Royal University of Phnom Penh

Department: **Computer Science**
Scholaship Students in Promotion 7 of Computer Science

Year IV, Semester I

Simulation & Modeling

on

"Barber Shop"

Group Number: 3

Mr. Iech Setha
Mr. Khek Sokhom
Mr. Tat Ratanak
Mr. Ngoun Chansaravuth
Mr. Vong Trim

Submit to Mr. Hin Sam Ath

Date: 18-Dec-2004

Abstract

The article describes about building a Simulation on **barbershop**. Choose the single server queuing system model to build this simulation. The program generation will describe on Tourbo C++ programming language.

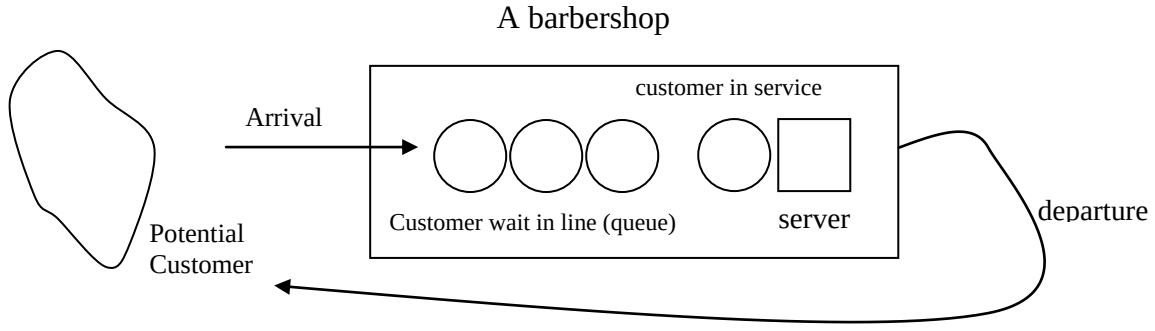
I- Problems and Objective:

Before building the real barbershop system, it is very important to build some models to simulate about the system because it is so risk to build the real once. It might face many problems during the system process. For instance, a barbershop that has only one chair of service, and customers come a lot, some customers will wait in line in the barbershop; some customer do not want to wait (go away), some customers delay for long time before they start service, so the barbershop will lose some customers. To solve the problem, we estimate and analyze the result of the simulation after running to make the decision to the barbershop.

Simulation is running on computer system, and builds on general-purpose programming languages such as C, C++, Java or special-purpose simulation languages. In this article, we just use general-purpose programming language ─ C programming language to build the simulation.

II- Define System Model:

There are many type of barbershop system model, such as, one chair model, two chairs model. We choose the first model is one chair model to simulate on a barbershop. Assume that the barbershop has a single chair served by a single service man. Naturally, barbershop system is the queuing system. It means that when customers arrive they always wait in line if the barber is busy, but if the barber is free then they start service. We build the simulation to reduce the number of customers waiting in queue and gain more customers in service. Practically, simulation on barbershop is to find out the mean of customer waiting in queue, mean of time that they wait for service, and the percentage of barber that busy. With the one chair model, we choose a single server queuing system to simulate this system.



III- Collect Data:

We need to collect data to get the sample input parameters such as, inter-arrival time and service time. We sample the two different barbershops in a whole two day. Both barbershops have only one chair of service.

✍ Barbershop Angtasom starts at 7:45 AM and end at 5:00 PM. We found the mean of inter-arrival is between 8.69 and the 34.65 minutes. The mean of service time is between 20.17 and 28.39 minutes.

* Inter-arrival time (minute): 30; 15; 25; 40; 40; 10; 15; 3; 17;.....
* Service time (minute): 27; 32; 24; 24; 20; 22; 21;.....

✍ Barbershop Samros Neary Khmer starts at 7:00 AM and end at 8:00 PM. We fount the mean of inter-arrival time is between 6.43 and 28.11 minutes. The mean of service time is between 20.84 and 28.3 minutes.

* Inter-arrival time (minute): 15; 22; 5; 38; 20; 3; 19; 10; 18; 8; 32;.....
* Service time (minute): 26; 28; 21; 25; 20; 30; 22;.....

After the collection of data, we found the operations in barbershop are *customer arrival, customer departure, customer start at service, and the operation of barber serve the service*.

IV- Conceptual Model:

In barbershop, when a customer arrives first time of a day he ready to start service because the barber is free. During the barber is busy other customers arrive, so they wait for the service in line (queue). We assume that the arrival times and service times of each customer are randomly, and the times are discrete. First, identify about:

- System model: single server queuing system
- Events: there are two events occur in the barbershop that is event of customers arrival and event of customers departure.
- System States: define the number of customers waiting for service and number of customer served at any moment.
- Random realization: generate the random number of each inter-arrival times and service times to define the next arrival time and departure time.

☞ Components or System variables

- *Server status:* defines the server (barber) is busy or idle.
- *Number in queue:* defines the number of customer waiting in queue.
- *List of arrival time:* stores the arrival time of customers are being waiting in queue. The time in list, use to calculate the delay time before they start service.
- *Time of last event:* stores the time of event just occur at moment whether the times are arrival or departure.
- *Event list:* store the next arrival time and next departure time to define the simulation clock.
- *Simulation clock:* identifies the time simulation time.
- *Statistic counter:* uses to estimate the system, such as find the mean of customer waiting in queue, the mean of time that customer wait for service, find the percentage of server busy. There are four elements of statistic counter to be calculated.

  + *Number of delay:* number of customers served at any moment. Number of delay increase one (Number of delay=Number of delay + 1) when customer starts of service.

  + *Total delay:* total the delay time of each customer from the arrival time to start service time.

    If the customer waiting in queue then:

    Total delay=Total Delay + (Start Service Time-First Arrival Time in List);

  + *Area under Q(t):* total time of all customer waiting in queue until now.

    $Q(t_n)=Q(t_{n-1})+(nq)*(t_n-t_{n-1})$

    $Q(t_n)$: Area under Q(t) current event

    $Q(t_{n-1})$: Area under Q(t) of previous event

    nq: Number of customer waiting in queue at now

t_n: time of current event – now
t_{n-1}: time of previous event

+ *Area under B(t):* total time of server that serve the service – server is busy
If server busy at the occurrence of current event then
$B(t_n)=B(t_{n-1})+(t_n-t_{n-1})$
$B(t_n)$: Area under B(t) of current event
$B(t_{n-1})$: Area under B(t) of previous event
t_n: time of current event (now)
t_{n-1}: time of previous event

☞ Specify what to do at each event. At event of arrival, create next arrival. If the server is free or idle, send the customer to start service, then server become busy. Otherwise, it joins the queue. At event of service end, then server become free. If any customers waiting in queue remove first customer from the queue; send it for start of service.

☞ When first customer arrive, the server become occupied, the number of delay increase one. Define the next event whether customer arrival or customer departure, is to generate the random number of inter-arrival time and service time between the minimum and maximum of inter-arrival and service time that have been given.

☞ Define the next event: arrival or departure then move the simulation clock the next event time. The event occur, update system state (server status, number in queue, time of last event), statistic counter. Generate the next arrival time or next departure time according to the event occur. Then practice this step again until the number of delays equal to the number of required customers.

☞ After the simulation end, we get all the value of statistic counter, and the time of simulation end. Then we can calculate the mean of customer waiting in queue, mean of delay time, and percentage of server utilization to estimate of simulation on system.

$$mean\,of\ customer = \frac{area\,under\,Q(t)}{time\,of\ simulation\,end}$$

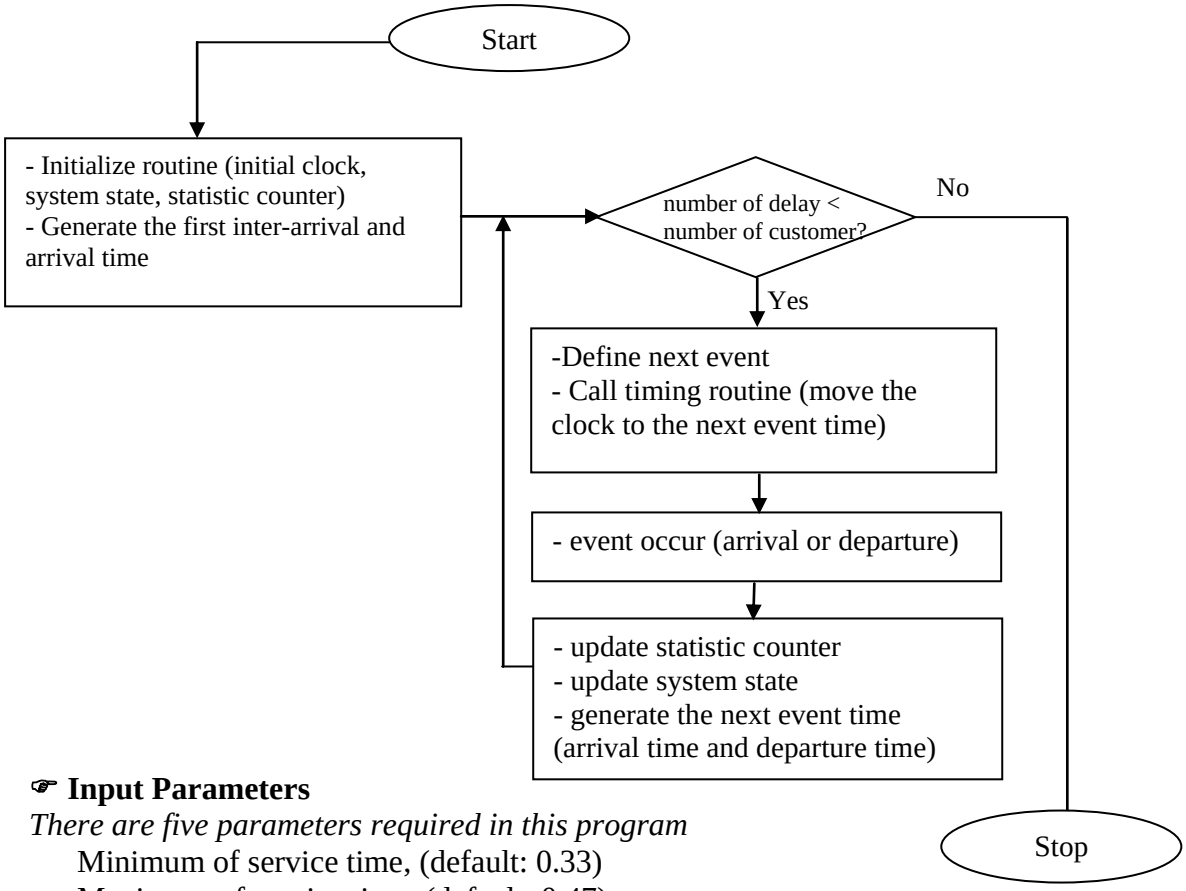
$$mean\,of\ delay\,time = \frac{total\ delay}{number\,of\ delay}$$

$$server\,utilizatio\,n = \frac{area\,under\,B(t)}{time\,of\ simulation\,end}$$

V- Program Organization and Logic:

The simulation program builds on C programming language in graphic mode. The program combines of program logic, input parameters, variables, events, sub routines, library routines, and other.

☞ **Program Flowchart**



☞ **Input Parameters**

There are five parameters required in this program

Minimum of service time, (default: 0.33)
Maximum of service time, (default: 0.47)
Minimum of inter-arrival time, (default: 0.14)
Maximum of inter-arrival time, (default: 0.57)
Number of required customers, (default: 30)

☞ **Define Variables**

BUSY	1
IDLE	0
MAX_CUSTOMER	1000

☞ **Variables Declaration (System state and Input parameter variables)**

Type	**Variable Name**	**System Description**
int	srv_status	Server Status
	num_delay	Number of Delay
	event	Event Type (Arrival or Departure)
	nq	Number in queue
	num_customer	Number of required customer
	in_srv	Store arrival time being in service
float	min_s_time	Minimum of service time
	max_s_time	Maximum of service time
	min_a_time	Minimum of inter-arrival time
	max_a_time	Maximum of inter-arrival time
double	l_event	Last event time
	clock_sm	Store simulation time

	tdl	Total delay
	auqt	Area under Q(t)
	aubt	Area under B(t)
	nextarv	Next arrival time
	nextdpt	Next departure time (0: infinite)
	queue[MAX_CUSTOMER]	List of arrival time

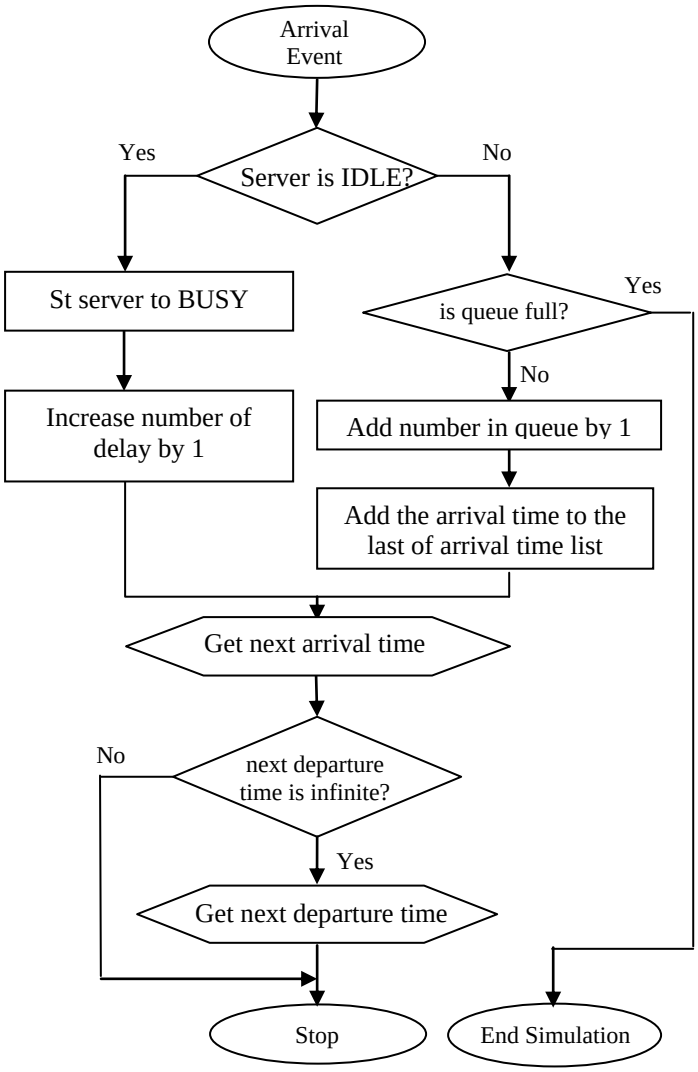
☞ **Events Description**

Event	**Event Type**
Arrival	0
Departure	1

***- Arrival Event*:**

When the customer arrives, check the server status. If the server status is idle, then update server status to busy. Otherwise, that customer waits in queue, so the number of customers in queue increases one and store the arrival time to the list of arrival time.

Flowchart



Pseudo-code

```
Arrival
if server is IDLE? then
       - set server status to BUSY
       - add 1 to number of delay
else
       if queue is full? then
              end simulation
       else
          - add 1 to number in queue
          - add the arrival time to
            the last of arrival list
       end if
end if

generate the next arrival time
if next departure time is infinite?
       generate the next departure
time
end if
```

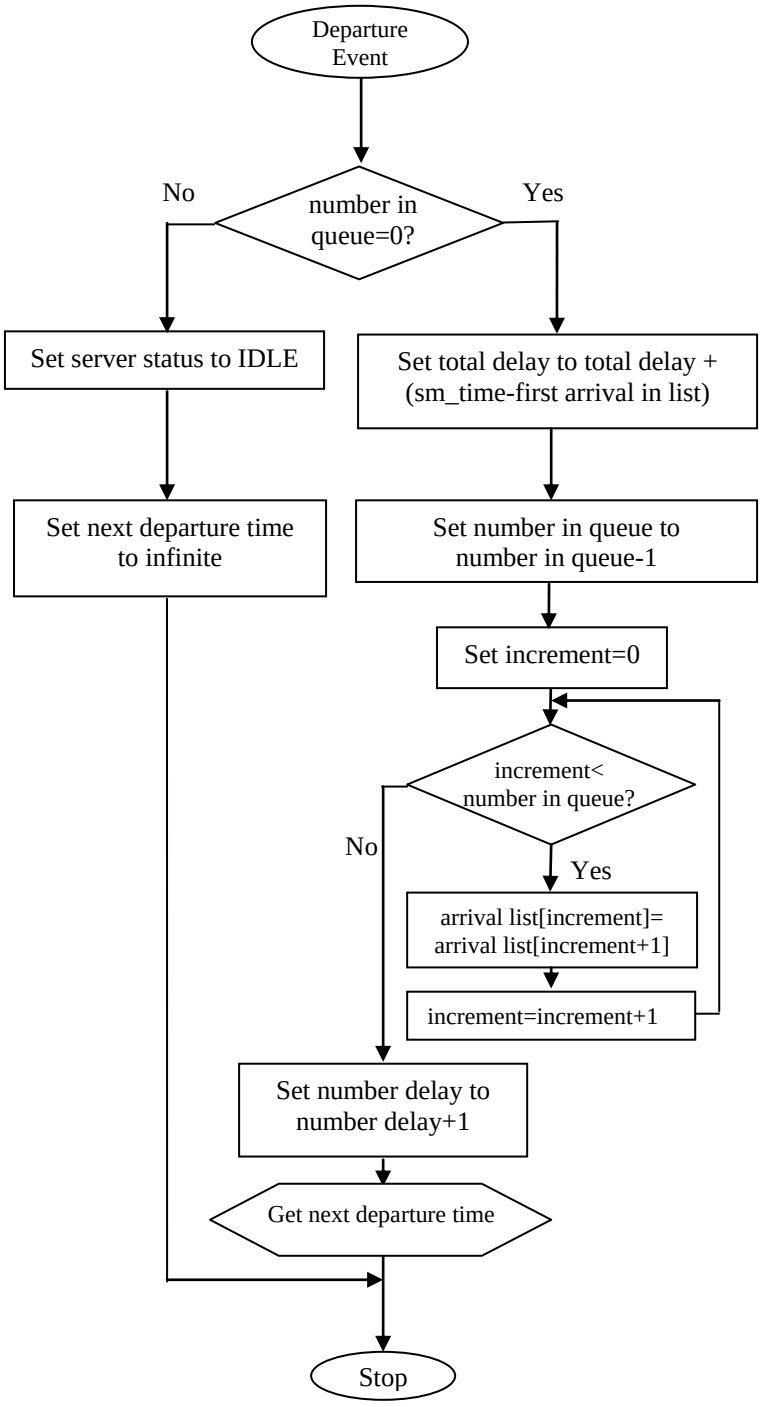
C Code

```
void arrival(){
   if(srv_status==IDLE) {
        srv_status=BUSY;
        num_delay+=1;

        }
   else {
       if(nq<100){
              nq=nq+1;
              queue[nq-1]=clock_sm;}
       else exit(1);

       }//end else
    nextarv=getnextarrival();
    if(nextdpt==0)
  nextdpt=getnextdepart();
}
```

- Departure Event

When a customer departs, check the customers in queue. If no customer in queue, server becomes idle and the next departure is infinite. If customers exist in queue, send the first customer of queue to start the service. Then update the total delay of time, shift the list of arrival to front, decrease the number in queue by number 1, and generate the next departure time.

Flowchart



Pseudocode

```
departure:

if number in queue is 0 then
       - set server status to IDLE
       - set next depart time to 0
         (infinite)
else
       - update total of delay
         total delay=total delay+
         (sm_time-queue[0]))
       - set number in queue to
         number in queue-1
       - loop to update the list of
         arrival time
         (repeat queue[i]=qeueu[i+1])
       - add 1 to number of delay
       - generate next departure time
end if
```

C Code

```
void departure()
{    int i;
     if(nq==0) {
       srv_status=IDLE;
       nextdpt=0;    //infinite
                }
     else
      {//calculate total delay
       tdl=tdl+(clock_sm-
                queue[0]);
       nq-=1; //number in q
       //update list
       for(i=0;i<nq;i++)
           queue[i]=queue[i+1];
       num_delay+=1;
       //next departure time
       nextdpt=getnextdepart();
       }//end else
}
```

☞ Simulation Routine

- initialize routine

Initialize routine use to initialize the simulation clock, system state, first arrival time, and statistic counters.

Pseudo-code

```
initialize_sm
- set simulation clock to 0
- set server status to IDLE
- set number in queue to 0
- set time of last event to 0
- set number of delay to 0
- set total delay to 0
- set area under Q(t) to 0
- set area under B(t) to 0
- set event type to -1 (no event)
- set the arrival time being
  in service to 0
- initialize seed of random number
- generate the first arrival time
- round the arrival time 2 digit
  after decimal place
- set next departure time to 0
```

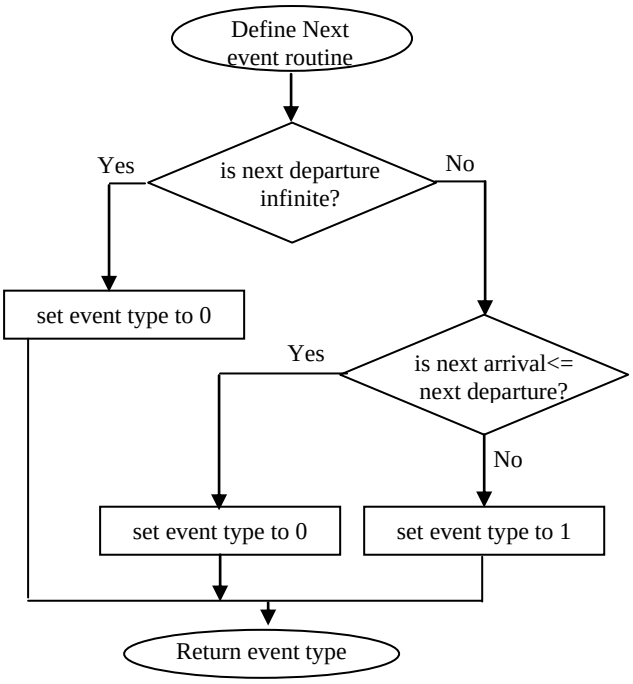
C Code

```
void initialize_sm()
{
    clock_sm=0;
    srv_status=IDLE;
    nq=0;
    l_event=0.0;
    num_delay=0;
    tdl=0.0;
    auqt=0.0;
    aubt=0.0
    event=-1;
    in_srv=0;

   //initialize random
    srand((unsigned) timer());
    nextarv=randfn(min_a_time,max_a_time);
    //initial first arrival
    nextarv=round(nextarv,2);
    nextdpt=0;       //infinite
}
```

- Define next event routine

Define the next event whether the event is arrival or departure. To get the next event, we just look to the next arrival time and next departure time.
Event type: Arrival:0, Departure: 1

Flowchart



Pseudo-code

```
nextevent
if next departure=0 then
       set event type=0
else if next arrival<=
     next departure then
       set event type=0
else
       set event type=1

return event type
```

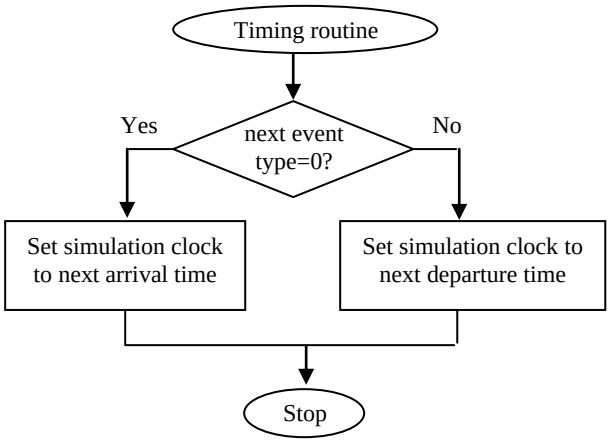
C Code

```
int nextevent()
{
       //define 0: arrival
       //define 1: departure

   if(nextdpt==0) event=0;
   else
      {if(nextarv<=nextdpt)event=0;
       else event=1;
      }

   return(event);
}
```

- Timing routine

Timing routine defines the simulation clock to the next event time.

Flowchart



Pseudo-code

```
sm_timing
if next event type=0 then
   - set clock to next
     arrival time
else
   - set clock to next
     departure time
end if
```

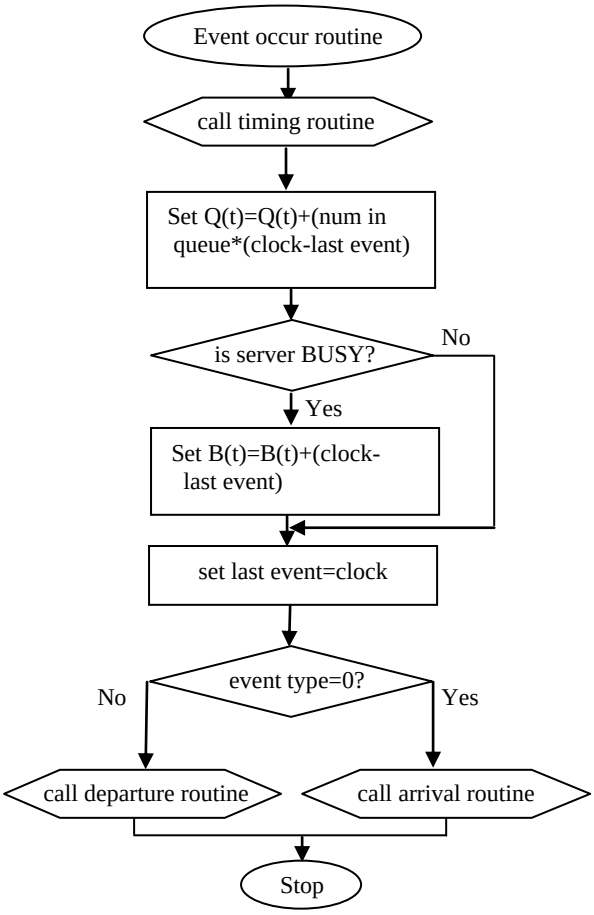
C Code

```
void sm_timing()
{
  if(nextevent()==0)
       clock_sm=nextarv;
  else clock_sm=nextdpt;
}
```

- Event occur routine

Firstly, call the timing routine to move the clock to the event. Then update the area under Q(t), area under B(t), and time of last event. If the next is arrival, call the function *arrival()*. Otherwise, call the function *departure()*.

Flowchart



Pseudo-code

event_occur

```
- execute timing routine
- set Q(t) to Q(t)+(num in queue)*(
  clock time-last event)
if server is BUSY then
  - set B(t) to B(t)+(clock time-last event)
end if

- set last event to clock time

if event type=0 then
  - execute arrival event
else
  - execute departure event
end if
```

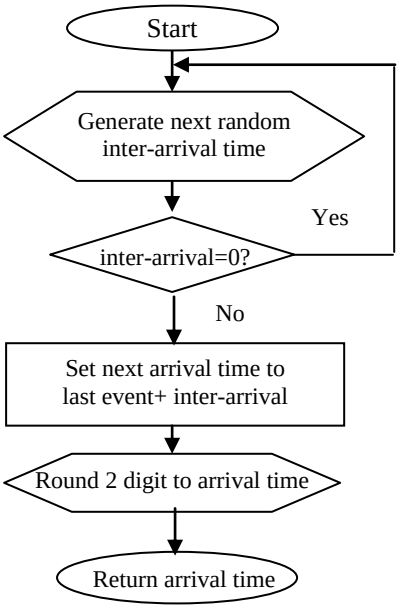
C Code

```
void event_occur()
{
       sm_timing();
       auqt=auqt+nq*(clock_sm-l_event);
       aubt=aubt+(srv_status==IDLE?
                  0:clock_sm-l_event);
       l_event=round(clock_sm,2);


       if(event==0)
              arrival();
       if(event==1)
              departure();
}
```

- Generate next arrival time routine

This routine is to generate the next arrival time randomly.

Flowchart



Pseudo-code

```
getnextarrival
- generate next random inter-arrival time
- if inter-arrival time is 0? then
       generate again
  end if
- next arrival=last event+inter-arrival
- round 2 digit to next arrival time
- return arrival time
```

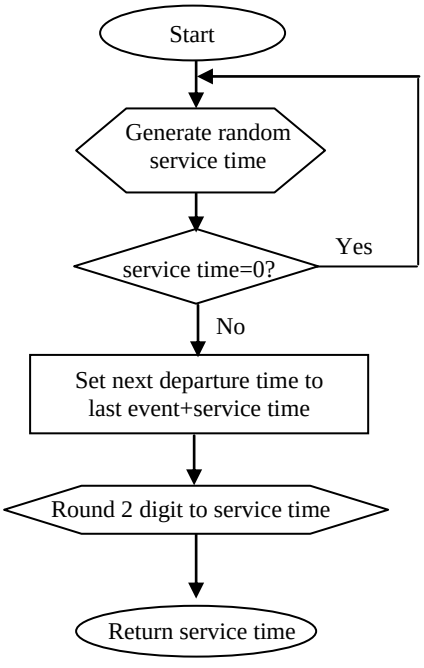
C Code

```
double getnextarrival()
{     double inarv;
    again:
      inarv=randfn(min_a_time,
                   max_a_time);
      if(inarv==0) goto again;
      inarv=inarv+l_event;
      //get next arrival time
      inarv=round(inarv,2);

      return inarv;
}
```

- Generate next departure time routine

This routine is to generate the next departure time randomly.

Flowchart



Pseudo-code

```
getnextdeparture
- generate random service time
- if service time is 0? then
       generate again
  end if
- next departure=last event+service time
- round 2 digit to the departure time
- return departure
```

C Code

```
double getnextdepart()
{     double srvtime;
     agd:
      srvtime=randfn(min_s_time,
                     max_s_time);

      if(srvtime==0) goto agd;

      srvtime=srvtime+l_event;
      srvtime=round(srvtime,2);
      return srvtime;
}
```

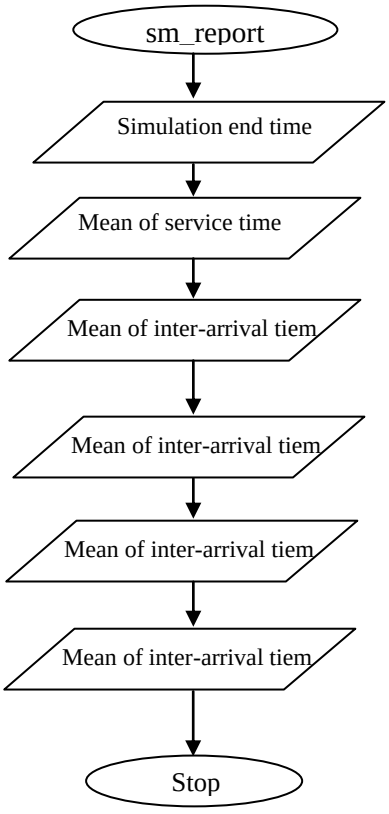
- Report routine

The report shows about time of simulation end, mean of customer in queue, mean of delay time, and the percentage of server utilization.

Pseudo-code

```
sm_report

- write "Mean of inter-arrival time:"
- write "Mean of service time: "
- write "Number of customers:"
- write "Time of simulation end: "=clock_sm
- write "Mean of customer in queueu: "=auqt/clock_sm
- write "Mean of delay time: "=tdl/num_customer
- write "Server utilization: "+aubt/clock_sm
```

Flowchart



☞ **Additional Routine**

- Generate random number between x and y

Generate random number as a floating point between two numbers.

C Code

```
double randfn(float fnum, float lnum){
      //(rand()/RAND_MAX) return between 0 and 1
      return (double)fnum+((double)(lnum-
              fnum)*(double)rand()/RAND_MAX);
}
```

- Timer routine

- Timer routine uses to get the seed value to initialize the random number.

- Timer routine returns a number of hundred-second from midnight until now.

C Code

```
long timer(){
      //get current system time
      struct time t;
      gettime(&t);
      //return number of hundred of second
return(((long)3600*t.ti_hour+60*t.ti_min+t.ti_sec)*100+t.ti_hund);
}
```

- Round routine

- Round routine uses to define the digit after decimal place of floating point number.

C Code

```
double round(double num, int r)
{     long n, m,md;
      m=(long)pow(10,r+1);    //value of 10**(r+1)
      n=num*m;          //increase value of num to n
      md=n%10;          //find the last digit
      if(md>=5) n=n/10+1;     //if last diit>5-=>round up
      else n=n/10;            //else round down

      return (double)n/pow(10,r);
}
```

Example: round(19.256,2)=19.26; round(28.393,1)=28.4

☞ **Usage:** Assume that we set the input parameters already

Pseudo-code

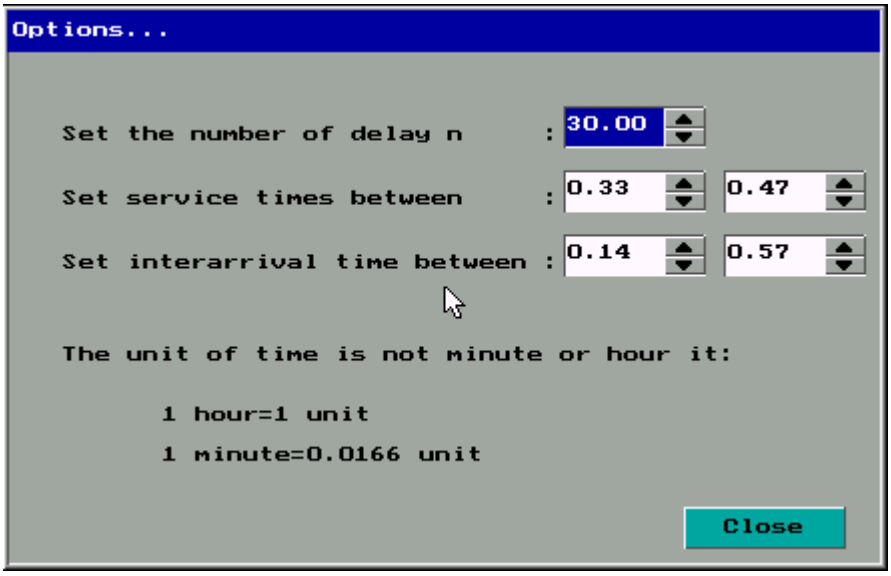
```
- call initialize function
- while number of delay<number customer
      call event_occur() routine
  end
```

C Code

```
void simulate(){
initialize_sm();
while(num_delay<num_customer)
       { event_occur(); }
}
```

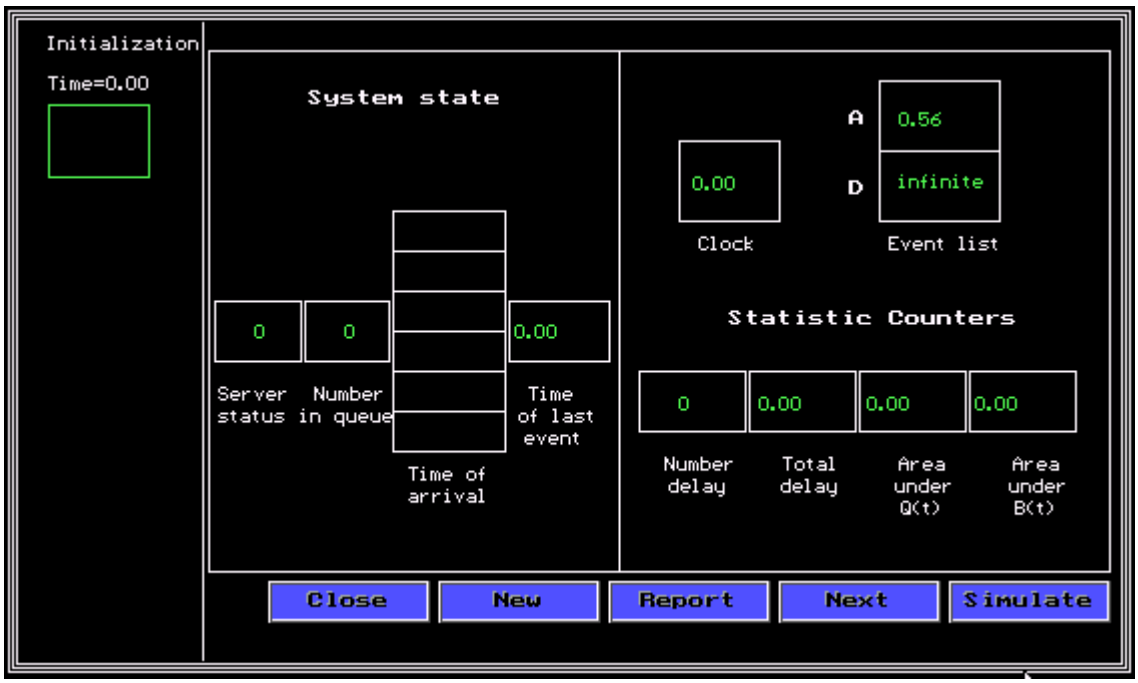
VI- Program Usage:

- Click <**Option**> button to set the input parameters



In each box, we can use <**UP ARROW**> and <**DOWN ARROW**> key to increase or decrease the number. Use the <**TAB**> key to change the box. The time is not minute or hour it is a unit time that **1 unit=1 hour, but 1 minute=0.0166 unit time**. After we define the input parameters already, click **close** button to close dialog box.

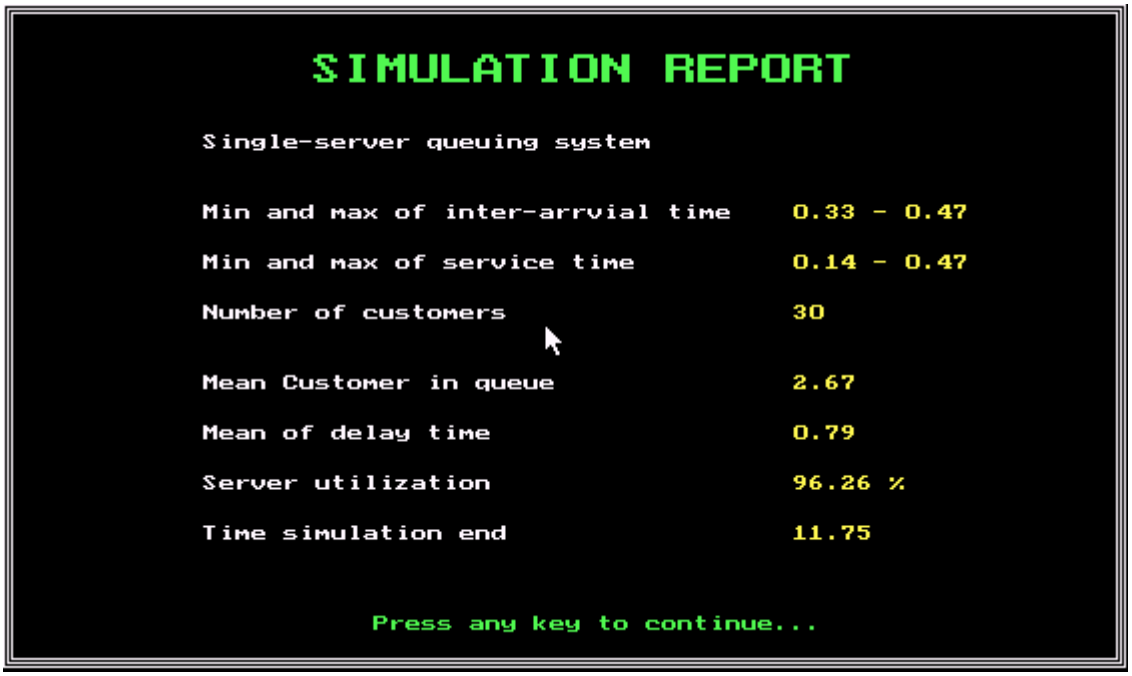
- Click <**Start**> button to start the simulation



Click on button **simulate** to run the simulation to the end of simulate. Then click button **next** or **Report** to show the report. Anyway, we can click on button **Report** to show the report once.

- Simulate Report

- Click on button <**Report**> to simulate and show the report.
- Click <**Simulate**> and click <**Next**> button to show the report.



VII- Conclusion:

In conclusion, we give the input parameters to run the program to get result. The input parameters are mean of service time = 0.33 to 0.47 unit time and mean of inter-arrival time = 0.14 to 0.57 unit time (1 minute = 0.0166 unit time). We just change the number of customers, and run the program many times. The result shown below:

No.	Number of customer	Server utilization	Mean of customer in queue	Mean of delay time
1	30	97.96 %	0.97	0.35
2	25	87.44 %	0.37	0.16
3	24	95.75 %	3.09	1.05
4	28	96.35 %	0.72	0.27
5	27	96.67 %	2.66	0.96
6	25	94.75 %	0.85	0.28
7	26	94.62 %	1.95	0.71
8	24	94.98 %	1.76	0.65
9	29	95.76 %	3.35	1.08
10	22	96.01 %	0.77	0.30
11	30	98.17 %	1.67	0.61
12	25	96.43 %	2.09	0.81
13	15	92.42 %	1.19	0.41
14	20	96.95 %	1.27	0.47
15	26	93.28 %	1.82	0.67
Avg	**25**	**95.17 %**	**1.64**	**0.59**

As the result, we simulate on the barbershop with a single server by 25 customers. The percentage that server busy is 95.17 % during the shop opening. Additionally, there are two customers waiting in queue per unit time or per hour before they start service, each customer have delay in queue more than a half of hour, and the service time to serve for each customer almost less than a half of hour. Therefore, we recommend that, the barbershop should add one more server to gain more customers. If not, that barbershop will lose some customers.