CZ1106 Problem Solving and Computation II

Programming Lab 5

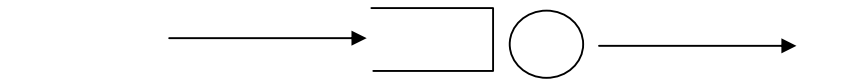
Discrete Event Simulation and Linked List

29 March 2007

In this assignment we are going to do discrete event simulation with the use of linked list taught in class. First, why do we write a simulation program to study the behavior of a phenomenon? A simulation study must have a purpose. For example, simulation can be performed to check and to optimize the design of seaport facilities and airport run-ways before their construction, thus avoid costly design errors. Other purposes include analysis, performance evaluation, tests of sensitivity, cost effectiveness, forecasting, safety, man-in-the-loop training, teaching, decision making, etc. How about the simulation of TOTO, and the 4D lotteries which you might have attempted? The simulation result will tell you that it does not gain to gamble.

In this assignment we are going to study the behaviour of the queue in front of a service counter. Queue is common everywhere, such as the ATM machines in Yusof Ishak House, the check out counter in Science Library, the queue you join in Science canteen when you take your lunch, etc. If you are frustrated when standing in a queue, this assignment will give you some insight of the queue behaviour and I hope this will make you less frustrated when you have understood the queue mechanism!! Here it is.

A queuing system is described by its calling population, the nature of the arrivals and services, the system capacity, and the queuing discipline. A simple queuing system can be modeled as follows:



Calling Population | Arrival | Queue | Server | Departure

In this system the calling population is infinite. That is, if a unit (this unit can be also called customer) leaves the calling population and joins the waiting line or enters service, there is no change in the arrival rate of other units that may need service. In this system, arrivals for service occur one at a time in a random fashion and once they join the waiting line they are eventually served. In addition, service times are of some random length according to a probability distribution which does not change over time. Also, the system capacity may be unlimited or limited. The system capacity includes the unit in service plus those waiting in line. Finally, callers are served in the order of their arrival (often called FIFO, for first in, first out) by one or more servers. In this assignment we assume that the queue capacity is unlimited, and the number of servers is 1.

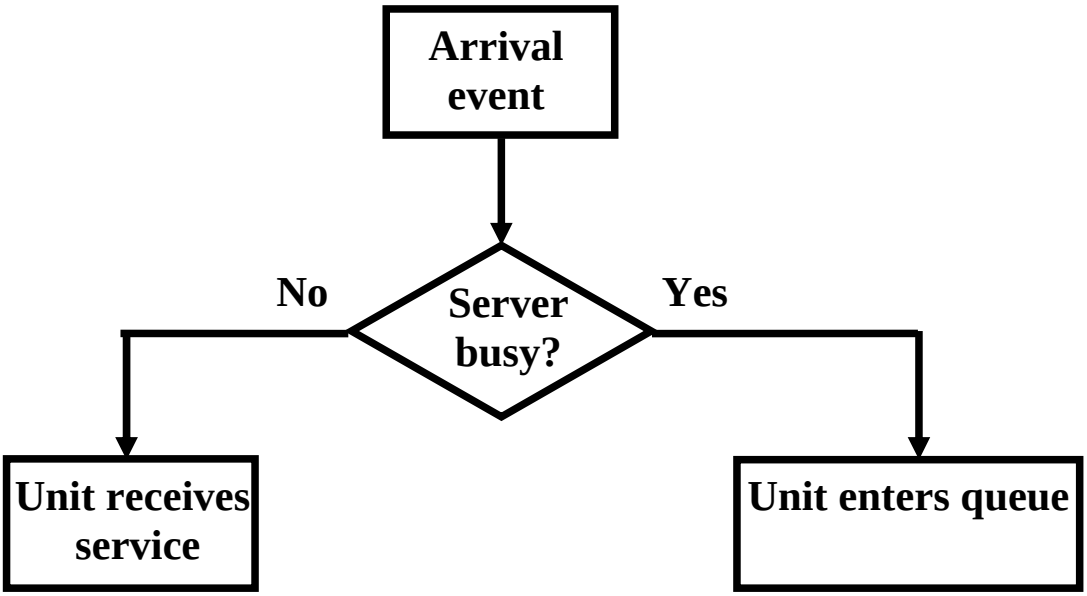
Arrivals are modeled by the distributions of the time between arrivals called inter-arrival time. Service time, which is also a random number, refers to the time needed to serve a customer at the counter. If the overall effective arrival rate is greater than the

overall service rate, the waiting line will grow without bound. These queues are termed as explosive queues - I hope you have not experienced it.

Prior to introducing several simulations of queuing systems, it is necessary to understand the concepts of **state** of the system, **events**, and **clock time**. The **state** of the system is the number of units in the system. The status of the server can be busy or idle. An **event** is a set of circumstances that cause an instantaneous change in the state of the system.

In a single-queue system there are only two possible events that can affect the state of the system. The two events are the entry of a unit into the system (the **arrival event**), and the completion of service on a unit (the **departure event**). The queuing system includes the server, the unit being served (which is the customer standing in front of the server – this customer is not standing in the queue), and the unit(s) in the queue (if any are waiting).

Arrival Event. Arrival event occurs when a unit enters the system. The unit may find the server either idle or busy. Therefore, the arrived customer will be served, or it will enter the queue as shown in the following flow chart.

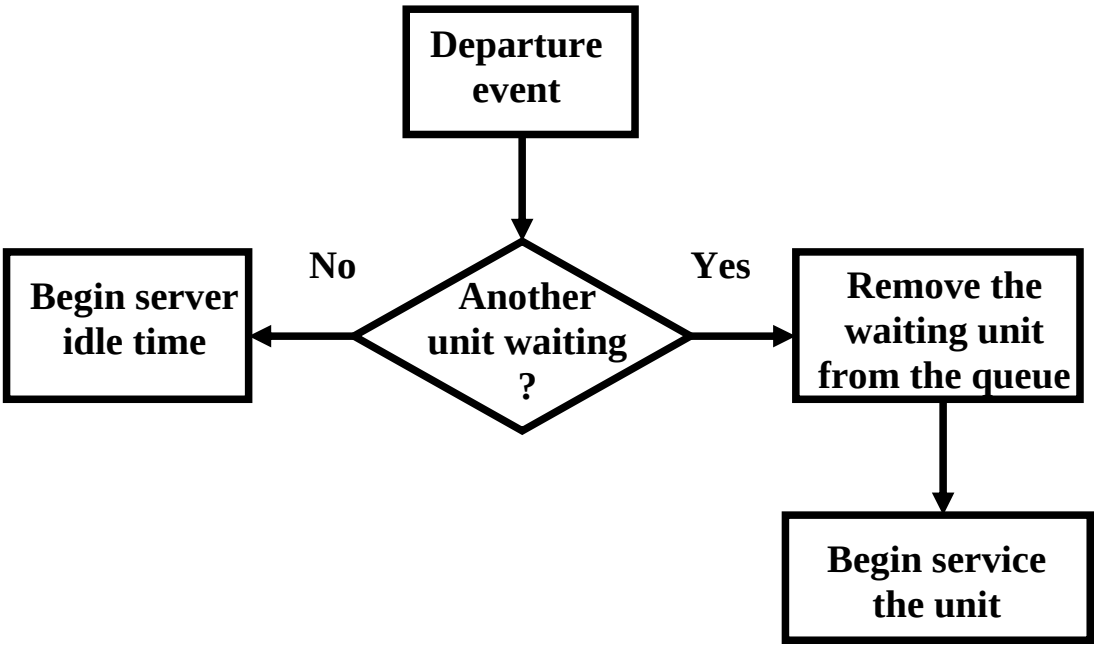


The arrived unit follows the course of action shown in the following table.

		Queue status	
		Not empty	*Empty*
Server status	*Busy*	Enter queue	Enter queue
	Idle	Impossible	Enter service

If the server is busy, the unit enters the queue. If the server is idle and the queue is empty, the unit will be served by the server. It is impossible for the server to be idle and the queue to be not empty at the same time.

Departure Event. If a service has just been completed, the simulation proceeds in the manner shown in the following flow chart. Note that the server has only two possible conditions either busy or idle.



After the completion of a service the server may become idle, or remain busy with the next unit. The relationship of these two outcomes to the status of the queue is shown in the following table.

		Queue status	
		Not empty	*Empty*
Server outcomes	*Busy*		Impossible
	Idle	Impossible	

If the queue is not empty upon a service completion, another unit will be served by the server and its status will remain as busy. If the queue is empty, the server will be idle after a service is completed. It is impossible for the server to become busy if the queue is empty when a service is completed. Similarly, it is impossible for the server to be idle after a service is completed when the queue is not empty.

Now, how can the arrival event and department event occur in a simulated time? Simulations of queuing systems generally require the maintenance of an event list for determining what happens next. The event list, which is also called calendar, indicates the times at which the different types of events occur for each unit in the queuing system. The times are kept on a "clock", which marks the occurrences of events in time. In simulation, the events usually occur at random. The randomness imitates real life to portray uncertainty. For example, it is not known with certainty when the next customer will arrive the loan counter in Science Library (inter-arrival time), or how long the bank teller will take to complete a transaction (service time). This event list is implemented by linked list in this assignment.

We shall now go through a series of hand simulation before embarking on writing the program. The examples that follow show how the inter-arrival time and service time are generated. For simplicity, we assume that the times between arrivals were generated by rolling a die six times and recording the value facing up. This will imitate the uniform distribution as the dice is unbiased. The following table contains a set of six inter-arrival times generated in this manner.

Customer	Inter-arrival Time	Arrival Time on Clock
1	2	2
2	1	3
3	4	7
4	1	8
5	2	10
6	6	16

These six inter-arrival times are used to compute the arrival times of six customers at the queuing system. The system clock time starts with 0. The first customer arrives at clock time 2. This starts the clock in operation. The second customer arrives one time unit later, at a clock time of 3. The third customer arrives four time units later, at a clock time of 7; and so on.

The second time of interest is the service time. The following table contains service times generated at random from a distribution of service times.

Customer	Service Time
1	2
2	1
3	3
4	2
5	1
6	4

The only possible service times here are one, two, three, and four time units for examples. Assuming that all four values are equally likely to occur, these values could have been generated by placing the numbers one through four on chips and drawing the chips from a hat with replacement.

Now, the inter-arrival times and service times must be meshed to simulate the single-queue system. As shown in table below, the first customer arrives at clock time 2, and immediately begins service with a duration of 2 minutes. Service is completed at clock time 4. The second customer arrives at clock time 3 and will have to be placed in the queue because the server has not completed the service for the previous unit. Once the service for the previous unit (1^{st} arrival) is completed at clock time 4, the service will start for the second unit and will finish at clock time 5.

Customer Number	Arrival Time (Clock)	Time Service Begins (Clock)	Service Time (Duration)	Time Service Ends (Clock)
1	2	2	2	4
2	3	4	1	5
3	7	7	3	10
4	8	10	2	12
5	10	12	1	13
6	16	16	4	20

The chronological ordering of these events are as follows:

Event Type	Customer Number	Clock Time
Arrival	1	2
Arrival	2	3
Departure	1	4
Departure	2	5
Arrival	3	7
Arrival	4	8
Departure	3	10
Arrival	5	10
Departure	4	12
Departure	5	13
Arrival	6	16
Departure	6	10

The simulation algorithm is rather systematic and iterative as shown in the following algorithm:

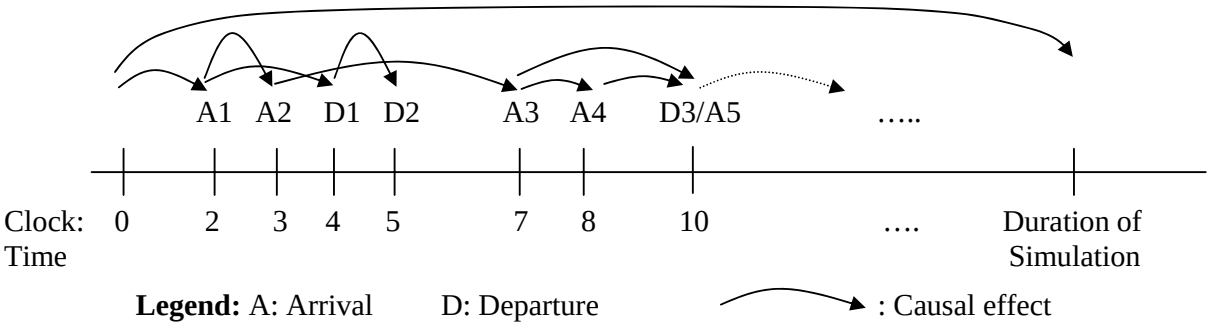
```
// initialization
set clock to 0;
set server to idle;
set queue length to 0;
schedule the first arrival event in the event list;
schedule a End_Of_Simulation Event in linked list;

// iteration
repeat
  select the next event with the smallest occurrence time
      from the outstanding events;
  update the system state in the time interval
      [clock, next event time];
  advance clock to the next event occurrence time;
  execute the next event according to its event type;
until  clock > duration_of_simulation;
```

You observe that the increment of the simulation clock is discrete. In each iteration in the algorithm these 4 steps are executed:

- select the event of least time;
- update the system state
- advance clock time
- execute event

The advancement of discrete clock time is shown below:



Legend: A: Arrival D: Departure : Causal effect

The algorithm for executing an arrival event is as follows:

```
increment the total number of arrivals by 1;
schedule the next arrival event in the linked list;
if server is idle
  {
    set server to busy;
    schedule a departure event for this
       arrival in the linked list;
  }
else
  {
    increment queue length by 1;
    increment the total number of waiters;
  }
```

The algorithm for executing a departure event is as follows:

```
increment the total number of departures by 1;
if queue length is 0
    set  server to idle;
else
  {
    decrement queue length by 1;
    schedule the next departure event in
       the linked list;
  }
```

Assignment

Now we are ready to write the program to simulate the checkout counter in a grocery store. Assume the store has only one checkout counter. Customers arrive at this counter at random from 1 to 8 minutes apart. The cashier requires 1.5 to 3 minutes to serve each customer at the counter. Both the inter-arrival and service time are uniformly distributed. The cashier complained that her workload was too high and asked the boss to add one more checkout counter. On 5 occasions where she complained, there were at least 4 persons standing in the queue waiting for her service. The cashier used these evidences to put pressure on the boss to do something or she would submit a compliant to the union as she claimed the situation had affected her mental state. As the operating cost for the 2 parallel checkout counters will be doubled, you are now engaged to analyze the situation. We are interested in the average queue length, average queue time, wait probability and the server (cashier) utilization. If the store operates from 9 am to 10 pm, how many customers are served by the cashier for each day? Print on the screen these answers:

```
Duration of Simulation : 1000000.00 hours
Total no. of Arrival events executed: ??
Total no. of Departure events executed: ??
Ave Queue Length: ??
Ave Queue Time: ??
Wait Probability: ??
Server Utilization:
Average number of customers served for each day (13 hours): ??
```

Write a report to advise the boss what to do based on your simulation results.

You can start from this program:

```
// server.c to compute
//   average queue length, average queue time, wait probability
//   and server utilization of a service counter.

// This program makes use of linked list in the
//   simulation event management.


#define IDLE 0
#define BUSY 1
#define ARRIVAL 2
#define DEPARTURE 3
#define END_OF_SIMULATION 4

#define working_hr 13.0
#define duration_of_simulation 1000000.0
#define seed 107

# include <math.h>
# include <stdio.h>
# include <conio.h>

struct node
{
  double key;
  int event;
  struct node * link;
};


double service_time()
{
   return (  (double)rand()/32767 )*1.5 + 1.5;
}

double inter_arrival_time()
{

}

void insert ( ..... )
{
  /*  insert a node with event time and event type to the link list in
     accending order
  ..
  ..
  */
}

void remove_head()
{
  /* remove the first node of linked list and free the space;
  ..
  ..
  */
}

main ()
{
   int server_status;
   long total_arrivals, total_departures;
   long total_waiters, queue_length;
```

```
   double total_idle_time, total_wait_time;

   double clock,time_of_next_arr, time_of_next_dep,
        time_of_close_shop, event_time;

   int  event;

   long last_que_len;
   double ave_que_len, ave_que_time, wait_probability, utilization;

   struct node *new_node, *current;

   clrscr();

// initilialzation

..
..

  srand (seed);

// insert first arrival event, and end of simulation event
//     to linked list in ascending order

  do
   {
      // step 1: remove first node as the event


      // setp 2: record progress
      if (server_status == IDLE)
        total_idle_time += (event_time-clock);
      else
        total_wait_time += (queue_length*(event_time-clock) );

      // step 3: advance clock to this event time
      // ..

      // step 4: execute the event accordingly
      switch (event)
      {
       case ARRIVAL:
         ..
         ..
         break;

       case DEPARTURE:
         ..
         ..
         break;
      case END_OF_SIMULATION: ; // do nothing
       }// case
   }
   while (clock < duration_of_simulation);

   ave_que_len      = total_wait_time/duration_of_simulation;
   ave_que_time     = total_wait_time/(double)total_arrivals;
   wait_probability = (double)total_waiters / (double)total_arrivals;
   utilization      = (duration_of_simulation - total_idle_time) /
                                              duration_of_simulation;

   printf ("\nSimulation simulation_duration : %.2lf ", duration_of_simulation);
   printf ("\nTotal Arrivals: %ld", total_arrivals);
   printf ("\nTotal Departures: %ld", total_departures);
```

```
   printf ("\nAve Queue Length: %lf",ave_que_len);
   printf ("\nAve Queue Time: %lf", ave_que_time);
   printf ("\nWait Probability: %lf", wait_probability);
   printf ("\nServer Utilization: %lf", utilization);

   printf ("\nAve number of customer served for each day: %lf ",
   ( (double) total_departures/duration_of_simulation)*working_hr*60);

   return 0;
}
```

Hand in your printed program and report on the next Wednesday in LT.

Send me your program in email attachment on the next Wednesday before 10am to scitaysc@nus.edu.sg.