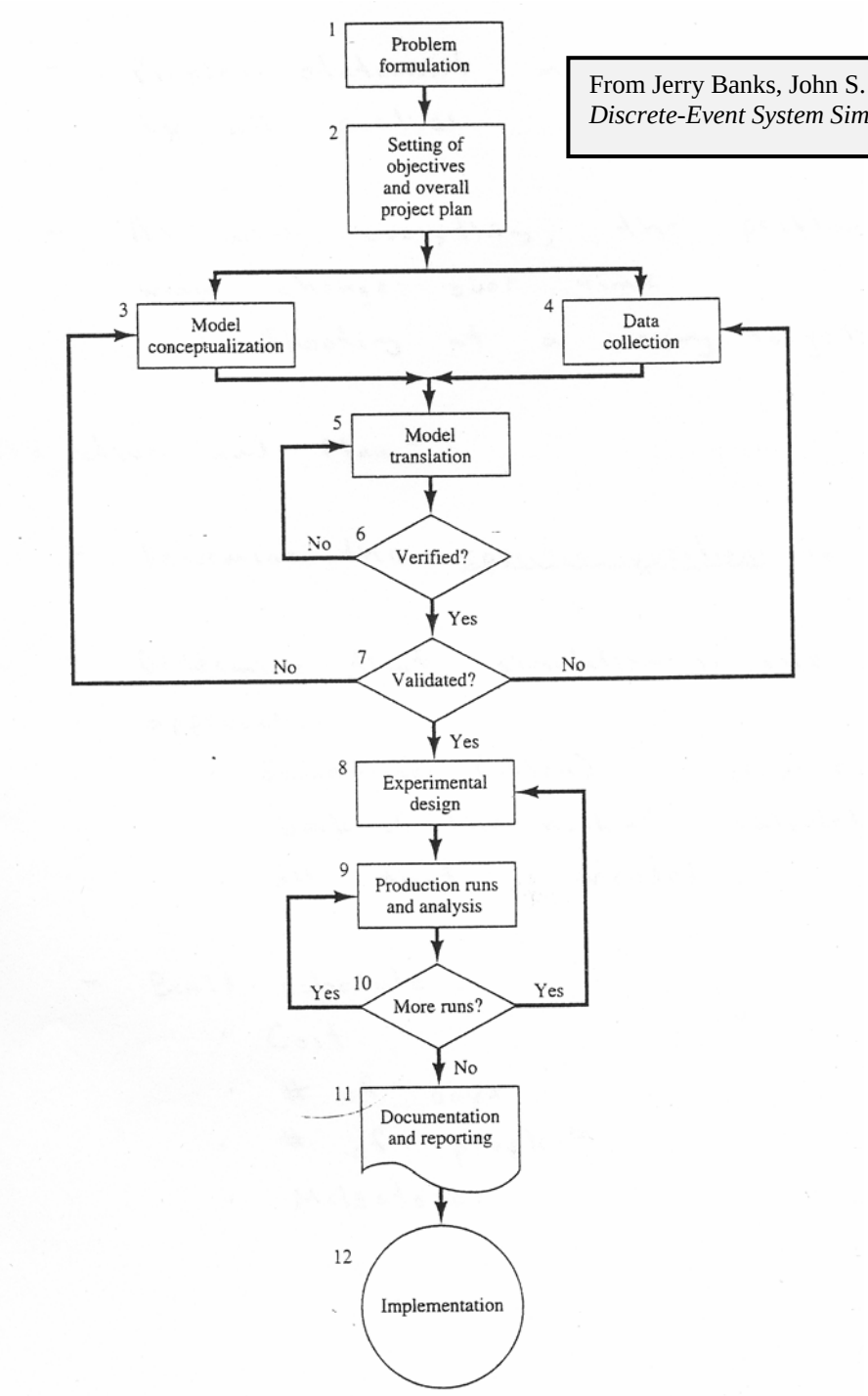


For a file read the head must move to the track in which the file is stored. The head then reads the file (assumed here to be in one sector in one track) when the platter has spun to the where it can be found. If the file is stored in the cache, then a cache hit will occur and the file is read directly from the cache. Cache read time is very fast, head move time is very slow (10s of ms), disk spin time is slow (1s of ms), and disk read rate is fast (but, slower than cache read).

```
//================================================================================
//==  Process to process read requests                                          ==
//================================================================================
void disk(double file_size, double org_time)
{
  create("disk");

  // Increment the total number of read requests
  Total_req++;

  // Process the file read request
  reserve(Disk_facility);
  if (uniform(0.0, 1.0) < Prob_hit)
  {
    Cache_hit++;                      // Increment the cache hit counter
    hold(file_size / Read_rate1);     // Read the file bytes from cache
  }
  else
  {
    hold(uniform(0.0, Seek_time));     // Move the head
    hold(uniform(0.0, Spin_time));     // Spin the disk to position
    hold(file_size / Read_rate2);      // Read the file bytes from disk
  }
  release(Disk_facility);

  // Record the response time
  record((clock - org_time), Resp_table);
}
```

Problem #9 Each box is is worth 1 pt.

Sketch the flowchart for the steps in a simulation study as discussed in class (as described by Banks (1996)).



From Jerry Banks, John S. Carson, and Barry L. Nelson, *Discrete-Event System Simulation*, Prentice-Hall, 1996.

Problem #10 Formulas are worth 6 pts, calculations and summary are worth 4 pts

You have simulated two systems for 7 replications each. The sample means for response times from system #1 are 110, 105 110, 108, 106, and 112 milliseconds, and from system #2 they are 108, 95, 102, 108, 100, and 114 milliseconds. Can you, with 95% confidence (also 90% confidence), state that system #2 has lower response time than system #1? Why or why not? Show your work. A T-score table is given below.

Selected values of $t_{\alpha/2;N-1}$

	$\alpha/2 = 0.05$	$\alpha/2 = 0.025$
N – 1	**t**	**t**
4	2.13	2.78
5	2.02	2.57
6	1.94	2.45
7	1.90	2.37
8	1.86	2.31
9	1.83	2.26
10	1.81	2.23
11	1.80	2.20
12	1.78	2.18

First we determine the difference between sample means (for system #1 minus system #2):

2 milliseconds
10
8
0
6
-2

It certainly looks like system #2 is better (i.e., has lower true mean response time).

$$\bar{Y} = \frac{1}{6} \cdot (2 + 10 + 8 + 0 + 28 + 6 + (-2)) = 4$$

$$S = \sqrt{\frac{1}{6-1} \cdot \left[(2-4)^2 + (10-4)^2 + \ldots + (-2-4)^2\right]} = 4.733$$

$t_{\alpha/2} = 2.57$ for 95% confidence and $t_{\alpha/2} = 2.02$ for 90% confidence (we use $\alpha = 0.05$ and N = 6 degrees of freedom, so we use the $N - 1 = 5$ row)

$$H_{95} = 2.57 \cdot \left(\frac{4.733}{\sqrt{6}}\right) = 4.966$$

$$H_{90} = 2.02 \cdot \left(\frac{4.733}{\sqrt{6}}\right) = 3.903$$

So, we can say with 95% confidence that the population mean (that is, true mean) lies between -0.966 and 8.966 (that is, 4 $+/-$ 4.966 Since this CI is **not** entirely above zero, we **cannot** say with 95% confidence that system #2 is better (better = lower response time in this case) than system #1. With 90% confidence the true population mean lies between 0.0.097 and 7.903 and thus we can say with 90% confidence that system #2 is better than system #1.

Extra Credit What it models is 4 pts. The output (blow up) is 6 pts.

What does the below program model? What is the output for the program?

```
#include <stdio.h>
#include "csim.h"

FACILITY Server;

void fred(double service_time);

void sim(void)
{
  double   lambda, mu;
  create("sim");
  Server = facility("Server");
  lambda = 5.0;
  mu = 1.0;
  while(1)
  {
    hold(exponential(1.0 / lambda));
    fred(exponential(1.0 / mu));
    if (clock > 1000000.0) break;
  }
  printf("= Utilization        = %f        \n", util(Server));
  printf("= Mean num in system = %f cust   \n", qlen(Server));
}

void fred(double service_time)
{
  create("queue1");
  reserve(Server);
  hold(service_time);
  release(Server);
}
```

This program models a single-server queue (an M/M/1 to be precise). The arrival rate is greater than the service rate, so it will “blow up” with the following error message:

****
CSIM EXECUTION ERROR 17 DETECTED AT TIME 235.155412
<<<<<<17. TOO MANY PROCESSES
****

Appendix A

```
//============================================================= file = mm1.c =====
//=  A simple "straight C" M/M/1 queue simulation with bugs                      =
//================================================================================
//=  Notes:                                                                      =
//=   1) This program is adapted from Figure 1.6 in Simulating Computer          =
//=      Systems, Techniqyes and Tools by M. H. MacDougall (1987).               =
//=   2) The values of SIM_TIME, ARR_TIME, and SERV_TIME need to be set.         =
//=-----------------------------------------------------------------------------=
//=  Build: gcc mm1.c -lm, bcc32 mm1.c, cl mm1.c                                 =
//=-----------------------------------------------------------------------------=
//=  Execute: mm1                                                                =
//=-----------------------------------------------------------------------------=
//=  History: KJC (03/09/99) - Genesis                                           =
//=           KJC (05/23/09) - Added RNG function (so not to use compiler RNG)   =
//=           KJC (05/24/09) - Added updated of busy time at end of main loop    =
//=           KJC (06/14/13) - Creates bugs for exam #1 summer 2013              =
//================================================================================
//----- Include files ------------------------------------------------------
#include <stdio.h>              // Needed for printf()
#include <stdlib.h>             // Needed for exit() and rand()
#include <math.h>               // Needed for log()

//----- Constants ----------------------------------------------------------
#define SIM_TIME   1.0e6        // Simulation time
#define ARR_TIME   1.25         // Mean time between arrivals
#define SERV_TIME  1.00         // Mean service time

//----- Function prototypes ------------------------------------------------
double rand_val(int seed);      // RNG for unif(0,1)
double exponential(double x);   // Generate exponential RV with mean x

//===== Main program ==========================================================
int main(void)
{
  double   end_time = SIM_TIME; // Total time to simulate
  double   Ta = ARR_TIME;       // Mean time between arrivals
  double   Ts = SERV_TIME;      // Mean service time
  int      time = 0.0;          // Simulation time
  double   t1 = 0.0;            // Time for next event #1 (arrival)
  double   t2 = SIM_TIME;       // Time for next event #2 (departure)
  unsigned int n = 0;           // Number of customers in the system
  unsigned int c = 0;           // Number of service completions
  double   b = 0.0;             // Total busy time
  double   s = 0.0;             // Area of number of customers in system
  double   tn = time;           // Variable for "last event time"
  double   tb;                  // Variable for "last start of busy time"
  double   x;                   // Throughput
  double   u;                   // Utilization
  double   l;                   // Mean number in the system
  double   w;                   // Mean residence time

  // Seed the RNG
  rand_val(1);

  // Main simulation loop
  while (end_time < time)
  {
    if (t1 < t2)                // *** Event #1 (arrival) ***
    {
      time = t1;
```

double

time < end time

```
      s = s + n * (tn - time);  // Update area under "s" curve
      n++;
      tn = time;                // tn = "last event time" for next event
      t1 = time + exponential(Ta);
      if (n == 1)
      {
        tb = time;              // Set "last start of busy time"
        t2 = time + exponential(Ta);
      }
    }
    else if (t1 < t2)          // *** Event #2 (departure) ***
    {
      time = t2;
      s = s + n * (tn - time);  // Update area under "s" curve
      n--;
      tn = time;                // tn = "last event time" for next event
      c++;                      // Increment number of completions
      if (n > 1)
        t2 = time + exponential(Ta);
      else
      {
        t2 = SIM_TIME;
        b = b + time - tb;      // Update busy time sum if empty
      }
    }
  }

  // End of simulation so update busy time sum
  b = b + time - tb;

  // Compute outputs
  x = c / time;   // Compute throughput rate
  u = b / time;   // Compute server utilization
  l = s / time;   // Compute mean number in system
  w = l / x;      // Compute mean residence or system time

  // Output results
  printf("============================================================= \n");
  printf("=           *** Results from M/M/1 simulation ***           = \n");
  printf("============================================================= \n");
  printf("=  Total simulated time         = %3.4f sec   \n", end_time);
  printf("============================================================= \n");
  printf("=  INPUTS:                                    \n");
  printf("=    Mean time between arrivals = %f sec      \n", Ta);
  printf("=    Mean service time          = %f sec      \n", Ts);
  printf("============================================================= \n");
  printf("=  OUTPUTS:                                   \n");
  printf("=    Number of completions      = %ld cust    \n", c);
  printf("=    Throughput rate            = %f cust/sec \n", x);
  printf("=    Server utilization         = %f %%       \n", 100.0 * u);
  printf("=    Mean number in system      = %f cust     \n", l);
  printf("=    Mean residence time        = %f sec      \n", w);
  printf("============================================================= \n");

  return(0);
}
```

time - tn

Ts

elide

time - tn

n--;

0

Ts

```
//=========================================================================
//= Multiplicative LCG for generating uniform(0.0, 1.0) random numbers    =
//=   - x_n = 7^5*x_(n-1)mod(2^31 - 1)                                    =
//=   - With x seeded to 1 the 10000th x value should be 1043618065       =
//=   - From R. Jain, "The Art of Computer Systems Performance Analysis," =
//=     John Wiley & Sons, 1991. (Page 443, Figure 26.2)                  =
//=   - Seed the RNG if seed > 0, return a unif(0,1) if seed == 0         =
//=========================================================================
double rand_val(int seed)
{
  const long  a =      16807;  // Multiplier
  const long  m = 2147483647;  // Modulus
  const long  q =     127773;  // m div a
  const long  r =       2836;  // m mod a
  static long x;               // Random int value (seed is set to 1)
  long        x_div_q;         // x divided by q
  long        x_mod_q;         // x modulo q
  long        x_new;           // New x value

  // Seed the RNG
  if (seed != 0) x = seed;

  // RNG using integer arithmetic
  x_div_q = x / q;
  x_mod_q = x % q;
  x_new = (a * x_mod_q) - (r * x_div_q);
  if (x_new > 0)
    x = x_new;
  else
    x = x_new + m;

  // Return a random value between 0.0 and 1.0
  return((double) x / m);
}

//=========================================================================
//=  Function to generate exponentially distributed RVs using inverse method   =
//=    - Input:  x (mean value of distribution)                           =
//=    - Output: Returns with exponential RV                              =
//=========================================================================
double exponential(double x)
{
  double z;                    // Uniform random number from 0 to 1

  // Pull a uniform RV (0 < z < 1)
  do
  {
    z = rand_val(0);
  }
  while ((z == 0) || (z == 1));

  return(-x * log(z));
}
```