ECE 358 Networking Lab 1

# Question 1

The equation of PDF for the exponential random variable is the following.

In CDF, the equation changes to the following.

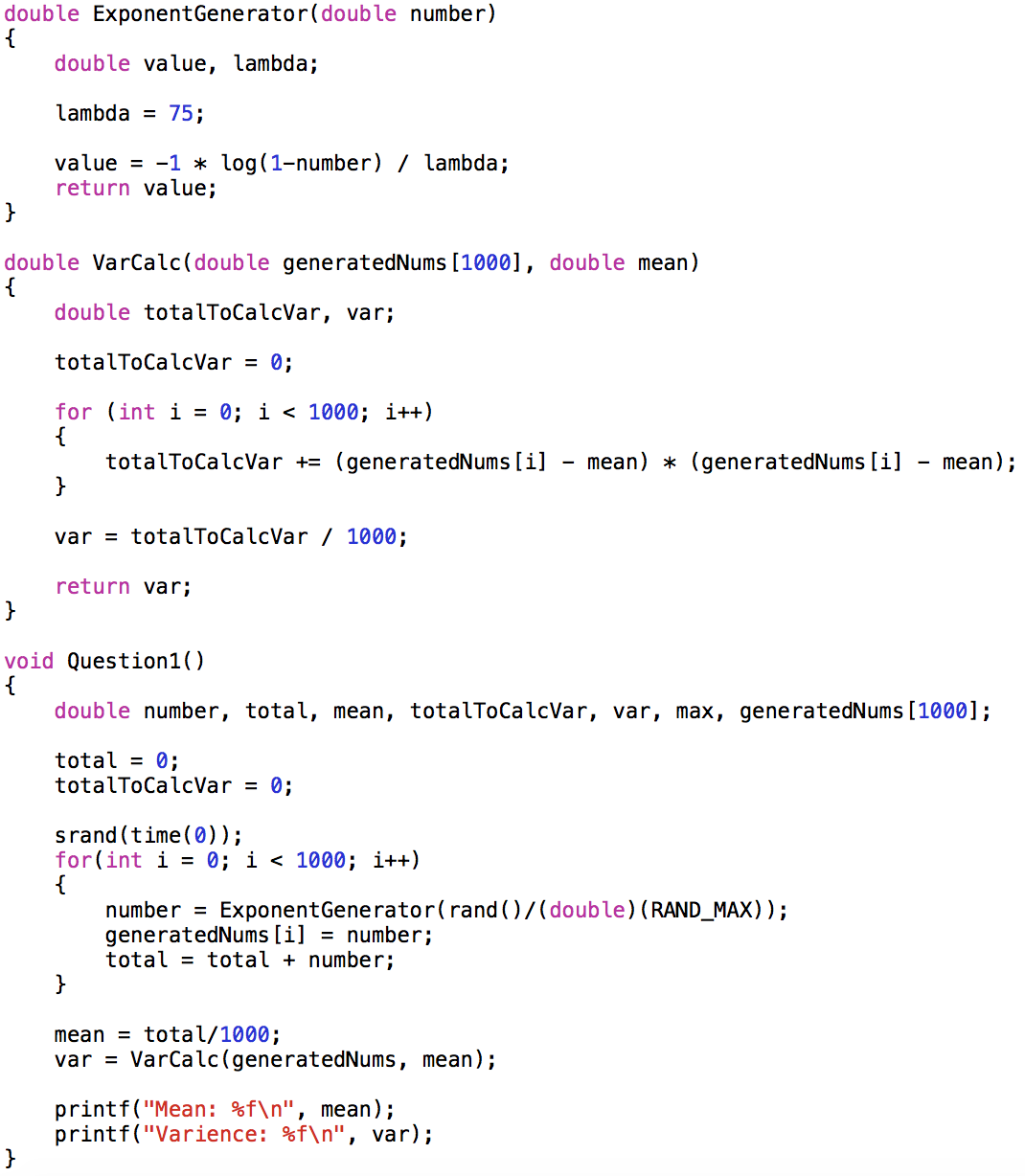


Figure 1. Exponential Generator



Figure 2. Mean and Variance of the Generator

I agree with the mean and variance. The expected mean of the exponential generator with is , which is 0.01333. The percent error is 2.82%. The expected variance of the exponential generator with is , which is . The percent error is 1.8123%.

# Question 2

enum EventType {a, d, o};

struct event

{

EventType TypeOfPacket;

double GenerationTime;

bool operator<(const event& rhs) const

{

return GenerationTime > rhs.GenerationTime;

}

bool operator>(const event& rhs) const

{

return GenerationTime < rhs.GenerationTime;

}

};

class q2

{

public:

q2 (int numberOfPackets);

void GenerateArrivalPackects(), GenerateObserverPackets(), GenerateDeparturePackets(), SortDES();

long Np(), No(), Ni();

~q2();

private:

double ExponentialRandomGenerator(double x);

long \_Na, \_No, \_Nd, \_Ni, \_Np, \_L, \_C, \_Time;

double \_Row, \_Lambda;

priority\_queue<event> \_DES;

};

q2::q2(int timeElapsed)

{

\_Na = 0;

\_Nd = 0;

\_No = 0;

\_Ni = 0;

\_Np = 0;

\_L = 12000;

\_C = 1000000;

\_Row = 20;

\_Lambda = \_Row \* (double)\_C / (double)\_L;

\_Time = timeElapsed;

srand(time(NULL));

}

void q2::GenerateArrivalPackects()

{

cout << "Beginning of Generation of Arrival Packets" << endl;

event arrival;

double ta = 0.0;

while (ta < \_Time)

{

ta += ExponentialRandomGenerator(\_Lambda);

arrival.TypeOfPacket = a;

arrival.GenerationTime = ta;

\_DES.push(arrival);

}

}

void q2::GenerateObserverPackets()

{

cout << "Beginning of Observer Packets" << endl;

event observer;

double alpha, to;

alpha = 3\*\_Lambda;

to = 0.0;

while (to < \_Time)

{

to += ExponentialRandomGenerator(alpha);

observer.TypeOfPacket = o;

observer.GenerationTime = to;

\_DES.push(observer);

}

}

void q2::SortDES()

{

cout << "Beginnig of SortDES" << endl;

event departure;

double packetLength, serviceTime, td;

td = 0.0;

while (!\_DES.empty())

{

if (\_DES.top().TypeOfPacket == a)

{

packetLength = ExponentialRandomGenerator(1.0/\_L);

serviceTime = packetLength/(double)\_C;

if (\_Na - \_Nd == 0)

{

td = \_DES.top().GenerationTime + serviceTime;

}

else

{

td += serviceTime;

}

departure.TypeOfPacket = d;

departure.GenerationTime = td;

\_DES.push(departure);

\_Na++;

}

else if (\_DES.top().TypeOfPacket == d)

{

\_Nd++;

}

else if (\_DES.top().TypeOfPacket == o)

{

\_No++;

\_Np += \_Na - \_Nd;

if (\_Na - \_Nd == 0)

{

\_Ni++;

}

}

\_DES.pop();

}

}

int main()

{

int timeElapsed;

timeElapsed = 10000;

q2 \* question2;

question2 = new q2(timeElapsed);

question2->GenerateArrivalPackects();

question2->GenerateObserverPackets();

question2->SortDES();

cout << "Average # of packets: " << (double)(question2 -> Np())/(double)(question2 -> No()) << endl;

cout << "Idle Probability: " << (double)(question2 -> Ni())/(double)(question2 -> No()) << endl;

question2 = NULL;

delete question2;

return 0;

}

First, I defined the struct of the event. In the event, there is enum, which defines the event time, and generation time, which specifies the timing the event, occurred. Then in the main function, the q2 class is initiated. Many variables are defined. They are

* \_Na to keep track of the number of arrival events,
* \_Nd to keep track of the number of departure events,
* \_No to keep track of the number of observer events,
* \_Ni to keep track of the number of idle moment,
* \_Np to keep track of the number of packets in the system counter,
* \_L, \_C, \_Row, \_Lambda constants, and
* \_Time, which will be time elapsed when running the program.

Times are declared in each function. Examples are

* ta for arrival event time,
* to for observer event time, and
* td for departure event time.

Then arrival packets are generated. After that, observer packets are generated. One thing to note is that for every arrival event, approximately 3 more observing event is generated. Then, through SortDES subroutine, departure packet is added and the records are kept for each observing event as the event in DES priority\_queue. Lastly, in main function, average number of packets in the system and the idle probability are calculated.

# Question 3

|  |  |  |
| --- | --- | --- |
| Ρ | Average # of Packets | Idle Probability |
| 0.25 | 0.331913 | 0.750841 |
| 0.35 | 0.532471 | 0.652414 |
| 0.45 | 0.818407 | 0.549584 |
| 0.55 | 1.2292 | 0.448846 |
| 0.65 | 1.86127 | 0.349281 |
| 0.75 | 2.99512 | 0.248603 |
| 0.85 | 5.6238 | 0.15201 |
| 0.95 | 19.3687 | 0.0498583 |

For each p value, I changed \_row and compiled the program. Then I copied the values into the table.

The results make sense. As rho increases, more events are produced. It means that more packets are in the system. Therefore, the system will be busier processing the packets. So the idle probability goes down and more and more packets wait for their turn to be processed since the system’s processing power does not increase. The processing time is constant, where as more events are generated.

# Question 4

If rho = 1.2, the idle probability is 9.99553e-07 and the average number of packets in the system is 83769.3. This indicates that the system is at its limit that the system is always busy and many of the packets are stored in the buffer. If the buffer is limited, many packets will be dropped.