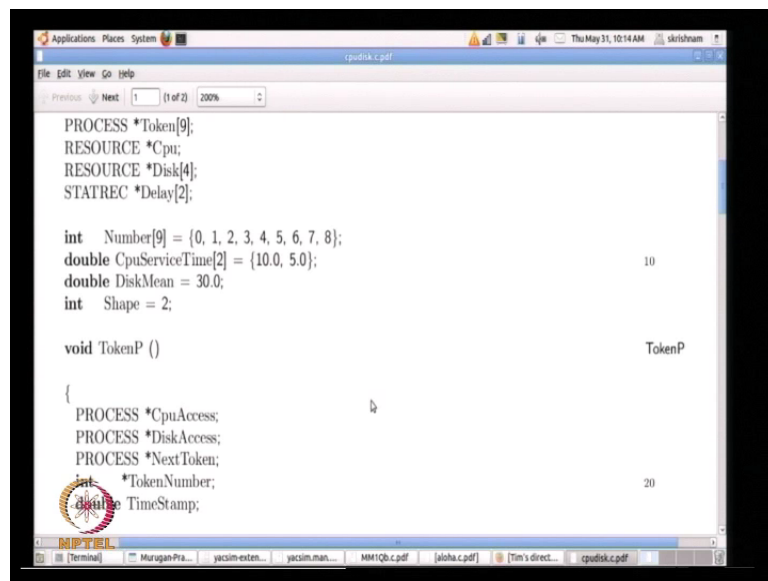


**Performance Evaluation of Computer Systems**  
**Prof. Krishna Moorthy Sivalingam**  
**Department of Computer Science and Engineering**  
**Indian institute of Technology, Madras**

**Lecture No. # 38**  
**Discrete-Event Simulations - III**

(Refer Slide Time: 00:20)



```
PROCESS *Token[9];
RESOURCE *Cpu;
RESOURCE *Disk[4];
STATREC *Delay[2];

int Number[9] = {0, 1, 2, 3, 4, 5, 6, 7, 8};
double CpuServiceTime[2] = {10.0, 5.0};
double DiskMean = 30.0;
int Shape = 2;

void TokenP ()

{
    PROCESS *CpuAccess;
    PROCESS *DiskAccess;
    PROCESS *NextToken;
    int *TokenNumber;
    TimeStamp;
}
```

Good morning, welcome back. So, we will come back to our action based discrete simulation programs. So, we saw two examples yesterday. The MM1 q system as well as the Aloha protocol, as this is the two basic examples that we saw. With MM1, we saw the infinite buffer, how use the resource object it is available in the axim, and with Aloha we showed how we can access around buffers to add packets, to delete packets and then process the packets based on that, both those buffers.

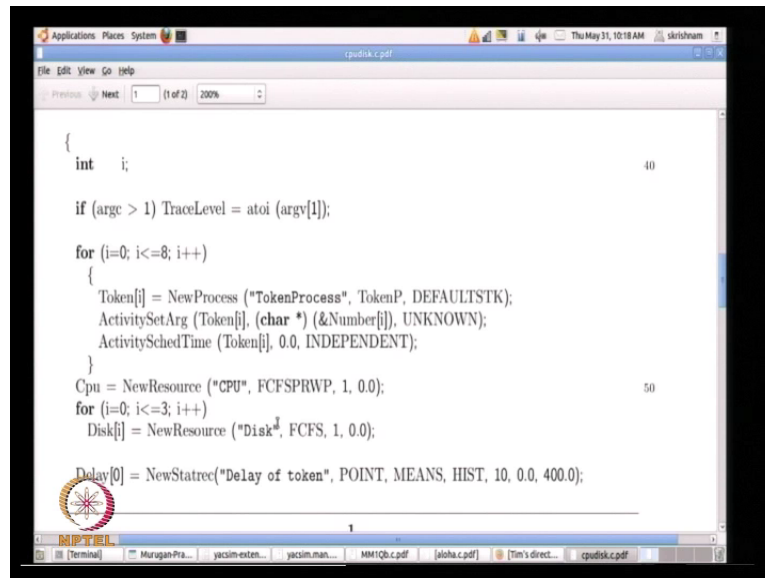
So, now will look at this set of examples; you have been looking at most of the course, where there is a job, where is set of jobs that are sequencing to the CPU and then taking or requesting service from one or more disks and then coming back to this CPU. So, your typical pros processing execution right, unique process, any system process for that matter right. Runs on the CPU, request some sort up input and output, then repeat is loop endlessly, finishes this system right, and this example today we will look at the close

queuing system, where there is a fixed number of jobs and circulation and we are seeing this sharing system. So, what we want to look at is the throughput out of the system and terms of... So, the number of jobs is fixed. Say, 10 or and then based on that, as this job sequence to this CPU and disk; what is effective throughput of the system? In this sense how many jobs per second can be completed? So, we have looked at techniques, mean value analysis and related system right, to understand how this is this can be measured, all this different performance metrics. But we have always we need a way to compare the results that may need either of the techniques provide us, and simulation is a good way of validity in this certificate from the theoretical analysis.

So, in this example we have 9 processes that are 9 jobs, as per the other terminology which is called as token; that which says as the process token right. So, this is 9 processes that are running in the system and this is single CPU that is time shared between these 4 processes or 9 processes sorry. And then there is set of disks, there are 4 disks in the system right which are shared among this 4 processes, 9 processes. Say, the processes can request the CPU then, 1 of these 4 disks and then repeat this forever. That is the basic queue; a close queuing network that you have seen before. The jobs are of two types; this is only for, just show how we can actually make something more flexible and more detailed in this simulation right.

So, we can see that the two types of jobs, one which have 10 units of time of CPU execution; other is 5 units of CPU execution. So, there are two classes of job that we have in this system. And the time, that you request the disk right, the amount of the average service time on each disk is defined to be 30 units right, 30 time units. So, that is our basic system that we have here. And now, let us look at how we create the set of processes that are needed to get this simulation going right.

(Refer Slide Time: 03:21)



```
Applications Places System cputisk.c.pdf Thu May 31, 10:18 AM skirsham
File Edit View Go Help
Previous Next 1 (1 of 2) 200%
{
  int i; 40

  if (argc > 1) TraceLevel = atoi (argv[1]);

  for (i=0; i<=8; i++)
  {
    Token[i] = NewProcess ("TokenProcess", TokenP, DEFAULTSTK);
    ActivitySetArg (Token[i], (char *) (&Number[i]), UNKNOWN);
    ActivitySchedTime (Token[i], 0.0, INDEPENDENT);
  }
  Cpu = NewResource ("CPU", FCFSRWP, 1, 0.0); 50
  for (i=0; i<=3; i++)
    Disk[i] = NewResource ("Disk", FCFS, 1, 0.0);

  Delay[0] = NewStatrec("Delay of token", POINT, MEANS, HIST, 10, 0.0, 400.0);
}
```

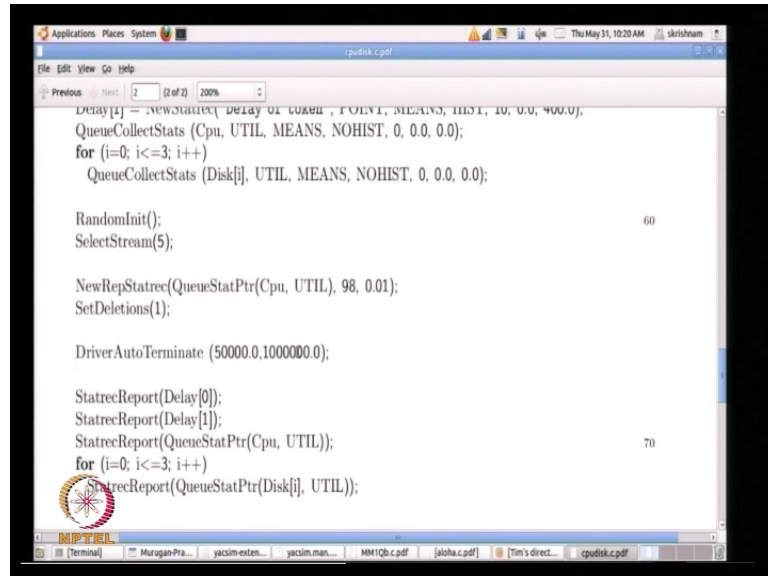
So, if we look at user main, so there we will create first the set of processes and the corresponding set of resources that these processes are going to share. So, this is the first loop. So, there are 9 processes. So, we create 9 processes; 1 for which with a corresponding ID, we use a activity set or to let each process know what its ID is; because that is needed later on a time. And then, we schedule this process to start right away. So, there are 9 parallel processes running in the system. Each is going to execute this token P, is the function that is going to be executed by each of those 9 processes.

So, that is now we established. So, we are now created 9 parallel processes for execution. Now we create, we use this new resource, the resource object that is provided by action to create the CPU, which is again simple, because we know that you simply request the CPU. And the CPU is busy then, simply you have to wait for the CPU to become free again. So, therefore, there is only one CPU and this is the first come, first serve with priorities. The one of the different scheduling algorithms that is, disciplines by default available action. So, we have now created this CPU, the second step.

Then, we go and create the disk. The disk is very simple; first come, first serve disk. That is nothing special about it. There are 4 disks and each disk has corresponding resource that is allocated for that. And now there are two types of statistics records; one for the short jobs and one for the long jobs that is alright. So, that we have delay 0 and delay 1. So, all the short CPU processes will go and update this short statistics required, short

CPU and other one the update the other statistics required. We also measured the CPU utilization. That is, we collect statistics.

(Refer Slide Time: 05:03)



```
Delay[i] = newStatree(DelayOfCpu, DelayOfDisk, MEANS, HIST, 10, 0.0, 400.0);
QueueCollectStats (Cpu, UTIL, MEANS, NOHIST, 0, 0.0, 0.0);
for (i=0; i<=3; i++)
    QueueCollectStats (Disk[i], UTIL, MEANS, NOHIST, 0, 0.0, 0.0);

RandomInit();
SelectStream(5);

NewRepStatree(QueueStatPtr(Cpu, UTIL), 98, 0.01);
SetDeletions(1);

DriverAutoTerminate (50000.0,1000000.0);

StatreeReport(Delay[0]);
StatreeReport(Delay[1]);
StatreeReport(QueueStatPtr(Cpu, UTIL));
for (i=0; i<=3; i++)
    StatreeReport(QueueStatPtr(Disk[i], UTIL));
```

We do not do that, then, this system will not collect statistics. We need to in work this. We also collect statistics in terms of utilization for the 4 disks in the system. So, that is are basic system done. Then we have this random minute and then notion of select stream is mini simulated provided with default set of random number streams. Usually, random numbers, the way is work you have a starting numbers, starting seed and then after that it is deterministic. All the subsequent random numbers in the same seed are going to follow this same pattern. It is usually some set of mode operations. And then, you have repeated sequence.

After a long time, this sequence will repeat for any given stream. And because we want to have some predictability in the output, rather than use some arbitrary random number like time or some other entropy of the system; we try to have a set of say 16 random number streams and we know that, but we want have at least 16 different combinations. So, what will normally do is, run the same simulation for 10 or more different random streams and then look at the main, the conference intervals based on that right. Here, select stream simply means select the fifth random stream. It does not matter what the stream is; we just want to you use a particular random stream and there is a again one more statistic point which a mode skip for now right.

So, then we have this driver auto terminate function which is the one that we talked about in the last class were you, run up to 1 million time units if necessary. If convergence takes place, based on the delay, then you simply stop the system from simulation at that point of time. So that, once driver auto terminates finishes automatically we know that, this is simulation is complete. Then we report all the statistics for the 0 delay 1, the CPU utilization and for the dis-utilization. So, this is our performance metric analysis. So, this is the basic system. Now, the 4 disks and the CPU are in place and the 9 processes have started execution right. So, now, we need to go and create the code for the token P the function that is going to be executed by each CPU process right job and execution.

(Refer Slide Time: 06:54)

```

{
PROCESS *CpuAccess;
PROCESS *DiskAccess;
PROCESS *NextToken;
int *TokenNumber;
double TimeStamp;

TokenNumber = (int *) ActivityGetArg(ME);
ProcessSetPriority (ME, (*TokenNumber>5)?1.0:0.0);
do
{
TimeStamp = GetSimTime();
ResourceUse(Cpu, Exponential(CpuServiceTime[*TokenNumber>5]));
ResourceUse(Disk[UniformInt(0,3)], Erlang(DiskMean, Shape));
StatrecUpdate(Delay[*TokenNumber>5], (GetSimTime()-TimeStamp), 1.0);
while (1 > 0);
}

```

So, thus, this process again as were saying before is going to be a endless right which of endless while loop or a do while loop. In this case, that simply executes forever and whenever this system simulation terminates; automatically this will just get terminated right. So, anyway, so this again is to get a (( )) every process, you need to find out which when this token P is executing. You need to know what is the corresponding process number and that is what is obtained from this activity get all, and then you set the priority based on the cores, corresponding token number. And then if you look at the actual code it is simply using the CPU for some exponential amount of service time right. And again here if you look at it right for the token number is greater than 5 then, you to have, array here right to CPU service time in array of 2 elements. If the token

number is greater than 5, then you will have mean service time of 5 units. If it is less than 5, mean service time is 10 units that is all. So, this is how your request the CPU. CPU is busy, you have to wait until the CPU gets allocated.

So, then resource use is slightly different from what we saw before right. Previously, we had a process which used activity scheduled resource. Here this simply says use the resource. So, resource use will automatically queue the process when the process is completed by the CPU, then it goes on to the next statement. So, this is actually a block statement. So, you have to understand that right. This is this will block this particular process until it gets selected by the CPU. So, there is a lot of fine detail which is conveniently hidden by action, but for us logically it simply, use the CPU, use the disk right. So, here again, I randomly choose 1 of the 4 disks and this is the service time; is based on the Erlang distribution right which is something again predefined in the axiom service. Let us We can look at the later if necessary, but instead of exponential service time, it is now some Erlang service time based on this 2 parameter. This is the main and if you look at Erlang there is also shape parameter right. This is A, B are the 2 parameters. Main is 1 and the shape parameter tells you how what will be the service time. So, random variable defined by this Erlang distribution.

So, then resource you use for the CPU, resource use for the disk and then finally, when we are done. So, I have used this CPU, I have used the disk and then I have simply updated the delay statistics required right based on the time that before the process, the current process executes forever. For in the current run, right for the current CPU dis-quantum, I get the time before that particular run started. When the run finishes, this is amount of time that a run of the disk on the CPU. And then, technically if you look at this is probably some other job in the system, but just a same job in our simulation. We are conveniently assuming it is a same job. In theory, it could be some other job right. It is like 1 job finishes and some other job replaces that. Only difference is these 2 jobs had a same parameter in terms of CPU execution time and disks services time. Therefore, it is right conveniently replaced. That is why we consider the close queue in system. It is how we do multi programming right.

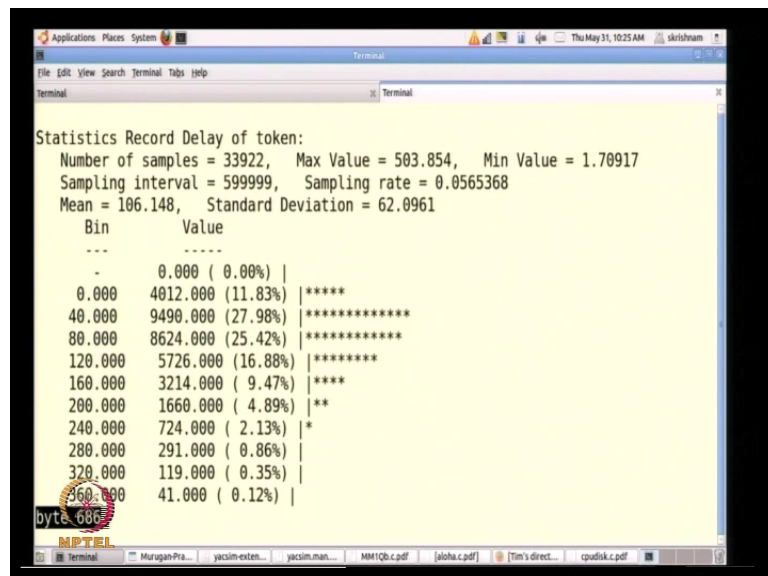
So, we want to look at, if I run 10 jobs in the system what is effective throughput of the system; if I run 5 job in the system, what is throughput of the system. And you want to plot this graph in terms of number of jobs in the system verses the effective throughput

of the system right; as a number of job is very small, throughput will be small. But will increase, the number of jobs to very large number, then what will happen is queuing will happen right lots of queuing will build up; which means throughput will come down. So, you want find out the optimal system operating point in terms of the number of jobs. It can be active in the system. That is the degree of multi programming as far as if you see that in over text books.

So, that is all and. So, this is the basic definition for the system. So, all the analysis will (C) right. We can conveniently compare when m B a. In m b a, we did have a notion of 2 difference service times. You assume that all that service times are same on the CPU here. I have this slide modification in terms of two difference service times depending on the process itself.

So, now, let us see, we can actually get this to compile, not compile will simply run it, there is a compile code that is available, so CPU disk, we do not have to define any specific parameters right because everything is predefined. This number of processes, service time all at is defined, and this is finally, what we get in the terms of the system output right.

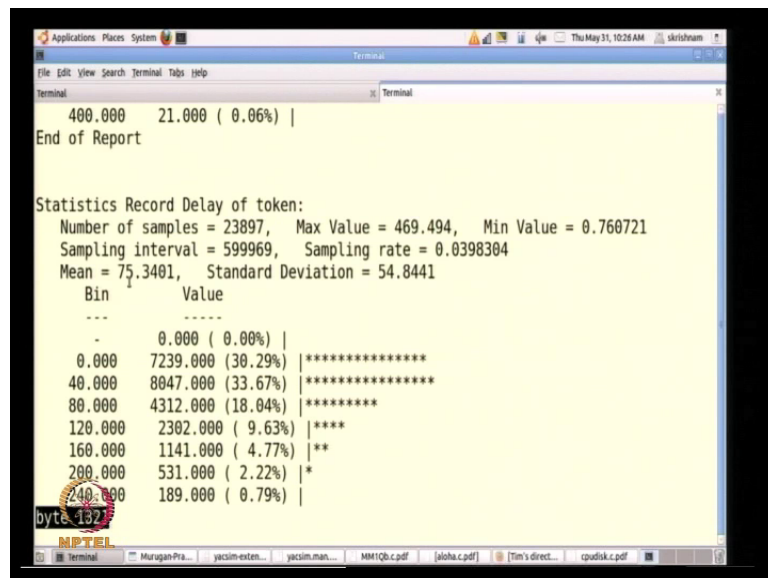
(Refer Slide Time: 11:15)



So, what we, so this is the we simply output. This is the histogram right of the different delays. So, this something that, the axiom also computes for you. It says that, between 0 and 40 right eleven percent of the process delays for the each process was eleven and

between 0 and 40 time units and 27 percent was in this time units and so on right. This gives you the distribution right in terms of the number of samples that for with in this different histogram. That is something in little more detail that, simply the mean value, right the mean value we know it is simply 106 units of time, but the distribution is given by this particular system. So, there are some jobs that take up to 360 units of time. So, the mean is only representative right. You find that, there are very small numbers of jobs that actually take 360. So, fairly long tail and, but about 25 percent of the jobs take right somewhere between 80 and 40 units of time and so on. That is the histogram for this is for the that is for longer job with service average service time of 10 units on the CPU, so 106 units of time.

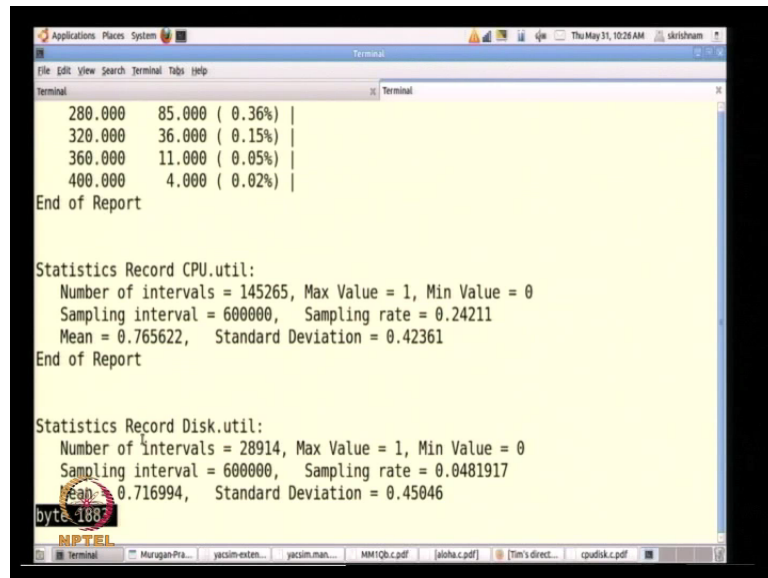
(Refer Slide Time: 12:18)



For the shorter job, it is 75 units of time is the main that is what we observe here right, and look at the number of samples and sampling interval and if you look at the max value, in the worst case it is 470 and some cases it is because is exponential distributed service time right. So, we get that point 76 is the minimum and maximum is about 470.



(Refer Slide Time: 12:37)



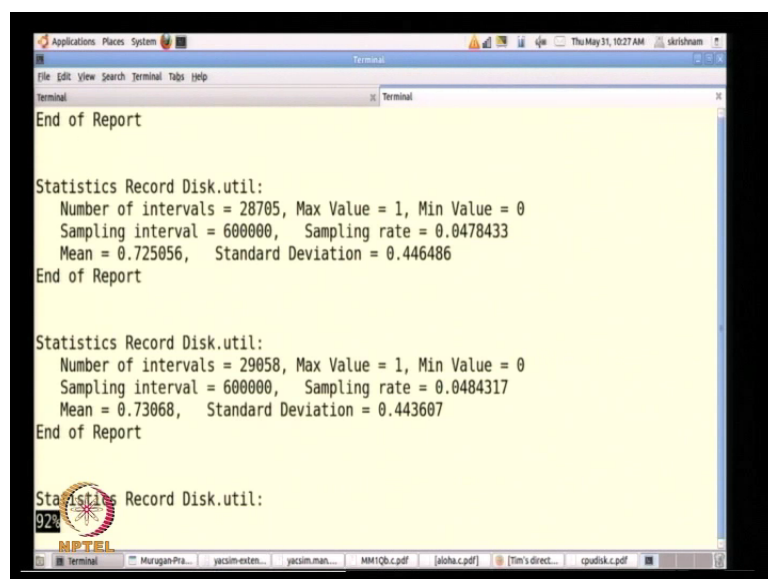
```
280.000 85.000 ( 0.36%) |
320.000 36.000 ( 0.15%) |
360.000 11.000 ( 0.05%) |
400.000 4.000 ( 0.02%) |
End of Report

Statistics Record CPU.util:
Number of intervals = 145265, Max Value = 1, Min Value = 0
Sampling interval = 600000, Sampling rate = 0.24211
Mean = 0.765622, Standard Deviation = 0.42361
End of Report

Statistics Record Disk.util:
Number of intervals = 28914, Max Value = 1, Min Value = 0
Sampling interval = 600000, Sampling rate = 0.0481917
Mean = 0.716994, Standard Deviation = 0.45046
```

Then, we have a utilization of the CPU. So, this CPU utilization is 0.765, if you remember m B a calculation, that is what we are trying to find right, utilization of the CPU, so that 0.76 and disk utilization. This is the first disk right, disk number 1 disk number 2 and so on. So, this is disk utilization are about 0.72, 0.72, 0.273, 0.7. So, there are all roughly the same because we are uniformly, randomly selecting any one of this 4 disk for the system operation.

(Refer Slide Time: 13:05)



```
End of Report

Statistics Record Disk.util:
Number of intervals = 28705, Max Value = 1, Min Value = 0
Sampling interval = 600000, Sampling rate = 0.0478433
Mean = 0.725056, Standard Deviation = 0.446486
End of Report

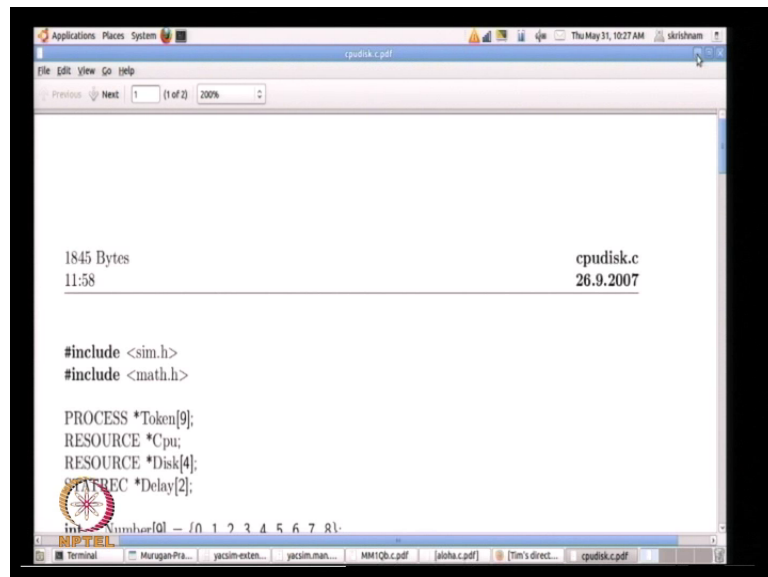
Statistics Record Disk.util:
Number of intervals = 29058, Max Value = 1, Min Value = 0
Sampling interval = 600000, Sampling rate = 0.0484317
Mean = 0.73068, Standard Deviation = 0.443607
End of Report

Statistics Record Disk.util:
92%
MPTEL
```

So, that is essentially what we have. This is a way you can actually compare your m B a results right and then validate it with help of this particular self –simulation. So, this can be expanded.

We can write more complicated programs right, with the help of this with the basic mechanism. So, all in all the basic queues are available. So, we can now construct larger and larger queuing networks. We only saw single queue before. Now, this is a network of queues right. So, I can actually feed packets from one queue to the other. So, that is the concluding part of the axiom that we wanted to talk about. So, question on this on the code analysis. No, questions, everything is clear?

(Refer Slide Time: 13:52)



```
1845 Bytes                                     cpudisk.c
11:58                                           26.9.2007

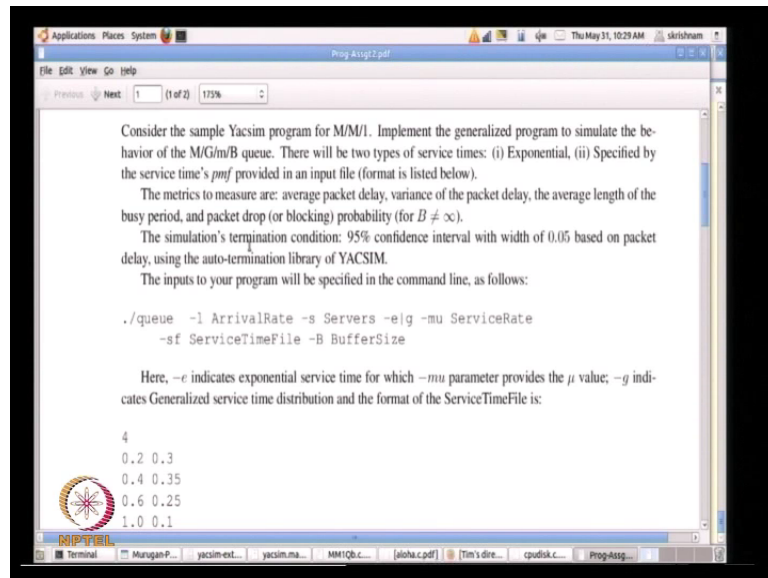
#include <sim.h>
#include <math.h>

PROCESS *Token[9];
RESOURCE *Cpu;
RESOURCE *Disk[4];
STATEEC *Delay[2];

int main() {
    int i;
    for (i = 0; i < 9; i++)
        Token[i] = new PROCESS("T", i, 1, 1);
    Cpu = new RESOURCE("C", 1, 1);
    for (i = 0; i < 4; i++)
        Disk[i] = new RESOURCE("D", i, 1, 1);
    Delay[0] = new STATEEC("D0", 1, 1);
    Delay[1] = new STATEEC("D1", 1, 1);
}
```

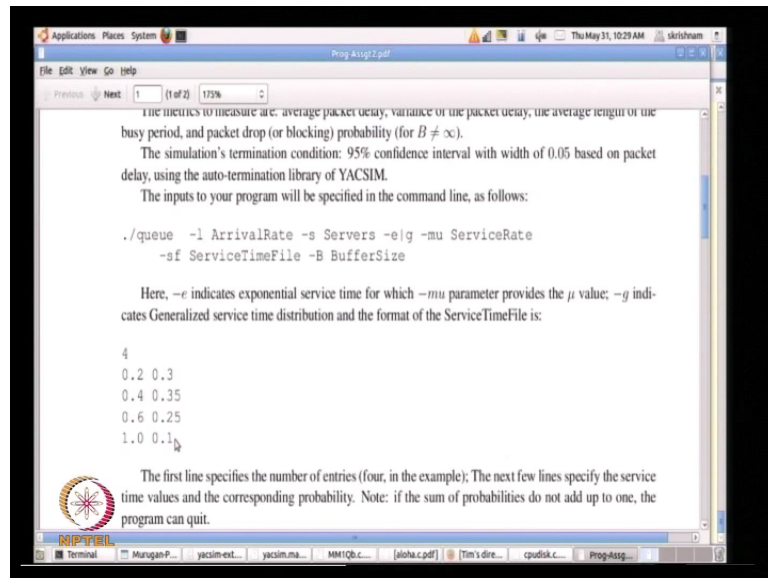
Now, we have seen right 3 sample programs and. So, if you are taking this class or going through these lectures on NPTEL, and then we also need to practice some of writing our own programs right. It is not just to look at code and look at samples and so you may be wanting write your own simulation program for whatever purpose, whatever be your application. You can write your own, but just to get you started there are couple of other programming assignments that we have developed.

(Refer Slide Time: 14:22)



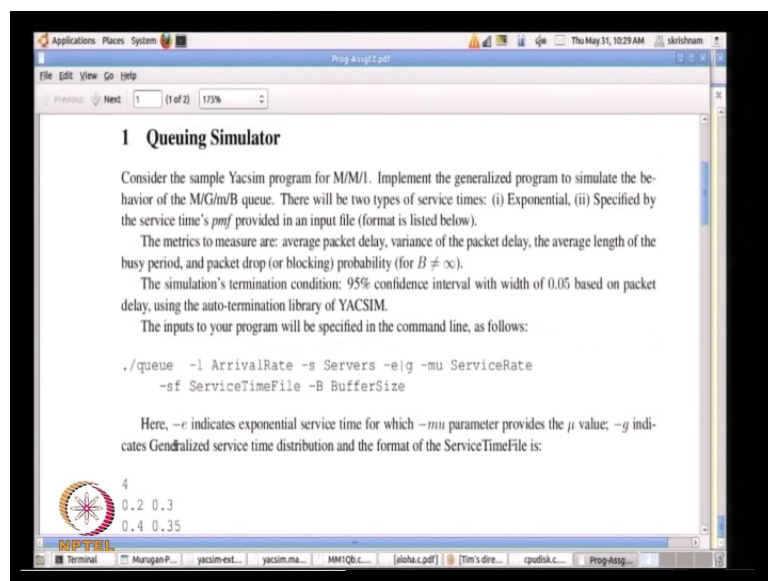
So, we will take a quick look at those. We saw the M M 1 queue sample program. So, we wanted to generalize this. So, you can take this session attempt right. This is part of the assignment that was given for the c a 6 2 1 course, but you can essentially try to generalize this to a M G m slash B queue, where the service time right can be either exponential or it can be specified by some set of B m f right. The service time can be actually given to an input rather than saying this service time is exponential. We can actually define the problem distribution. The problem demand function for this service times and then based on that you should be able to randomly generate for every packet that arrives, the customer that arrives. You will determine this service time based on this distribution.

(Refer Slide Time: 14:22)



So, for example, if you look at this system here, we are saying there are 4 different service times possible. So, with probability 0.3, which is the second parameter here the service will time will be point 2 units, probability 0.35 the service time is point 4 units, point 6 service time units and so on right. These are the all possible combination and if you look at it, all the probability should add up to one right. This is the P m f for the particular system. So, now we have to generate which is easy to do with help of the uniform random distribution all that is sort of left unspecified.

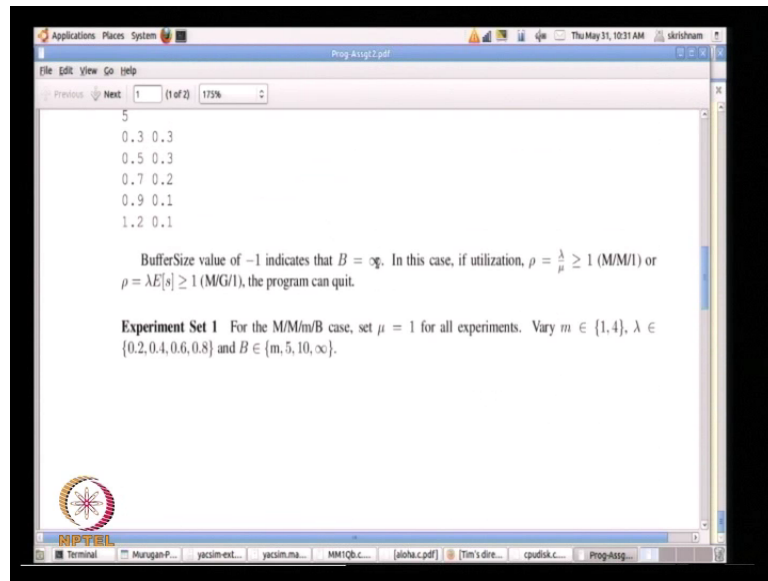
(Refer Slide Time: 15:41)



But, we can convert this MM1 program into very generic  $m/g/m/B$  program. So, to get in to B, you will have to implement your own servers to bring in the finite buffers we said, talked about; whether axiom provides finite buffers or not is not clear. But we can, very easily find out finite buffer capabilities and also multiple servers. Everything we can handle by a writing this things ourselves. And in terms of measurements, the metrics that we want to the measure from the system, average packet delay this is something that we can very easily measure. We have seen examples for that and also variance of the packet delay. It is now that you have all the samples you can easily compute the variance.

We can, if you look at this result from  $(C)$  right variances for available mm1, but if you want to do  $M/G/m/B$  then, variances are not going to be easily available as a closet formula. So, you have to closet formula. So, you have to restart something like simulation to actually get the service variance time in this particular. We can also measure the average length of busy period for the different servers, and also the packet drop probability. Because when the buffers size is infinite, there is no packet drop. When the buffer size is not infinite then, arriving packet will find the buffer full packets can get dropped and those thing again and those things we can measure easily with help of simulation. And likewise, if we look at mm1 B, there are results for the closed form system results for the block and probability right. That is easily available, but for a more generic system, it is not easily available. So, that way we need to resort to this simulation as we have seen in this the particular case.

(Refer Slide Time: 17:10)



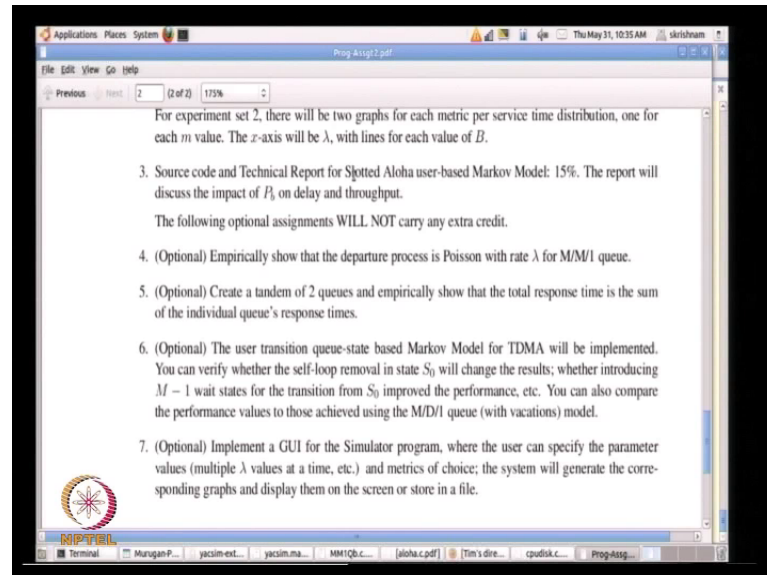
So, that is the basic set up and then there are right also itself variations in terms of different number of servers, different arrival rates for the system as well as different values for the buffer size.

So, these are all combinations that one can look at and there is also a slotted Aloha model that is available, that was discussed in class, in earlier lecture. And what we can do is, we can also try to, in part of that model, what we had the large transition matrix. So, we defined that as the ((C)) where there are different states for the system in terms of the number of packets that is being transmitted, number of packets in buffer, in back of in and so on.

So, we had a fairly law complex state transition probability matrix. And so what did you do with the state run transition probability matrix? This is a way of, you can actually code that in, codifying math lab or mathematic or lab are any other package right. And you can solve you can solve for that and get this steady state probabilities based on the transition matrix. That is also very straight forward exercise, and so once we do that. So, once we define the number of users in the system, the back of transition probability, the new packet transition probability right, you can simply generate the transition matrix based on those values. And then we can, if you remember that this from ((C)) textbook that was discussed in the class. So, this is the way of trying to take a mathematical model and actually getting numerical results for that right. Sometimes, numerical results have to

be a  $(\infty)$  at achieved by this means and anyway and as long as your matrix size is small, you should be able to get the results in a reasonable amount of time. Otherwise, you have to resort to some sort of approximation methods to get this.

(Refer Slide Time: 18:56)



This is the other experiment, that you could try, in order to right validate some of what you seen in terms of the theoretical model, so we have looked at the extension from mm1 to  $m g m$  slash  $B$  as well as that theoretical slotted Aloha model. That we saw can also be verified numerically with help of implementing this equation in Mat lab. So, that is 1 exercise that you can try out, as lot of other optional exercise also, which they have listed here. I will talk about it right 1 or 2 of them. In class we had seen that, the departure process from mm1 queue is also poison right. The arrival processes is poison that is an input with rate  $\lambda$  and we proved in class, departure is also poison with rate  $\lambda$ . Now, how do you verify that help of simulation right.

So, what you have to do is, you have to record the departure process. Every time a process or a customer leaves the system you note the time, which we only right now record the delays. We should now also record the instant of in time when those customers are leaving the system right. So, you keep track of the inter departure times. So, you have a set samples, set of sample inter departure times. Now, we will have to map that and find out whether that actually follows the exponential process right, the inter departure time is exponential. Then we know that, the departure process is also exponential. So, if

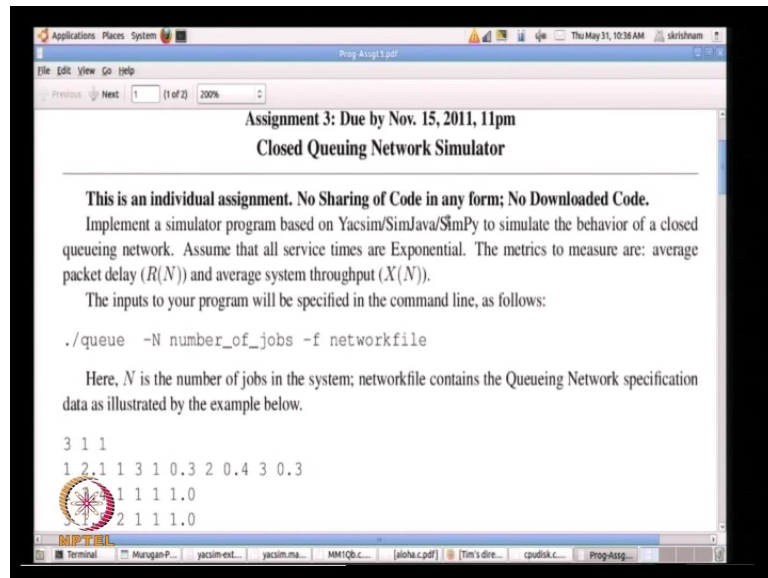
you look at Raj Jain's book, there is a way of looking at the quantile distributions and showing whether the sample values from the given system is matching an exponential distribution or not. So, that is an exercise that is fairly easy to figure out.

And then we had also seen in class that I can concatenate the random of 2 queues; both M M 1, where the first M M 1 queue finishes service, and feeds that to the second M M 1 queue. We saw that the delay in the system is the sum of the delay spent in each of the 2 queues, that is also that something that was the covered in class. Now, that is very easy to verify the simulation right. That was, we said that, there are 2 queues with arrival rate  $\lambda$  and  $\mu_1$  and  $\mu_2$  as the corresponding departure rates. Then the total delay in the system is,  $\frac{1}{\mu_1 - \lambda} + \frac{1}{\mu_2 - \lambda}$ .

So, you can verify that. So, we can verify that also, with the help of creating at the tandem of queues, where packets arrive to 1 queue when a packet departs the first queue, it is simply gets joint at the next queue in the system. So, that is the simple tandem of queues. It is a very simple example of queuing network. It is not real a network, but anyway there are 2 more than 1 k, therefore, technically it becomes a network. So, there are lots of other variation also that 1 can try out. So, only when you try these things, will you get comfortable with the action and then try to use those in your corresponding instruments of studies or project you are planning to use later on. So, that is one of the programming assignments that you could try out. Now, there is other one which you will discuss in about 5 minutes. Then we will start with it.

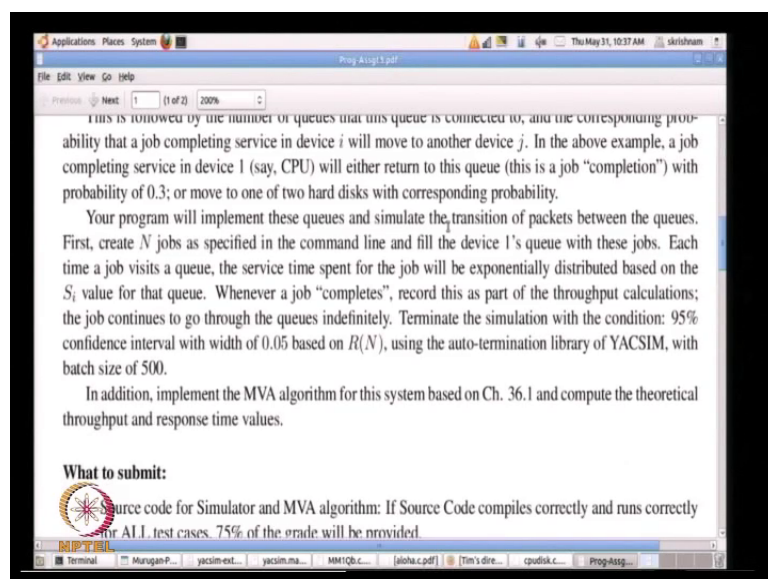


(Refer Slide Time: 21:38)



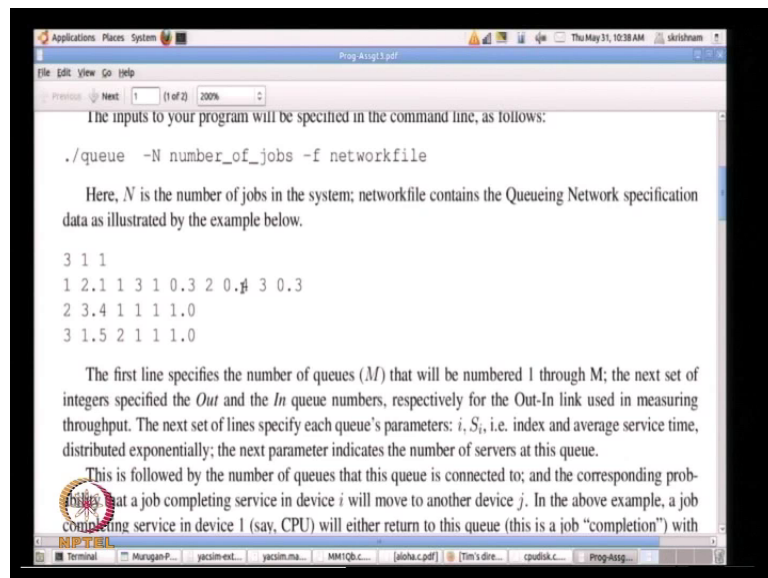
So, that was the open queuing right, open queuing system primarily. Now, we also defined in other program for creating a close queuing network. it any We have seen lot of examples in class like this CPU disk problem we saw earlier today. We can create more complex network right and we can again create a set of queues and connect those queues together, where the packets from 1 queue can reach some other queue based on some problem distribution and so on and that was this assignment is all about.

(Refer Slide Time: 22:22)



So, you will create a set of queues and simulate the transition of packets between these queues. When a packet finishes service in 1 queue, it automatically goes to 1 of the other queues in the system. And then, this again a close queuing system. So, we have a number of jobs right and jobs in the system is predefined. And then we want to find out throughput for the system right. So, we saw, we have seen  $x$  of  $n$   $r$  of  $n$  in class,  $x$  of  $n$  is the throughput;  $r$  of  $n$  is the corresponding response time. So, we can actually implement this as well as, compare right, these results with the  $m$   $b$   $a$  algorithm. So,  $m$   $b$   $a$  algorithm is defined, it is a very straight forward algorithm. It is a simply creative algorithm and where you start from  $n$  equals 1, 2, 3 and so on up to  $n$  equals 10. Whatever be the number of values that you have specified. So, you can compare your  $m$   $b$   $a$  algorithm, with your algorithm, with this particular algorithm and we can basically construct any network that you want to.

(Refer Slide Time: 23:15)



So, the specifications for the network creation are given in the example here right, where you specify the number of queues. And which is going to be the in queue which is going to be the out queue because in the case of close queuing system right, something (( )) in and the out queue 2 to find out in link and then we have passed that right.

So, we need to specify that for each queue in the system right, the corresponding parameters, in terms of the index for that, average service time right, the number of service of the queue and things like that. So, we can actually simulate not just single

server, you can simulate multiple servers at each queue. And then, right and then look at this system performance based on that. So, this is also little bit more complex, but it is doable and this combined with implementing m b a, will get a better will give a better understanding of the performance that you can get with this system right. Otherwise, it is all theory, on a paper and you would not really get the same value add or understanding that you will get from implementing this system. So that, you can use them for later, of your own exercises later on. So, that is the basic part about the discrete advance simulation that I want to talk about. So, any questions? No questions, so we will stop here for this part and then we re going to continue next on stochastic Petri nets which are Petri nets in general and followed by stochastic Petri nets.