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

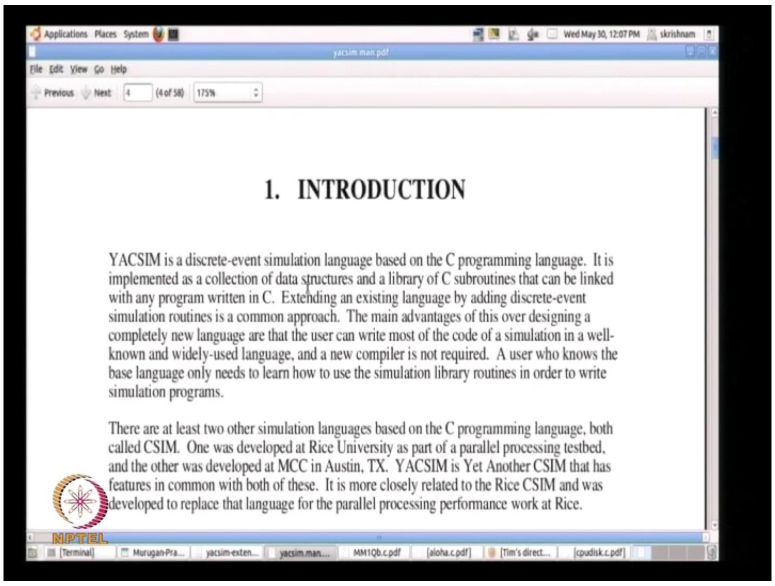
Lecture No. # 36
Programming Aspects of Discrete - Event Simulations-1

Today, we are going to focus on programming aspects of discrete event stimulation. We are used to looking at mathematics models before. And we also talked about what is discrete event stimulation is. So, there is are several software tools available that can help us they realize models, right. Models for event stimulation and then based on the two, performance evaluation and so on. This is a tool that goes back to, almost 20 years and it might be old, but it is still useful, it is basically a collection of C routines. And therefore, those who know C programming and C plus plus programming, can start queries easily from this.

There are other more sophisticated tool like, n s 2 and ns 3. More recently things like Omnet plus plus so on, particularly design for network stimulators, network stimulation. And there are tools for other domains too. But the reason, the axiom is particularly useful is, it is very generic, it gives you bare bones set of functions that you can use and with of the help of that we can consider you can construct fairly reasonably, good size stimulations. The advantage, we decide the abstraction you want. In the case of ns 2 and ns 3, the abstractions are not that easy. You have t c p layer and i p layer and so on.

With reaction, u can just exactly decide what part of the system u want abstract and build the model for that and realize that with the help of C code. So, we will go through is some other details of reaction manner. It is fairly fairly large manual, 60 pages or so also, but most of it is self understanding, self understandable. You can read once ones and figure of out things, but we want to highlights some of the main function right. So, the program itself, look at right, samples later on, that has a set of function that qualify a logic that we want to relies realize to capture the system behavior and action the axiom is still available, it still works and we can find it on net. If not, we will put not directly links for that Indian Institute Technology website which we have.

(Refer Slide Time: 02:14)



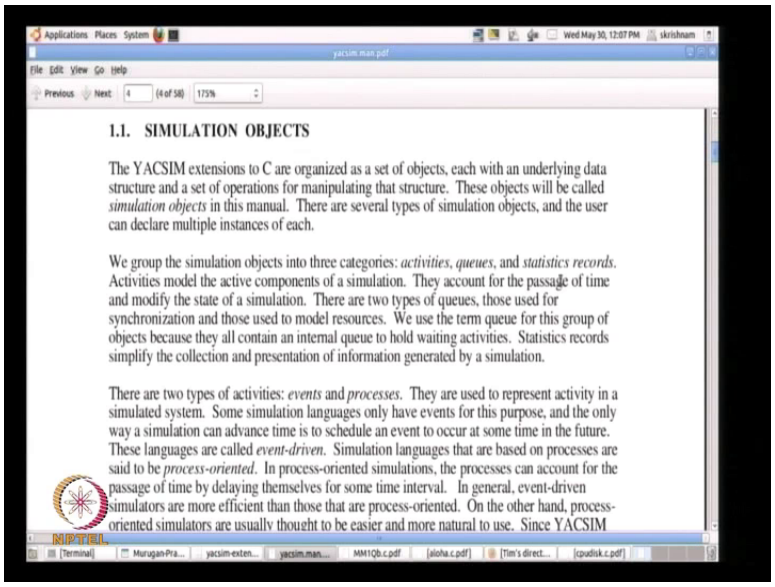
1. INTRODUCTION

YACSIM is a discrete-event simulation language based on the C programming language. It is implemented as a collection of data structures and a library of C subroutines that can be linked with any program written in C. Extending an existing language by adding discrete-event simulation routines is a common approach. The main advantages of this over designing a completely new language are that the user can write most of the code of a simulation in a well-known and widely-used language, and a new compiler is not required. A user who knows the base language only needs to learn how to use the simulation library routines in order to write simulation programs.

There are at least two other simulation languages based on the C programming language, both called CSIM. One was developed at Rice University as part of a parallel processing testbed, and the other was developed at MCC in Austin, TX. YACSIM is Yet Another CSIM that has features in common with both of these. It is more closely related to the Rice CSIM and was developed to replace that language for the parallel processing performance work at Rice.

So, again this is something which we already said right. So, this discrete describe even event stimulation language based on the C programming language. So, it is the collection of the data structures and Librans libraries and C of subroutines that can be linked to with any program written in C and link that in C plus plus.

(Refer Slide Time: 02:35)



1.1. SIMULATION OBJECTS

The YACSIM extensions to C are organized as a set of objects, each with an underlying data structure and a set of operations for manipulating that structure. These objects will be called *simulation objects* in this manual. There are several types of simulation objects, and the user can declare multiple instances of each.

We group the simulation objects into three categories: *activities*, *queues*, and *statistics records*. Activities model the active components of a simulation. They account for the passage of time and modify the state of a simulation. There are two types of queues, those used for synchronization and those used to model resources. We use the term queue for this group of objects because they all contain an internal queue to hold waiting activities. Statistics records simplify the collection and presentation of information generated by a simulation.

There are two types of activities: *events* and *processes*. They are used to represent activity in a simulated system. Some simulation languages only have events for this purpose, and the only way a simulation can advance time is to schedule an event to occur at some time in the future. These languages are called *event-driven*. Simulation languages that are based on processes are said to be *process-oriented*. In process-oriented simulations, the processes can account for the passage of time by delaying themselves for some time interval. In general, event-driven simulators are more efficient than those that are process-oriented. On the other hand, process-oriented simulators are usually thought to be easier and more natural to use. Since YACSIM

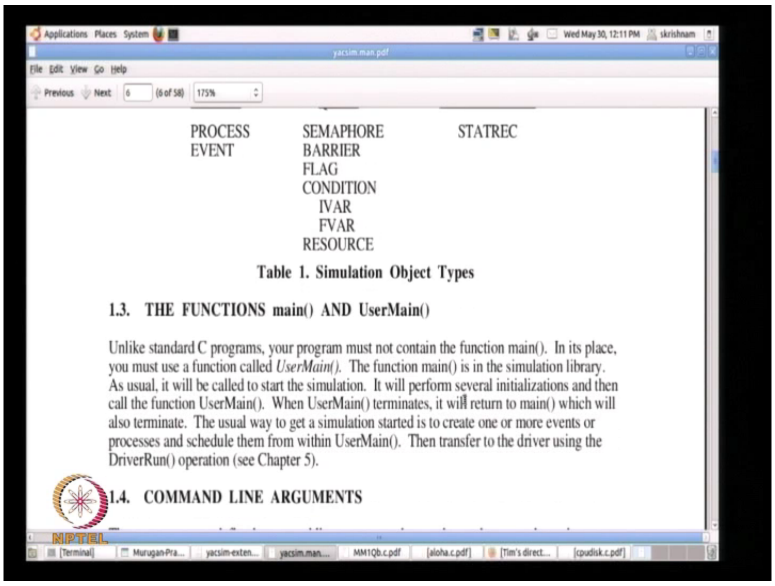
So, it is fairly flexible, that is the reason we are particularly talk talking about it today. We going to, so the three main entities then that when we look at stimulation right in a axiom environment. So, there are three objects and stimulation objects. One of this is called activity and we talk about activity related later on. Then, we when we have on queues or resources. When we are looking on at a system, this is particularly designed for stimulating a set processor processes that communicate with each other. This is more of parallel travel computing where, we also in use network based stimulation. We can build any other form of stimulation where, if you fundamentally required require the notion or of an event or process. That is the basic absorption abstraction, and also need queue servers in the system, wherever is the queuing system we are looking about at we talked about CPU, disk use earlier on in the course. If you want moral to model those, then this is a pretty, good mechanism to achieve that. We have basically activities, queues and satisfaction statistics records that is all.

So, the behavior of the system is going to be captured with the to help of activities, queues and strictest the statistics records help us obtain mean, thorough throughput, mean standard deviation and last lots of other histograms and so on. Some of these are be the built in. you can always extend the way you want. Exton First, we look at activities right; there are two types of activities. So, they describe stimulation system, the basic entities in event right; the system is described as a sequence of events a that happen in different point in time. of system And event one will take this at time and 0, event take two will take at and 10, event 3 at time fifteen and so on. And as we it is discussed early earlier on right, the system will simply jump from say, the first event time 0, the next event say time 10, the next event say time fifteen and so on. So, that is why it is called discrete event stimulation, described event stimulation The event is your desire basic of unit of activity in a given system.

We can write programs using the event based model or processed based model. Those of student s who are familiar with the, sorry unique Unix processors, we know how to write but multi tuner threaded system or multiprocessor programs where we can create a very set of processors and the processors can communicate with each others. The us for example, we use the fork in Unix to create a new process, or use the thread library, we can use corresponding thread create package. For example, we will create (( )) new thread for u right. So, you can imagine the system as of the sequence of events that happen over a on period of the time or sequence, set of communicating processors. And each processor will have responsibility for and managing some for other system activity.

So, sometimes you will find that we can find it is more efficient to write event driven programs. First time For some application, it is easy to write process driven programs. So, both are possible. And we look at combination of the examples as we go later on, and role it.

(Refer Slide Time: 05:16)



PROCESS	SEMAPHORE	STATREC
EVENT	BARRIER	
	FLAG	
	CONDITION	
	IVAR	
	FVAR	
	RESOURCE	

Table 1. Simulation Object Types

1.3. THE FUNCTIONS main() AND UserMain()

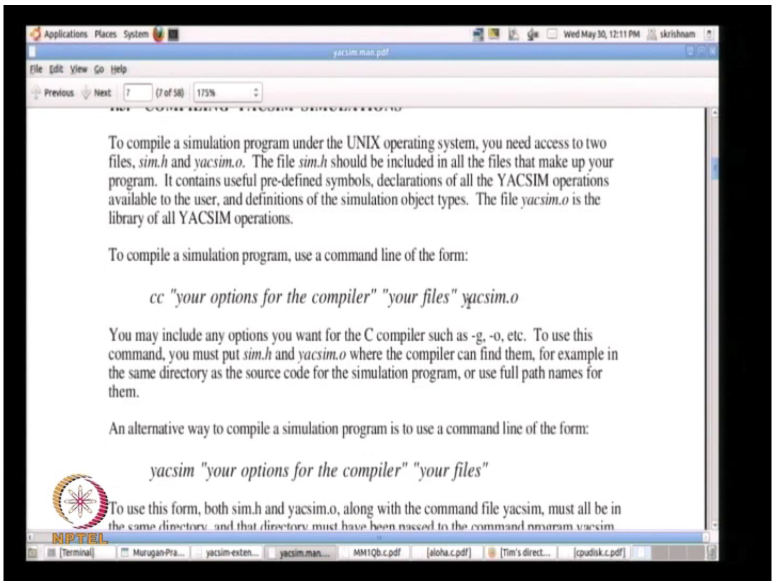
Unlike standard C programs, your program must not contain the function main(). In its place, you must use a function called *UserMain()*. The function main() is in the simulation library. As usual, it will be called to start the simulation. It will perform several initializations and then call the function UserMain(). When UserMain() terminates, it will return to main() which will also terminate. The usual way to get a simulation started is to create one or more events or processes and schedule them from within UserMain(). Then transfer to the driver using the DriverRun() operation (see Chapter 5).

1.4. COMMAND LINE ARGUMENTS

So, I will just highlight some of the main commands in of the system right and this is what we said and so there is a the process activity and event activity, and queues there are different kinds of queues. For those of you, who have done operating system scores, you what a those different kind of queues operating system is, what is semaphore is, flag is and so on fly design. So, these primarily used when you have only have paralleling processing and or thread base processors, which have do to coordinate quarantines and it probably synchronization, you probably must have learnt about right. And semaphore is basic construct constraint very as Barriers, flags, condition variables, all these are different type of queue stature structure and what we will use on particular system in since we talked about primarily what a queue is notation. Notion of what is called resource? The resource is nothing but a server, which has a queue associated with that.

So, these are the 3 main. Statrec is the statistic record we talked about early earlier on. And we just look at the program and just a couple of things about how to invent invoke the program. You will have a function called write user main right. User main, is equivalent of a main event of in the C program. The main in the action program, so the axiom of the program is defined elsewhere. So, we cannot hat may user rewrite the main. So, we write the User Main and that is where and that is where your code starts executing executed. So, the whatever our program you will write, it will start executing right from the User main.

(Refer Slide Time: 06:29)


To compile a simulation program under the UNIX operating system, you need access to two files, *sim.h* and *yacsim.o*. The file *sim.h* should be included in all the files that make up your program. It contains useful pre-defined symbols, declarations of all the YACSIM operations available to the user, and definitions of the simulation object types. The file *yacsim.o* is the library of all YACSIM operations.

To compile a simulation program, use a command line of the form:

cc "your options for the compiler" "your files" yacsim.o

You may include any options you want for the C compiler such as -g, -o, etc. To use this command, you must put *sim.h* and *yacsim.o* where the compiler can find them, for example in the same directory as the source code for the simulation program, or use full path names for them.

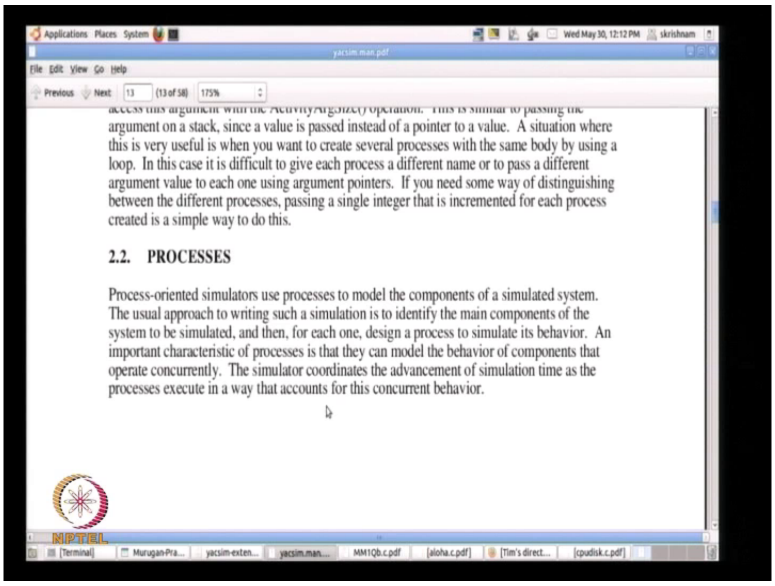
An alternative way to compile a simulation program is to use a command line of the form:

yacsim "your options for the compiler" "your files"

To use this form, both sim.h and yacsim.o, along with the command file yacsim, must all be in the same directory, and that directory must have been passed to the command program yacsim

So, that is a small basic and there are several command line arguments, programs which you can go through. And completion compilation is also very simple. That, there is file called axiom dot o, which is the object file all the entities that contains all the complied files. And then you simply have your files which is what our, a dot c b dot c so on. Compile that with any other axiom C executable and others we can compare which I will not can talk about which you can figure out in the manual later. So, now we come to events and eighteen even in processes. I am and skipping many of this as we will look at this only this week.

(Refer Slide Time: 07:16)

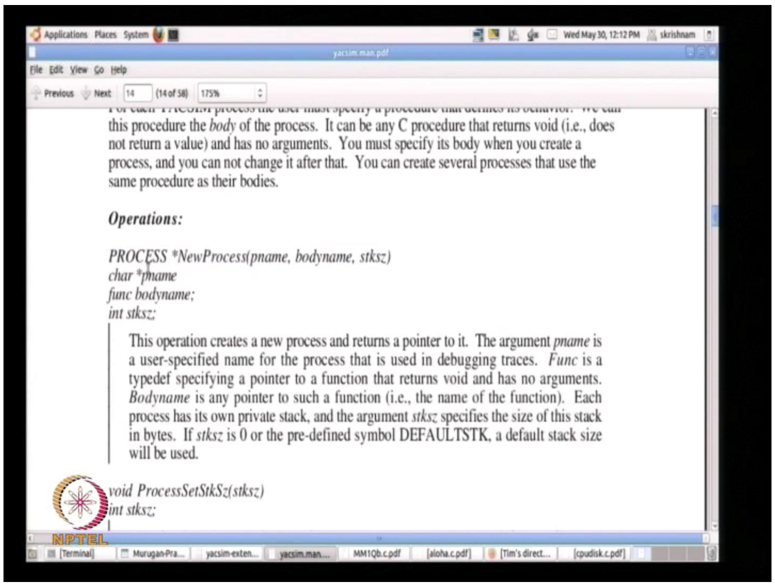

argument on a stack, since a value is passed instead of a pointer to a value. A situation where this is very useful is when you want to create several processes with the same body by using a loop. In this case it is difficult to give each process a different name or to pass a different argument value to each one using argument pointers. If you need some way of distinguishing between the different processes, passing a single integer that is incremented for each process created is a simple way to do this.

2.2. PROCESSES

Process-oriented simulators use processes to model the components of a simulated system. The usual approach to writing such a simulation is to identify the main components of the system to be simulated, and then, for each one, design a process to simulate its behavior. An important characteristic of processes is that they can model the behavior of components that operate concurrently. The simulator coordinates the advancement of simulation time as the processes execute in a way that accounts for this concurrent behavior.

So, let us look at us how to create the a process. So, the process if u look at thread 4 right, unique In Unix, it is a new process, the child process and parent process can do something and child process can do something else. Here, it is here may be similar to what you would do in a us windows based system where you call a function called function called new process right based system

(Refer Slide Time: 07:35)


this procedure the *body* of the process. It can be any C procedure that returns void (i.e., does not return a value) and has no arguments. You must specify its body when you create a process, and you can not change it after that. You can create several processes that use the same procedure as their bodies.

Operations:

*PROCESS *NewProcess(pname, bodyname, stksz)*
*char *pname*
func bodyname;
int stksz;

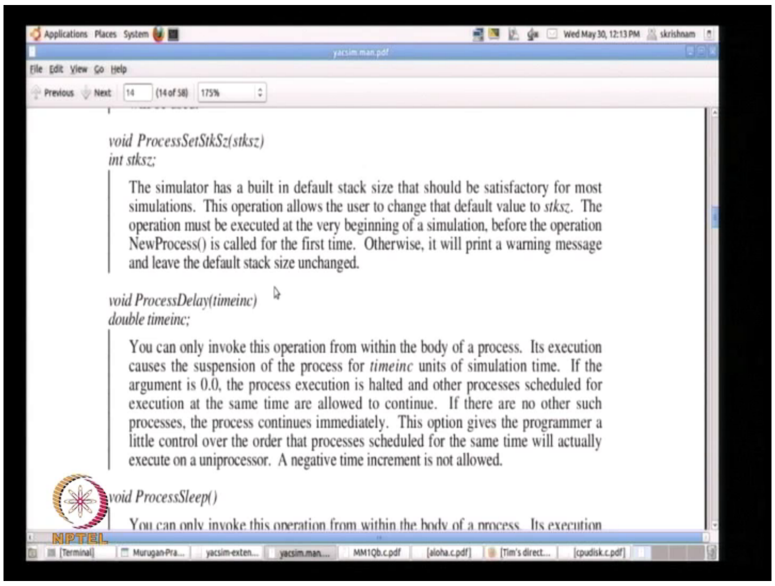
This operation creates a new process and returns a pointer to it. The argument *pname* is a user-specified name for the process that is used in debugging traces. *Func* is a typedef specifying a pointer to a function that returns void and has no arguments. *Bodyname* is any pointer to such a function (i.e., the name of the function). Each process has its own private stack, and the argument *stksz* specifies the size of this stack in bytes. If *stksz* is 0 or the pre-defined symbol DEFAULTSTK, a default stack size will be used.

void ProcessSetStkSz(stksz)
int stksz;

So, this is basic function that you are going to use create a new processor. When you start the action axiom, there is no process in the system. There is just one basic process right that is, you can now create additional processors as you need and then define the function for inch each of these processors We with the corresponding function right.

So, this is the gain again new process in the function that you called to create the process. And the pname is just the name given for the sake of shaking tracking purposes and diagnostic purposes and debugging purposes. And there is function and definition. The body name is simply the predefined name function that of you write in some other part of code. For example, you can write the process say, simply count from one to 10 say. So you will write the function the count one to 10 and define whatever our is going to have happen there right. So, write this is basically process logic is defined as a in logic that defined the function called the body name. So, there is called some tab functions accessories, which we will skip for the time being. So, basically you need to give the processor name and function that the contains the logic that is the process that is going to execute. That is the main interface to create process; that how will you create a new process.

(Refer Slide Time: 08:37)



Applications Places System
Wed May 30, 12:13 PM skrishnam
yacsim.man.pdf
File Edit View Go Help
Previous Next 14 (14 of 58) 175%
[Terminal] Murugan-Pra... yacsim-exten... yacsim.man... MM1Qb.c.pdf [aloha.c.pdf] [Tim's direct... [cpudisk.c.pdf]

void ProcessSetStkSz(stksz)
int stksz;

The simulator has a built in default stack size that should be satisfactory for most simulations. This operation allows the user to change that default value to *stksz*. The operation must be executed at the very beginning of a simulation, before the operation NewProcess() is called for the first time. Otherwise, it will print a warning message and leave the default stack size unchanged.

void ProcessDelay(timeinc)
double timeinc;

You can only invoke this operation from within the body of a process. Its execution causes the suspension of the process for *timeinc* units of simulation time. If the argument is 0.0, the process execution is halted and other processes scheduled for execution at the same time are allowed to continue. If there are no other such processes, the process continues immediately. This option gives the programmer a little control over the order that processes scheduled for the same time will actually execute on a uniprocessor. A negative time increment is not allowed.

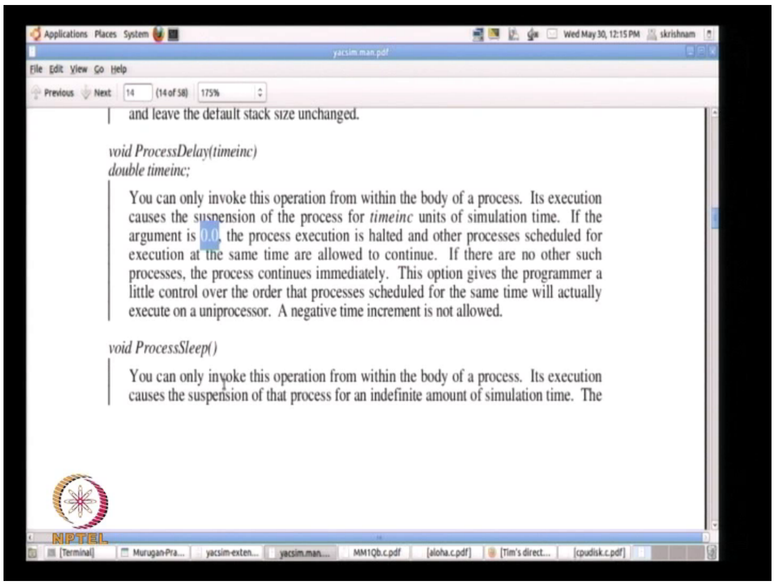
void ProcessSleep()

You can only invoke this operation from within the body of a process. Its execution

So, the process will be containing C a sequins of C lines stricter (( )) statement that is all right. And then basically references it represents the logic. The couple of important logic statement which are not available traditionally right, in numerical computation; one is called the process delay. So, the process delay is the special function which is in work when invoked within a process will simply delay of suspend the process for the passepied specified amount of time. So, I can say for example, for process delay 10, and will system suspend this process for 10 units of time. That unit of the time is important. It is‘s not your real clock seconds; it simply the simulation time.

So, Intel internal to your simulator, there is a that clock that is running; that you are in control of. And this is simply in terms of simulation time units, that there is something very important; and if u want, and you can also make this also 0.0 which is the special case, which, I will skip for now. 0.0 simply to eel yield process to some other processes. So every ever action your axiom program action program can have 15 processes running and u might want just this is running on single CPU right; actually emulating the execution of multiples process on the single CPU. So, if you want to give up the CPU and like let some other process, which is every action a axiom process run; this is the way of doing that. So, it is simply to eel led yield and if there is molar weighting no other waiting process that is executed that; then control will come back to this particular action process that you are dealing with.

(Refer Slide Time: 08:37)


Applications Places System
Wed May 30, 12:15 PM skrishnam
yacsim.man.pdf
File Edit View Go Help
Previous Next 14 (14 of 58) 175%
[Terminal] Murugan-Pra... yacsim-exten... yacsim.man... MM1Qb.c.pdf [aloha.c.pdf] [Tim's direct... [cpudisk.c.pdf]

and leave the default stack size unchanged.

void ProcessDelay(timeinc)
double timeinc;

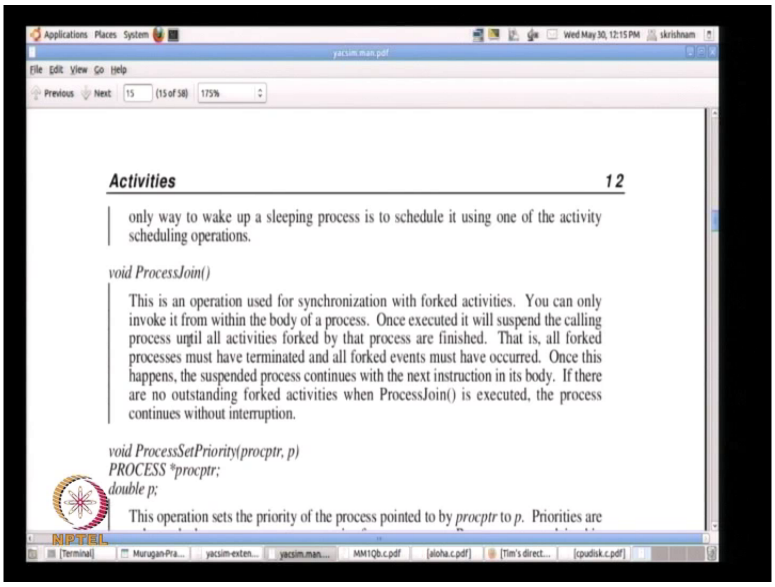
You can only invoke this operation from within the body of a process. Its execution causes the suspension of the process for *timeinc* units of simulation time. If the argument is 0.0, the process execution is halted and other processes scheduled for execution at the same time are allowed to continue. If there are no other such processes, the process continues immediately. This option gives the programmer a little control over the order that processes scheduled for the same time will actually execute on a uniprocessor. A negative time increment is not allowed.

void ProcessSleep()

You can only invoke this operation from within the body of a process. Its execution causes the suspension of that process for an indefinite amount of simulation time. The

So, the process delay is use full useful, when we look at some example we see, where it is to be going used. And process sleep is a special case of process is delay. It is more like putting this process to sleep, but you do not specify how long this is sleep is going to be. That will wake up related later on based on some condition variables right. That again we do not know have an example in the particular class, but this gives more flexibility of writing a generate generic system where for example, time work mode right. Simple example using process sleep is, you put the process to sleep after sending a packet, and you you accessions as soon as the acknowledgement comes back, you wake up the process and process the acknowledgement. And in you never know when is this action is going to come back. That can be done using a special condition variable or some other flags with in which is used to wake up the particular process. So, there this is the process sleep. So, two main activities are creating the process and leading letting the process go to be delayed are or put to sleep.

(Refer Slide Time: 10:45)



Activities 12

only way to wake up a sleeping process is to schedule it using one of the activity scheduling operations.

void ProcessJoin()

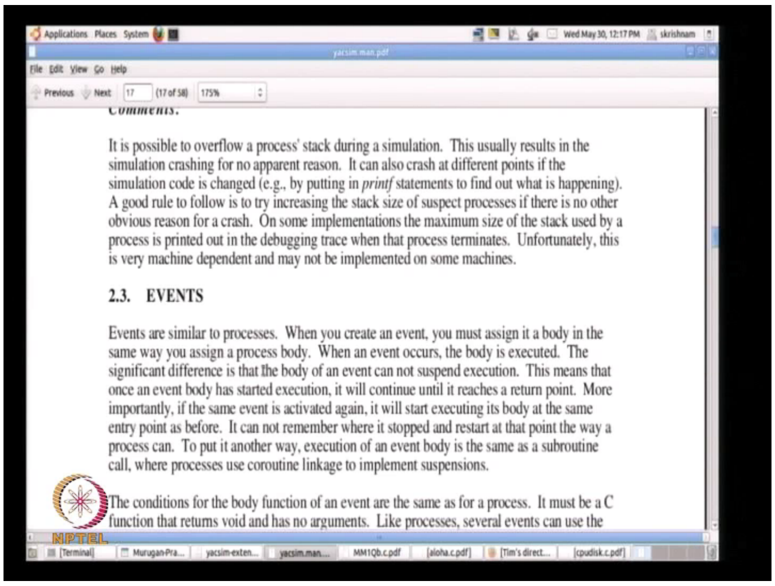
This is an operation used for synchronization with forked activities. You can only invoke it from within the body of a process. Once executed it will suspend the calling process until all activities forked by that process are finished. That is, all forked processes must have terminated and all forked events must have occurred. Once this happens, the suspended process continues with the next instruction in its body. If there are no outstanding forked activities when ProcessJoin() is executed, the process continues without interruption.

void ProcessSetPriority(procptr, p)
*PROCESS *procptr;*
double p;

This operation sets the priority of the process pointed to by *procptr* to *p*. Priorities are

Process join, priority and all those things are optional which you want we look at. If you writing programs where if you remember where four can for ken join right, you can join to wait for some other process to complete before those are particularly proceeding. Those are specify particularly specific to parallel process kind of parallel kind of things routines. We can send messages which again we going to pass over for now.

(Refer Slide Time: 11:06)


Applications Places System
Wed May 30, 12:17 PM skrishnam
yacsim.man.pdf
File Edit View Go Help
Previous Next 17 (17 of 58) 175%
NPTEL
[Terminal] Murugan-Pra... yacsim-exten... yacsim.man.... MM1Qb.c.pdf [aloha.c.pdf] [Tim's direct... [cpudisk.c.pdf]

Comments:

It is possible to overflow a process' stack during a simulation. This usually results in the simulation crashing for no apparent reason. It can also crash at different points if the simulation code is changed (e.g., by putting in *printf* statements to find out what is happening). A good rule to follow is to try increasing the stack size of suspect processes if there is no other obvious reason for a crash. On some implementations the maximum size of the stack used by a process is printed out in the debugging trace when that process terminates. Unfortunately, this is very machine dependent and may not be implemented on some machines.

2.3. EVENTS

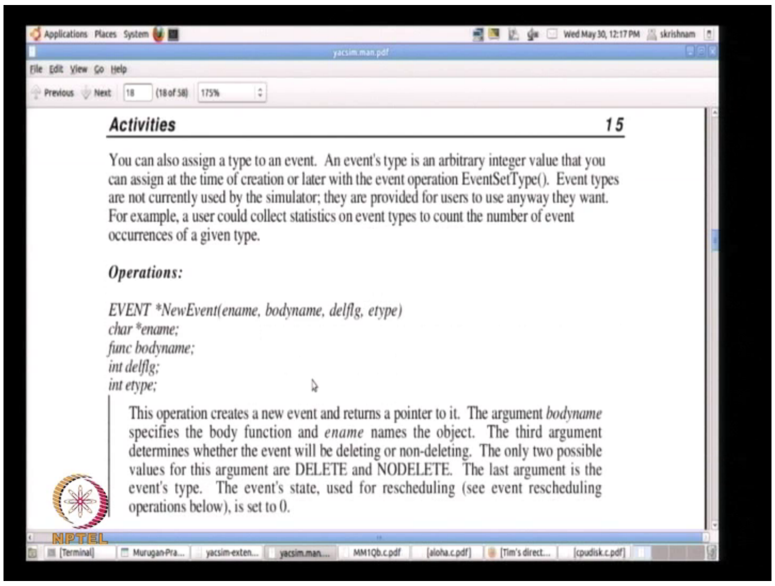
Events are similar to processes. When you create an event, you must assign it a body in the same way you assign a process body. When an event occurs, the body is executed. The significant difference is that the body of an event can not suspend execution. This means that once an event body has started execution, it will continue until it reaches a return point. More importantly, if the same event is activated again, it will start executing its body at the same entry point as before. It can not remember where it stopped and restart at that point the way a process can. To put it another way, execution of an event body is the same as a subroutine call, where processes use coroutine linkage to implement suspensions.

The conditions for the body function of an event are the same as for a process. It must be a C function that returns void and has no arguments. Like processes, several events can use the

So, that is about the process. So, event is simulate similar to a the process. So, the process has the body that right, the function that can be, associated with that the process. And the process can be a long time running process. For example, in an arrival the process in have, stimulation program can run for entire duration of the simulation right. So, it has much longer life time association. That is the definition of the process. It simply can run. You can kill process in any time, but in the theory process has can run forever never. And event is similar to the process, only made major the difference is, the event is instances instantaneous. It is gets created, it gets triggered. Whenever get triggered, event anyone triggers; you take event, say some activities right. When interrupt is triggered, some action is take him taken and that is it. And trigger is programmed. The write event is forgotten, go back in you can reschedule the event if you want to, but event typically happens only once. That happens, function code associated with the event executes and that is all. That is the end of the (( )) it simply dies ok.

So, you will find that events, so essentially it simply will give you less over head right. Writing a program, event driven takes less over head and requires less memory, but on the other hand the other hand tool the other two will require more memory is required more than.

(Refer Slide Time: 12:19)



Activities 15

You can also assign a type to an event. An event's type is an arbitrary integer value that you can assign at the time of creation or later with the event operation EventSetType(). Event types are not currently used by the simulator; they are provided for users to use anyway they want. For example, a user could collect statistics on event types to count the number of event occurrences of a given type.

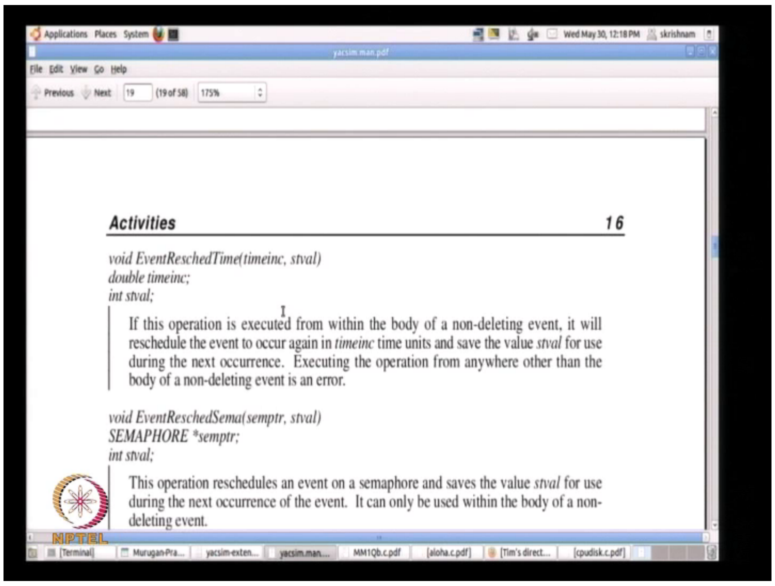
Operations:

```
EVENT *NewEvent(ename, bodyname, delflg, etype)
char *ename;
func bodyname;
int delflg;
int etype;
```

This operation creates a new event and returns a pointer to it. The argument *bodyname* specifies the body function and *ename* names the object. The third argument determines whether the event will be deleting or non-deleting. The only two possible values for this argument are DELETE and NODELETE. The last argument is the event's type. The event's state, used for rescheduling (see event rescheduling operations below), is set to 0.

So, that for therefore, an event is the other they took to create an activity, and then syntax for creating a event is very similar right, you have the notion of event right. The new event is a function that will create a event for you. And just like before, the name associated for debugging and diagnostic and other purposes, whereas body name just like the case of process, there is a logic there that is associated here, is also that a we function you can define. That will handle this particular event right. This is event handler that is trigger. So, the action axiom will take handle while calling this particular function when this event takes place and there are other flags associated with that. For now, simply ignore. So, all I can do is I can like create the event and I can define the logic that will be executed and when the particular event occurs this system. And then there are again lot of other functions that are associated you know.

(Refer Slide Time: 13:06)



Activities 16

void EventReschedTime(timeinc, stval)
double timeinc;
int stval;

If this operation is executed from within the body of a non-deleting event, it will reschedule the event to occur again in *timeinc* time units and save the value *stval* for use during the next occurrence. Executing the operation from anywhere other than the body of a non-deleting event is an error.

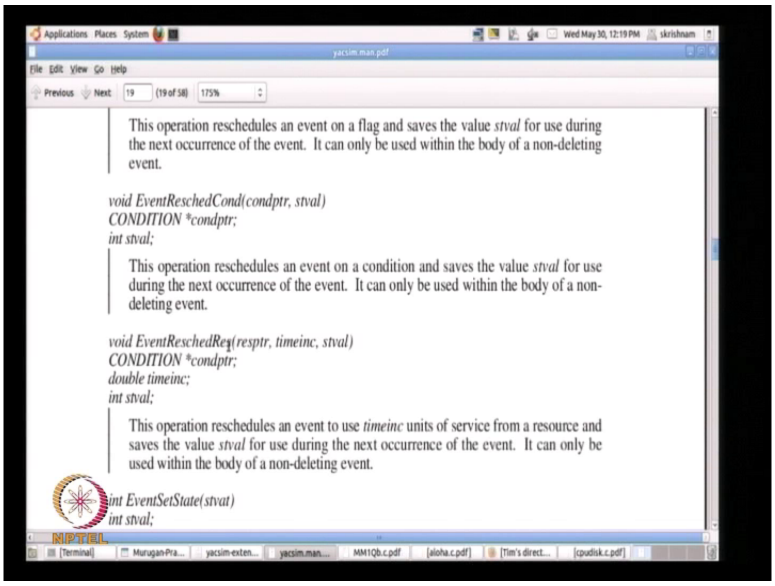
void EventReschedSema(semptr, stval)
*SEMAPHORE *semptr;*
int stval;

This operation reschedules an event on a semaphore and saves the value *stval* for use during the next occurrence of the event. It can only be used within the body of a non-deleting event.

I create an event right, sometimes I want to reschedule the event for some other function point in of time. So, let us take say, some other packet arrival then take place time that then the time t equals to 10 seconds. That is when the event is triggered, so you have now to schedule the packet. Now, you are going to transmit this packet and you know you are going to get that acknowledgement back in 30 seconds right, 30 time units of for the stimulation time. So, you would want to reschedule the same event to occur at some future pointer point in of time. That is for event reschedule purposes. You know how to desire do you decide? That is your part of the logic.

The whole purpose of the simulation program writer is that to understand behavior of system and capture all the behavior using this event processor right. So, if u know that you want the same event to occur red in a future point in time; like to birthday for example time you know that 365 days later the same event has to reoccur again. So, you can reschedule the particular event at future point at time. That is the base case right that we, but all know that I have event that I can schedule, that can happen at some other point in future. There are other more sophisticated things like based on semaphore right, based on flags, like there are some other things like that which I think are not relevant to the examples you're we were going to look at, that we are going to look for more sophisticated things.

(Refer Slide Time: 14:14)



Applications Places System
Wed May 30, 12:19 PM skrishnam
yacsim.man.pdf
File Edit View Go Help
Previous Next 19 (19 of 58) 175%
NPTEL
[Terminal] Murugan-Pra... yacsim-exten... yacsim.man... MM1Qb.c.pdf [aloha.c.pdf] [Tim's direct... [cpudisk.c.pdf]

This operation reschedules an event on a flag and saves the value *stval* for use during the next occurrence of the event. It can only be used within the body of a non-deleting event.

void EventReschedCond(condptr, stval)
*CONDITION *condptr;*
int stval;

This operation reschedules an event on a condition and saves the value *stval* for use during the next occurrence of the event. It can only be used within the body of a non-deleting event.

void EventReschedRes(resptr, timeinc, stval)
*CONDITION *condptr;*
double timeinc;
int stval;

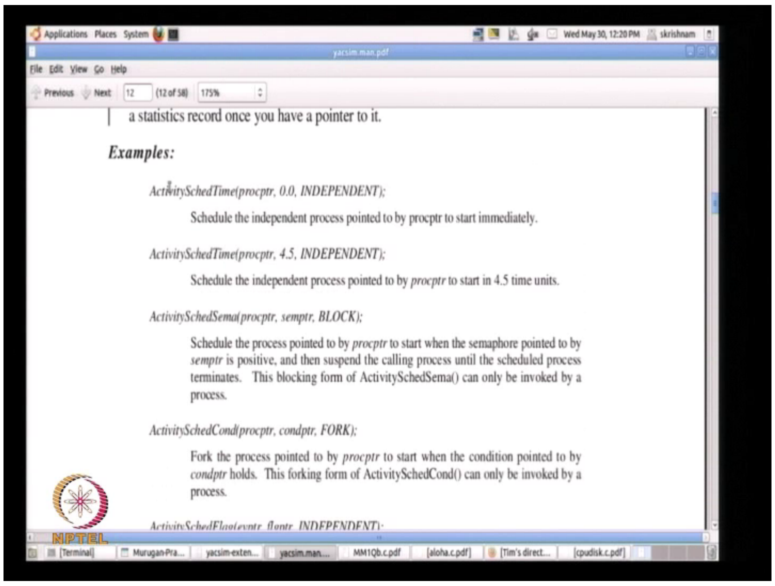
This operation reschedules an event to use *timeinc* units of service from a resource and saves the value *stval* for use during the next occurrence of the event. It can only be used within the body of a non-deleting event.

int EventSetState(stvat)
int stval;

And something special is called the event reschedule, based on the resources availability. So, I talked about the resource being the server; so server has a queue associated with that and the server is normally a single server. We can have multipliers servers also. So, the single server associated that is sever (( )) given queue.

So, when I schedule an event to be scheduled with respect to resources. So, this resources with event will in the go and weight in the servers queue; and when it's turn arrives right, then what will happen; the server will handle all other packets or other event that is in its queue and this particular event will be handled when ever gets to reach sever or getting serviced Servest at the server. And little bit continues confusing no? you We will look at the example, but the other function you will see, would the event reschedule, the resources and other thing we will see what a script early on? There is on a general thing called activity reschedule resources all. So, which is also available. So, the same thing can be done, it can you can also use the activity schedule, event schedule, process schedule and so on.

(Refer Slide Time: 15:18)


Applications Places System
Wed May 30, 12:20 PM
skrishnam
yacsim.man.pdf
File Edit View Go Help
Previous Next 12 (12 of 58) 175%
NPTEL
[Terminal] Murugan-Pra... yacsim-exten... yacsim.man... MM1Qb.c.pdf [aloha.c.pdf] [Tim's direct... [cpudisk.c.pdf]

a statistics record once you have a pointer to it.

Examples:

ActivitySchedTime(procptr, 0.0, INDEPENDENT);

Schedule the independent process pointed to by procptr to start immediately.

ActivitySchedTime(procptr, 4.5, INDEPENDENT);

Schedule the independent process pointed to by *procptr* to start in 4.5 time units.

ActivitySchedSema(procptr, semptr, BLOCK);

Schedule the process pointed to by *procptr* to start when the semaphore pointed to by *semptr* is positive, and then suspend the calling process until the scheduled process terminates. This blocking form of ActivitySchedSema() can only be invoked by a process.

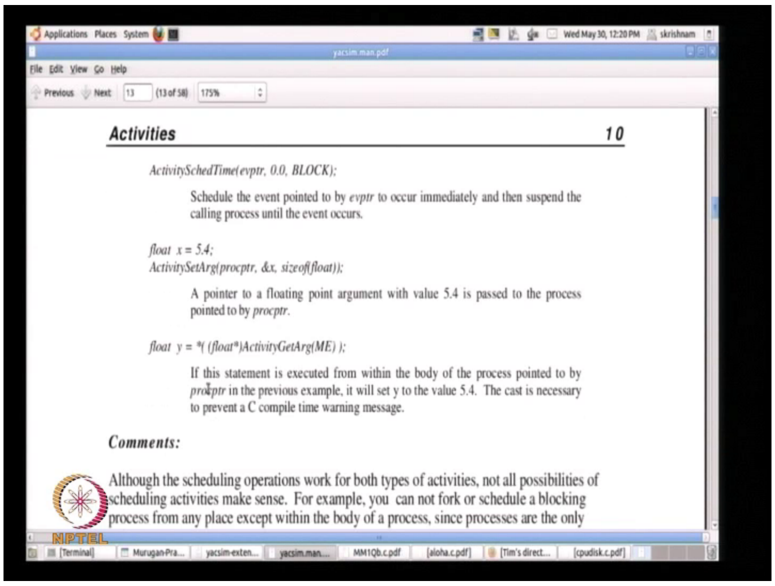
ActivitySchedCond(procptr, condptr, FORK);

Fork the process pointed to by *procptr* to start when the condition pointed to by *condptr* holds. This forking form of ActivitySchedCond() can only be invoked by a process.

ActivitySchedFlag(evptr, flagptr, INDEPENDENT);

You can simply call the thing for generate generic all activity schedule time; the same function, the same functionality as saw before, where the event in the process, we can schedule that to happen in some future point point in time. For example, this case your saying that, the I want this process to scheduled at 4.5 units in the future or we can also schedule this activity based on a resource which particular on example is not there, this starter.

(Refer Slide Time: 15:40)


Applications Places System
Wed May 30, 12:20 PM
skrishnam
yacsim.man.pdf
File Edit View Go Help
Previous Next 13 (13 of 58) 175%
NPTEL
[Terminal] Murugan-Pra... yacsim-exten... yacsim.man... MM1Qb.c.pdf [aloha.c.pdf] [Tim's direct... [cpudisk.c.pdf]

Activities **10**

ActivitySchedTime(evptr, 0.0, BLOCK);

Schedule the event pointed to by *evptr* to occur immediately and then suspend the calling process until the event occurs.

float x = 5.4;
ActivitySetArg(procptr, &x, sizeof(float));

A pointer to a floating point argument with value 5.4 is passed to the process pointed to by *procptr*.

*float y = *((float*)ActivityGetArg(ME));*

If this statement is executed from within the body of the process pointed to by *procptr* in the previous example, it will set y to the value 5.4. The cast is necessary to prevent a C compile time warning message.

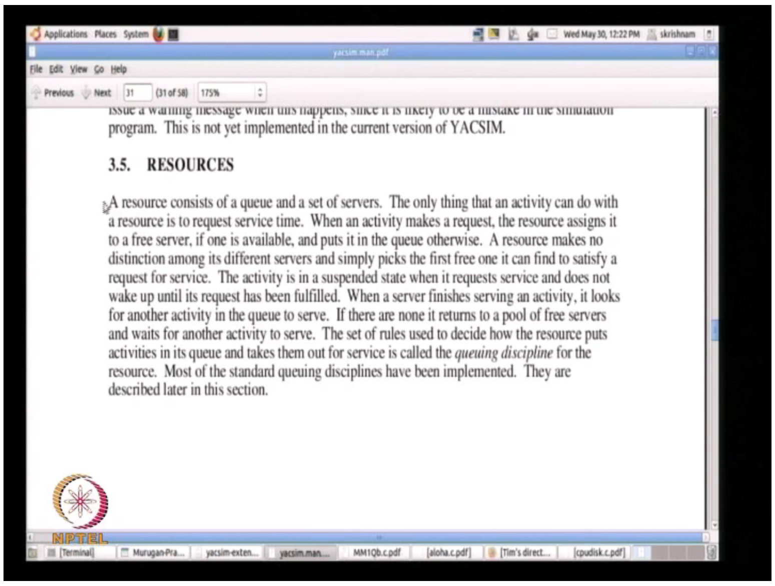
Comments:

Although the scheduling operations work for both types of activities, not all possibilities of scheduling activities make sense. For example, you can not fork or schedule a blocking process from any place except within the body of a process, since processes are the only

When we come to our core we will see that. That is the basic of creating events and creating processes. And we know that, know by action codes has punches of process that would be running, each process contains some part of system logic right. For example, get have an arrival process, departure process, CPU process, disk processors, all those different processors each executing some core logic of the system. And in between, I can through throw in a bunch of events that will created, that will get deleted as and when went near it is needed in the system. That is the these are two main things much most of which pretty much the code is know taken and care of.

And we can create severs from by ourselves, but we can write our notion of the sever we know all node we know how to create a link list right. A sever basically queue or link list stature right. There are five or more structure stature or some other more complicated stature which want to right. So, you can create packet or add packet to queue and serves service the queue the queue any how we add that you want. That is something we can do. It offers all nodes, singly linked list, doubly linked list, even list stack queue whatever you want to implement, they can implement, but action axiom gives you some set of predefined queue stature structure that saves you has that hassle, because you because maintaining pointers, all those things can always leave bulls bugs in the system. So, we have no a special node notion of what our are called as queue as right. So, these are queues are called resources. That is described the stature structure is called queue right.

(Refer Slide Time: 17:11)


Applications Places System
Wed May 30, 12:22 PM skrishnam
yacsim.man.pdf
File Edit View Go Help
Previous Next 31 (31 of 58) 175%

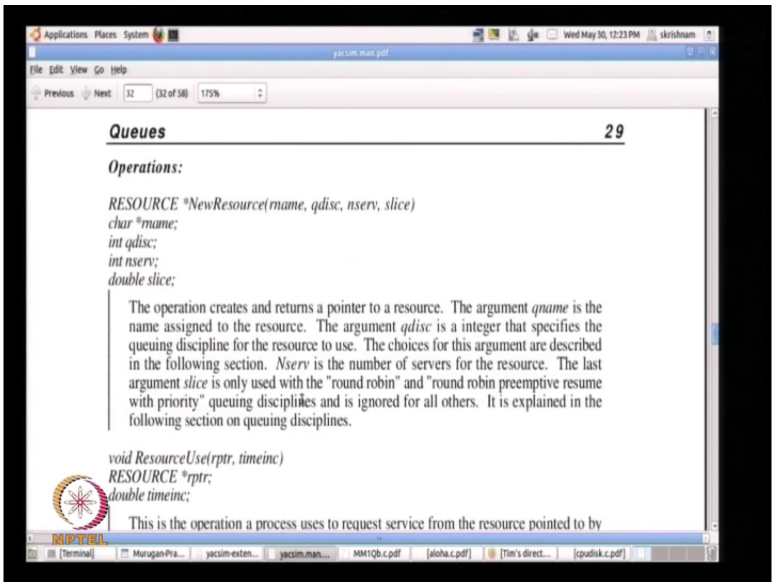
program. This is not yet implemented in the current version of YACSIM.

3.5. RESOURCES

A resource consists of a queue and a set of servers. The only thing that an activity can do with a resource is to request service time. When an activity makes a request, the resource assigns it to a free server, if one is available, and puts it in the queue otherwise. A resource makes no distinction among its different servers and simply picks the first free one it can find to satisfy a request for service. The activity is in a suspended state when it requests service and does not wake up until its request has been fulfilled. When a server finishes serving an activity, it looks for another activity in the queue to serve. If there are none it returns to a pool of free servers and waits for another activity to serve. The set of rules used to decide how the resource puts activities in its queue and takes them out for service is called the *queuing discipline* for the resource. Most of the standard queuing disciplines have been implemented. They are described later in this section.

Let us see. This is the resource. So, the resources action in the axiom consists of set of queue and set of servers. And important what an activity can do at any event or process is to request service time on a particular declare resource. So, the service time is to. So, you need 4.5 units of service time on the particular resource. If it is the CPU time, for the example, you want to use it for 20 milliseconds, the disk you want to use it for 500 milliseconds right. Whenever an activity makes the request for a resource and if resource is free, then that all are the maintenance is done by the axiom action itself; if the resource is free, it will utilize immediately service you. If the resource is currently is servicing in some other activity, then some other packet whatever, then this packet will simply queued in some later pit of service related point in the time. So, that implicit queuing is done whenever you simply schedule an activity on a given resource. And then on all the business of the queuing, de queuing, all that is transparently handled by action axiom itself right. We do not have to see that. As to how that is getting that done.

(Refer Slide Time: 18:08)


Applications Places System
Wed May 30, 12:23 PM skrishnam
yacsim.man.pdf
File Edit View Go Help
Previous Next 32 (32 of 58) 175%
NPTEL
[Terminal] Murugan-Pra... yacsim-exten... yacsim.man... MM1Qb.c.pdf [aloha.c.pdf] [Tim's direct... [cpudisk.c.pdf]

Queues 29

Operations:

*RESOURCE *NewResource(rname, qdisc, nserv, slice)*
*char *rname;*
int qdisc;
int nserv;
double slice;

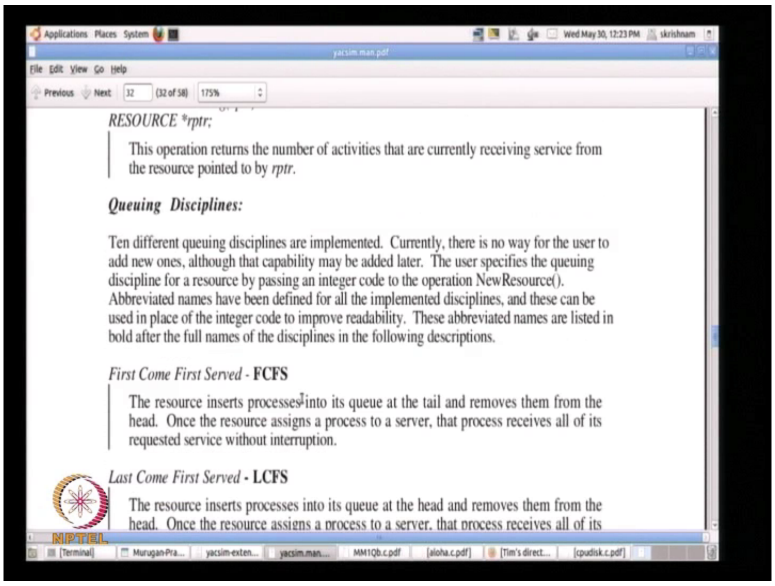
The operation creates and returns a pointer to a resource. The argument *qname* is the name assigned to the resource. The argument *qdisc* is a integer that specifies the queuing discipline for the resource to use. The choices for this argument are described in the following section. *Nserv* is the number of servers for the resource. The last argument *slice* is only used with the "round robin" and "round robin preemptive resume with priority" queuing disciplines and is ignored for all others. It is explained in the following section on queuing disciplines.

void ResourceUse(rptr, timeinc)
*RESOURCE *rptr;*
double timeinc;

This is the operation a process uses to request service from the resource pointed to by

So, let us see how to create resources. So, basically again, will like new event, new process, we have something called new resource. New resource will simply create the resource or some other parameter as before. So, the name of the resources again the for diagnostic purpose what we will use that and then discipline, queue the queue discipline we are used to; first in first out, but there are last lots of other discipline of last come first out is there else so let us look at some of those queuing discipline.

(Refer Slide Time: 18:38)


RESOURCE *rptr;

This operation returns the number of activities that are currently receiving service from the resource pointed to by *rptr*.

Queuing Disciplines:

Ten different queuing disciplines are implemented. Currently, there is no way for the user to add new ones, although that capability may be added later. The user specifies the queuing discipline for a resource by passing an integer code to the operation NewResource(). Abbreviated names have been defined for all the implemented disciplines, and these can be used in place of the integer code to improve readability. These abbreviated names are listed in bold after the full names of the disciplines in the following descriptions.

First Come First Served - **FCFS**

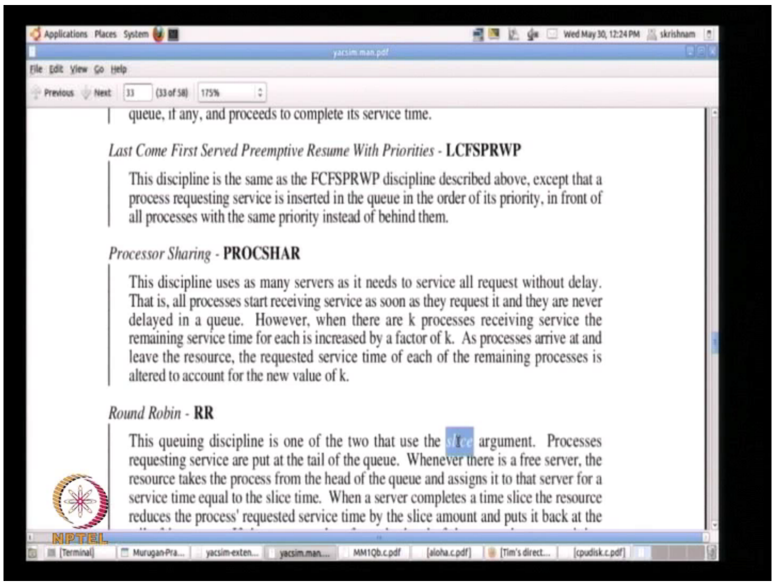
The resource inserts processes into its queue at the tail and removes them from the head. Once the resource assigns a process to a server, that process receives all of its requested service without interruption.

Last Come First Served - **LCFS**

The resource inserts processes into its queue at the head and removes them from the head. Once the resource assigns a process to a server, that process receives all of its

So, first come, first out simply order of getting in to the queue, that is what is the that is used full for. Removing packet from queue, last come first out, and then last come first out, pre empty to the process right. And then you can do and then FCFS with preempty to resume with priority, write right. And same thing as the sharing; some of you have the done at the networking course might remember. It is your GPS, which comes from this are or we can simply have the basic round robin scheduler.

(Refer Slide Time: 19:03)


queue, if any, and proceeds to complete its service time.

Last Come First Served Preemptive Resume With Priorities - **LCFSPRWP**

This discipline is the same as the FCFSPRWP discipline described above, except that a process requesting service is inserted in the queue in the order of its priority, in front of all processes with the same priority instead of behind them.

Processor Sharing - **PROCSHAR**

This discipline uses as many servers as it needs to service all request without delay. That is, all processes start receiving service as soon as they request it and they are never delayed in a queue. However, when there are k processes receiving service the remaining service time for each is increased by a factor of k. As processes arrive at and leave the resource, the requested service time of each of the remaining processes is altered to account for the new value of k.

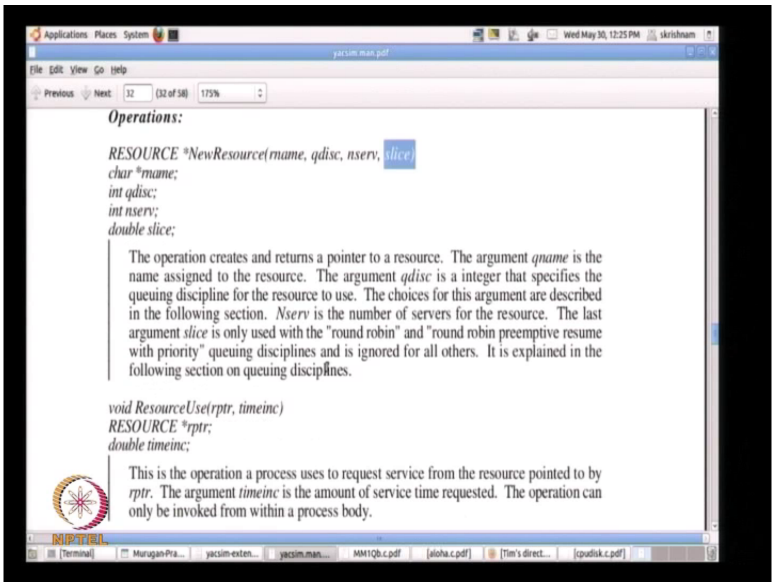
Round Robin - **RR**

This queuing discipline is one of the two that use the slice argument. Processes requesting service are put at the tail of the queue. Whenever there is a free server, the resource takes the process from the head of the queue and assigns it to that server for a service time equal to the slice time. When a server completes a time slice the resource reduces the process' requested service time by the slice amount and puts it back at the

So, the round robin scheduler for the example of it you put 10 packets in queue and each packet is called got hundred bytes right. And with round robin, you can simply say the that each packet say, 10 byte is may the make a quantum or time slices.

So, there is the notion of the slice argument that are given; that have you to file. We also have to specify in round robin u also specify the time quantum that going to use to be every process execute for the particular sever.

(Refer Slide Time: 19:36)



Applications Places System
Wed May 30, 12:25 PM
skrishnam
yacsim.man.pdf
File Edit View Go Help
Previous Next 32 (32 of 58) 175%
[Terminal] Murugan-Pra... yacsim-exten... yacsim.man.... MM1Qb.c.pdf [aloha.c.pdf] [Tim's direct... [cpudisk.c.pdf]
NPTEL

Operations:

```
RESOURCE *NewResource(rname, qdisc, nserv, slice)
char *rname;
int qdisc;
int nserv;
double slice;
```

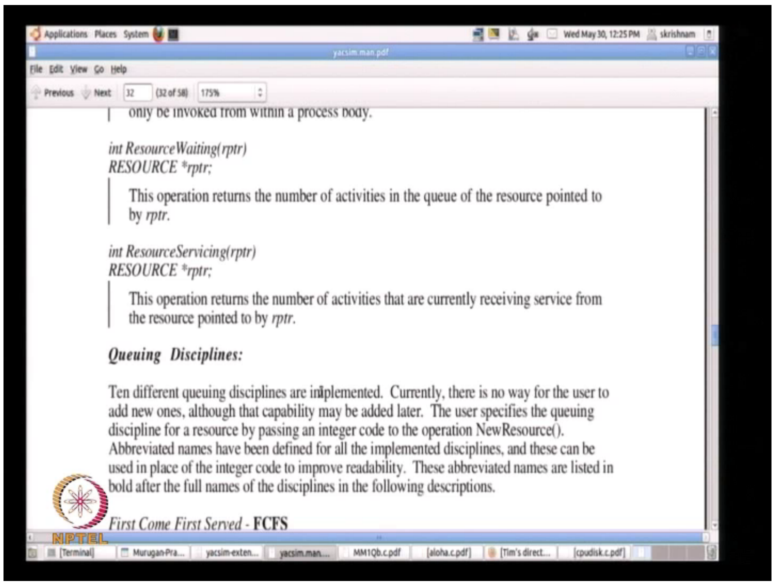
The operation creates and returns a pointer to a resource. The argument *qname* is the name assigned to the resource. The argument *qdisc* is a integer that specifies the queuing discipline for the resource to use. The choices for this argument are described in the following section. *Nserv* is the number of servers for the resource. The last argument *slice* is only used with the "round robin" and "round robin preemptive resume with priority" queuing disciplines and is ignored for all others. It is explained in the following section on queuing disciplines.

```
void ResourceUse(rptr, timeinc)
RESOURCE *rptr;
double timeinc;
```

This is the operation a process uses to request service from the resource pointed to by *rptr*. The argument *timeinc* is the amount of service time requested. The operation can only be invoked from within a process body.

So, that is what we specify. So, we specify the name, we specify the queuing discipline, and also the number of severs. We called M M 1, M M 3, M M 5 and so on right. So, we can specify the number of servers in axiom. It will automatically create that many servers. But this is a the single queue right, there are no multiple queues, but one queue in the system. And then but I can M M (( )) system. Then slices is the basically it can be round robin alone. We are have to specify what is time slice, the basic unit of schedule. So, you can now create the a resources. So, we are now seen processes, we have seen events, we can see resources, that is that we need. Everything else can come in.

(Refer Slide Time: 20:12)



only be invoked from within a process body.

int ResourceWaiting(rptr)
*RESOURCE *rptr;*

This operation returns the number of activities in the queue of the resource pointed to by *rptr*.

int ResourceServicing(rptr)
*RESOURCE *rptr;*

This operation returns the number of activities that are currently receiving service from the resource pointed to by *rptr*.

Queuing Disciplines:

Ten different queuing disciplines are implemented. Currently, there is no way for the user to add new ones, although that capability may be added later. The user specifies the queuing discipline for a resource by passing an integer code to the operation NewResource(). Abbreviated names have been defined for all the implemented disciplines, and these can be used in place of the integer code to improve readability. These abbreviated names are listed in bold after the full names of the disciplines in the following descriptions.

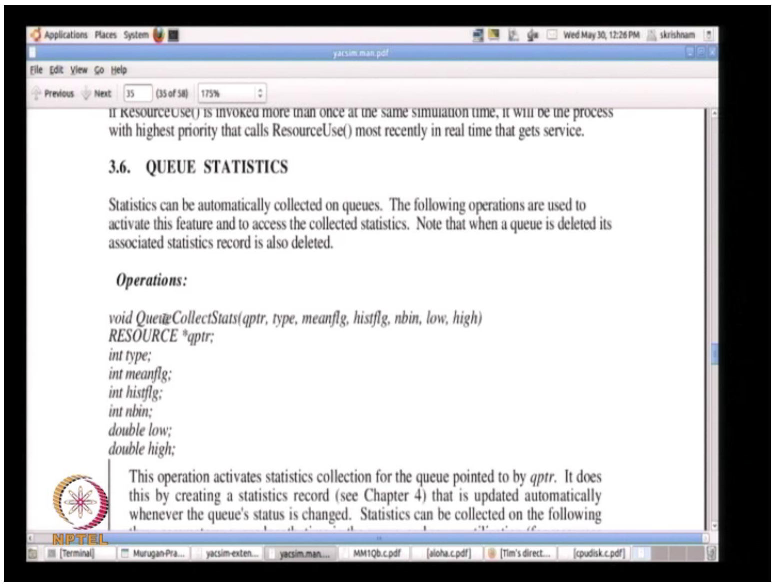
First Come First Served - **FCFS**

So, event sometimes whatever be the schedule you are using, with similar using the weather Omnet or whether it is the ns two; ultimately that is called is happening. You create a bunch of logic and it is organized is term of modules. In for example, Omnet derides the module of action ns two, you have write various objects in that are created, ultimately underlying what happens is there are events scheduled; and this happens over and over again, and finally you can stop the stimulation is at some point of in time.

So, the before the stimulation terminates, what do you we want to do? Want to measure performance. So, whole idea running the stimulation this is which gives values of some system. So, we need to measure performance matrices matrix. And performance matrices matrix, we can do that all sources ourselves right and depending on the system. In case of Opnet probably, you can have a whole bench bunch of performance matrices that it you will have go and search right, depend dig up and find what they are. In the case of n s two, you normally have trace file right; that the simply dumps everything into a round large file, new script or script right, what the packet are; what the throughput also this is done manually are or in all our case of we can choose to dump in what our values.

So, you keep on aggregating some statistic variables as go long. And we can just for example, number of packets completed right, you can simplify count that. Count every time packet is finished, to keep incrementing the count. And at the end of the time system, we can say we delivered 10,000 packets, whatever 50,000 time units. Therefore, throughput is point 5. So, that is something that the we can do. The axiom provides you with a set of basic static record right, which little bit which help you. In some cases mostly we write other our own statistic our routine also.

(Refer Slide Time: 21:51)



If ResourceUse() is invoked more than once at the same simulation time, it will be the process with highest priority that calls ResourceUse() most recently in real time that gets service.

3.6. QUEUE STATISTICS

Statistics can be automatically collected on queues. The following operations are used to activate this feature and to access the collected statistics. Note that when a queue is deleted its associated statistics record is also deleted.

Operations:

```
void QueueCollectStats(qptr, type, meanflg, histflg, nbin, low, high)
RESOURCE *qptr;
int type;
int meanflg;
int histflg;
int nbin;
double low;
double high;
```

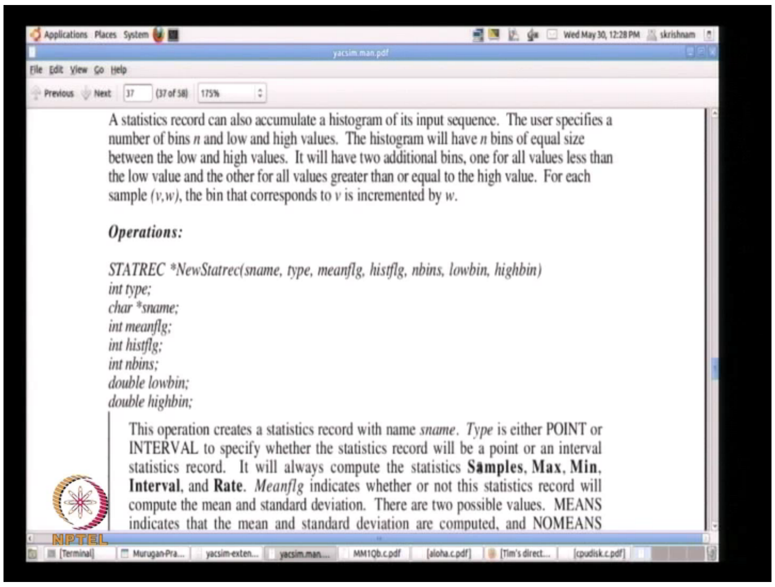
This operation activates statistics collection for the queue pointed to by *qptr*. It does this by creating a statistics record (see Chapter 4) that is updated automatically whenever the queue's status is changed. Statistics can be collected on the following

For example, we can initiate collection of statistics the for particular queue. So, the resources we created above, we can ask the axiom of to keep track in various things right, the amount of time, the busy, for the example, let us That is one thing, utilization of sever right, row value; that we defined in M M 1 system. For example, you can keep, you can ask the axiom to do that for you are or the service time for you, different packet. You can just calculate name mean service time; you also calculate statistics in terms of histogram right.

So, we you defined what your different number of bins are and write values of and then it will keep atomically keep track those things. So, there are So, there very sophisticated things you can do that. And decide besides the queue is alone right, u can directly gather statistics for the queue queue collect, but you can we can also gather other statistics for example, number packet are delivered or delayed of the particular packet right. All those things you do with the help, notion of the statistic record that is the predefined in the axiom. So, new static Statrec, that is the function that is available and here we can specify right whether it interval point interval and interval, very where you want to write.

It computes number of samples, maximum value of the sample, minimum value of the sample.

(Refer Slide Time: 22:57)



A statistics record can also accumulate a histogram of its input sequence. The user specifies a number of bins *n* and low and high values. The histogram will have *n* bins of equal size between the low and high values. It will have two additional bins, one for all values less than the low value and the other for all values greater than or equal to the high value. For each sample *(v,w)*, the bin that corresponds to *v* is incremented by *w*.

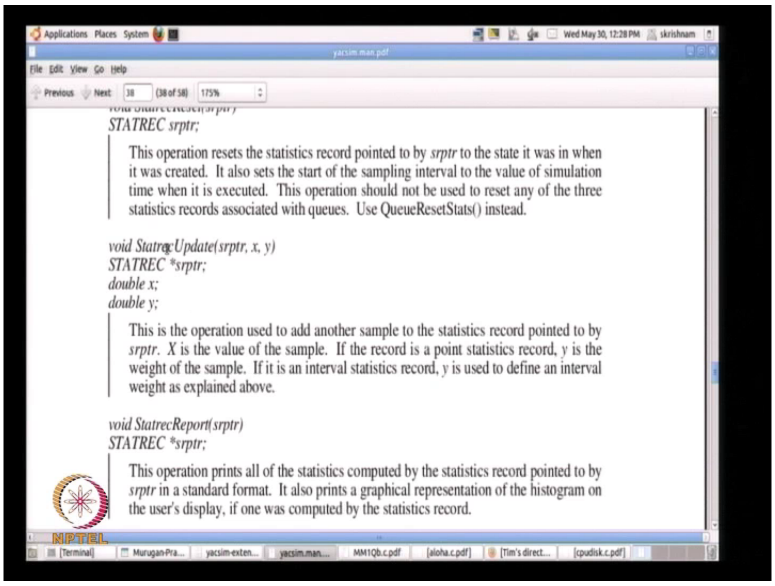
Operations:

*STATREC *NewStatrec(sname, type, meanflg, histflg, nbins, lowbin, highbin)*
int type;
*char *sname;*
int meanflg;
int histflg;
int nbins;
double lowbin;
double highbin;

This operation creates a statistics record with name *sname*. *Type* is either POINT or INTERVAL to specify whether the statistics record will be a point or an interval statistics record. It will always compute the statistics **Samples**, **Max**, **Min**, **Interval**, and **Rate**. *Meanflg* indicates whether or not this statistics record will compute the mean and standard deviation. There are two possible values. MEANS indicates that the mean and standard deviation are computed, and NOMEANS

So, every time you transmit the packet, you will call the this new satiric Statrec. Automatically, it will increment the number of the samples. It updates the max value, min value right and mean value all of things are automatically taken it care of it. This is the conveying convenient way of doing it. We are also doing it in different ways.

(Refer Slide Time: 23:19)



STATREC srptr;

This operation resets the statistics record pointed to by *srptr* to the state it was in when it was created. It also sets the start of the sampling interval to the value of simulation time when it is executed. This operation should not be used to reset any of the three statistics records associated with queues. Use QueueResetStats() instead.

void StatrecUpdate(srptr, x, y)
*STATREC *srptr;*
double x;
double y;

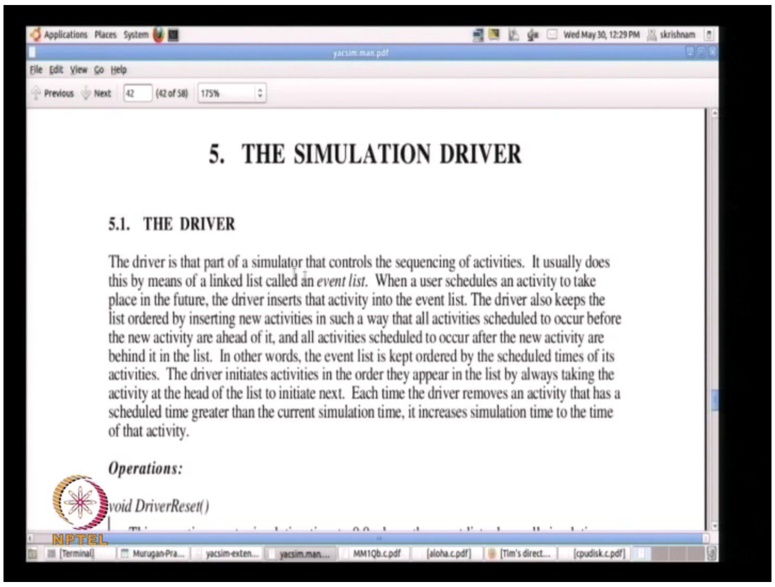
This is the operation used to add another sample to the statistics record pointed to by *srptr*. *X* is the value of the sample. If the record is a point statistics record, *y* is the weight of the sample. If it is an interval statistics record, *y* is used to define an interval weight as explained above.

void StatrecReport(srptr)
*STATREC *srptr;*

This operation prints all of the statistics computed by the statistics record pointed to by *srptr* in a standard format. It also prints a graphical representation of the histogram on the user's display, if one was computed by the statistics record.

And so function, will call to every time you want to update the new value or update the new value for the static collection is called as static update Statrec update collection function, which will see in the example of as we go. So, now we have they are satiric set of routines now we are everything in place. We have the process, events, queue all the satiric statisitic records, then only one important thing is remaining and that is, and actually running that stimulation, actually some out of the trigger stimulation. What we look at the stimulation core code, you will see the raw definition here, all set of process here, all set of events, all those get created here. But somewhere you have to give the initial trigger right. It is like creation; somebody has come in and start that the first event right to occur. Then the rest of events the will automatically start happening right. So, that first event creation function is this handled with the help of a set of special function which we will talk about.

(Refer Slide Time: 24:16)



Applications Places System
Wed May 30, 12:29 PM skrishnam
yacsim.man.pdf
File Edit View Go Help
Previous Next 42 (42 of 58) 175%

5. THE SIMULATION DRIVER

5.1. THE DRIVER

The driver is that part of a simulator that controls the sequencing of activities. It usually does this by means of a linked list called an *event list*. When a user schedules an activity to take place in the future, the driver inserts that activity into the event list. The driver also keeps the list ordered by inserting new activities in such a way that all activities scheduled to occur before the new activity are ahead of it, and all activities scheduled to occur after the new activity are behind it in the list. In other words, the event list is kept ordered by the scheduled times of its activities. The driver initiates activities in the order they appear in the list by always taking the activity at the head of the list to initiate next. Each time the driver removes an activity that has a scheduled time greater than the current simulation time, it increases simulation time to the time of that activity.

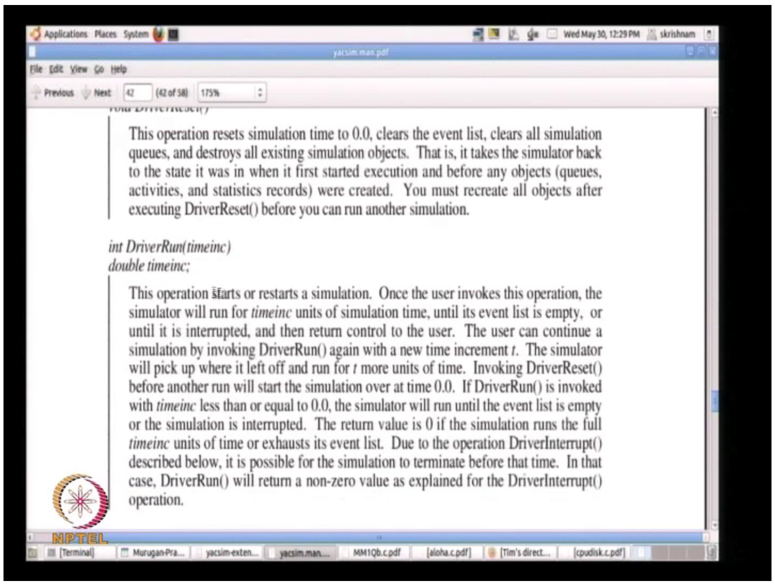
Operations:

void DriverReset()

[Terminal] Murugan-Pra... yacsim-exten... yacsim.man... MM1Qb.c.pdf [aloha.c.pdf] [Tim's direct... [cpudisk.c.pdf]

So, basically called this is called stimulation driver right. Only when the stimulation driver start executing do all the events has start scheduled and process has starts scheduling and so on. And until then it is a static set of definition and nothing this is really happening right. So, this is the main part of stimulation and that is actually controlling all the sequencing events of all the activities of the system right.

(Refer Slide Time: 24:36)


Applications Places System
Wed May 30, 12:29 PM skrishnam
yacsim.man.pdf
File Edit View Go Help
Previous Next 42 (42 of 58) 175%
NPTEL
[Terminal] Murugan-Pra... yacsim-exten... yacsim.man.... MM1Qb.c.pdf [aloha.c.pdf] [Tim's direct... [cpudisk.c.pdf]

This operation resets simulation time to 0.0, clears the event list, clears all simulation queues, and destroys all existing simulation objects. That is, it takes the simulator back to the state it was in when it first started execution and before any objects (queues, activities, and statistics records) were created. You must recreate all objects after executing DriverReset() before you can run another simulation.

int DriverRun(timeinc)
double timeinc;

This operation starts or restarts a simulation. Once the user invokes this operation, the simulator will run for *timeinc* units of simulation time, until its event list is empty, or until it is interrupted, and then return control to the user. The user can continue a simulation by invoking DriverRun() again with a new time increment *t*. The simulator will pick up where it left off and run for *t* more units of time. Invoking DriverReset() before another run will start the simulation over at time 0.0. If DriverRun() is invoked with *timeinc* less than or equal to 0.0, the simulator will run until the event list is empty or the simulation is interrupted. The return value is 0 if the simulation runs the full *timeinc* units of time or exhausts its event list. Due to the operation DriverInterrupt() described below, it is possible for the simulation to terminate before that time. In that case, DriverRun() will return a non-zero value as explained for the DriverInterrupt() operation.

And, the basic function that you would C see is the driver run. The driver will start the stimulation to be get executing executed and specify the time line of how long you want it to run. You can say, 10,000 second; again time is important. This is not micro second, millisecond. This is simply stimulation time unit right. This is something which you have to aware of. something And whenever, creating the system 100 m b ps links and so on; you have to know how to translate that real time values into stimulation time values; and again you can define the granularity of the clock in this particular one. It can be one stimulation time is equal to 10 micro seconds or 10 milli or 10 seconds whatever. You want define and based on that you can advance the clock internally create and delete.

So, this is the main thing. Once you call the driver run, it will keep running until and unless it is interrupted. You can always interrupt using a special call called as driver interrupt. So, early on you can interrupt right the existing stimulation, or it will simply run to his million units or whatever, it will simply keep running. The most important thing is stimulation is when to stop the stimulation right, whether 10,000 unit ok, 100 1000 units ok, we never really know. And each stimulation run can converge in a different stimulation time. So, want we do is, we simply talked about terms and termination condition right.

So what we do is we have 1 thing we talked about in the other lectures right, in theory part other lecture we talk about terms and terminating conditions based on mean values right and also lots of different condition that we have discussed. And trying to we find out of is little bit of a challenge. So, mostly people just pop out and specify some large value right. Whatever the guides says you simply follow that. If it says run million units of time then you want to because it takes two and of days to finish the work. That is not option right. You prepared 100 units of time, you get some samples and it get some you are very happy; but unless you collect statistically large number of samples, but you can't get significant results. If you run 1000 time units, you may get about of 50 samples. that There is no way near to get meaningful interpretation of the results.

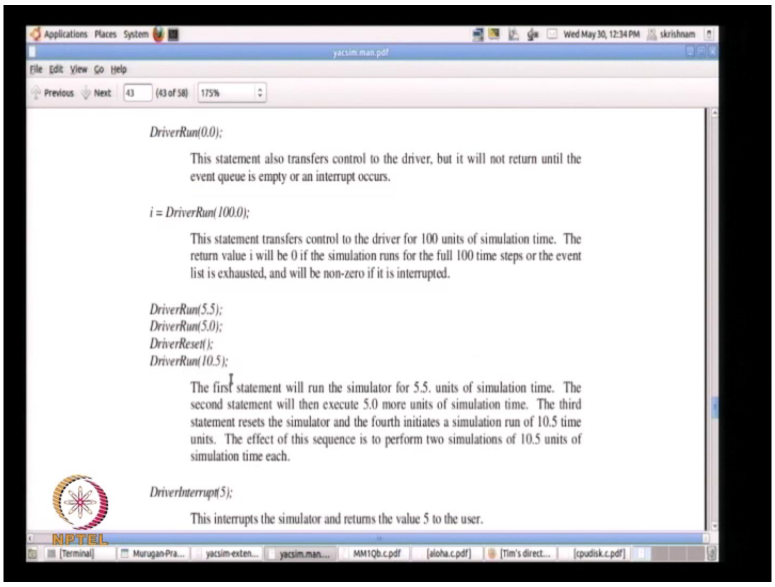
So, running it for large number of samples is important, collection of how much time is not as relevant as large number of samples how time samples you are collecting. not to relevant has a number samples required Because if you are collecting sampling for delay you need to know. For example, time you are trying to find out to bit error rate in a particular system right or a packet draft drop rate in particular system right; and the packet error rate is 10 to the power minus 6. Then you have run at least the a million packet generation or packet completions before you can see the first packet drop. If really you want to stop at 1000, then you would say that the packet drop rate is 0.

But, even for 10 to the power 6, you have at least 1000 times; 10 to the power 9 completions you have to get to make sure that he your packet drop rates are remaining meaningful. So that, and those are also important condition right. And right now, most of the time we just select a large enough number. But what we actually do is, we run it for increasing units but it by increasing number run for 2 million units, 5 million units and what you will find out what that compromise between the time taken, the real time taken the for stimulation hand and the accuracy of the values. can be That you will find that the out code is converging reasonably well, so you can stop the earlier.

In fact, it is the exterior extension to the axiom, called as the new Statrec, which is actually called the driver auto of terminate, which I will go to that later on where can related where we can automatically terminate the stimulation once at converge conditions are attained. Again, we talked about termination conditions so those are again actually programmed as the separate set of function. So, ideally what we can do is, converge on delay. What I will we do is, If delay converge of delay if delay with every 500 samples that change the mean delay, the mean delay after the 9000 samples, after 10,000 samples is roughly he same; then simply you want to stop at 9000 samples the right. You want special converge condition, you can define, assuming that the delay converges right, then you want to, because you want to know running for one parameter value you want the parameter value for 30 replications, 10 replication; that simply takes time. And several combination of parameters values those have are finished M Tech recently will know 8 number of replications, amount stimulation that takes place.

No Know those to working and on stimulation know, how long it takes to get the basic result. That you take 10 times a day, 25 days for one data value, you have value hundred parameter combination right it takes for even ever. And unit you need more machines, then are you have to be claver clever to stop your termination as soon as your converge takes place. So, that is something that is very important.

(Refer Slide Time: 29:02)


Applications Places System
Wed May 30, 12:34 PM skrishnam
yacsim.man.pdf
File Edit View Go Help
Previous Next 43 (43 of 58) 175%
NPTEL
[Terminal] Murugan-Pra... yacsim-exten... yacsim.man... MM1Qb.c.pdf [aloha.c.pdf] [Tim's direct... [cpudisk.c.pdf]

DriverRun(0.0);

This statement also transfers control to the driver, but it will not return until the event queue is empty or an interrupt occurs.

i = DriverRun(100.0);

This statement transfers control to the driver for 100 units of simulation time. The return value i will be 0 if the simulation runs for the full 100 time steps or the event list is exhausted, and will be non-zero if it is interrupted.

DriverRun(5.5);
DriverRun(5.0);
DriverReset();
DriverRun(10.5);

The first statement will run the simulator for 5.5. units of simulation time. The second statement will then execute 5.0 more units of simulation time. The third statement resets the simulator and the fourth initiates a simulation run of 10.5 time units. The effect of this sequence is to perform two simulations of 10.5 units of simulation time each.

DriverInterrupt(5);

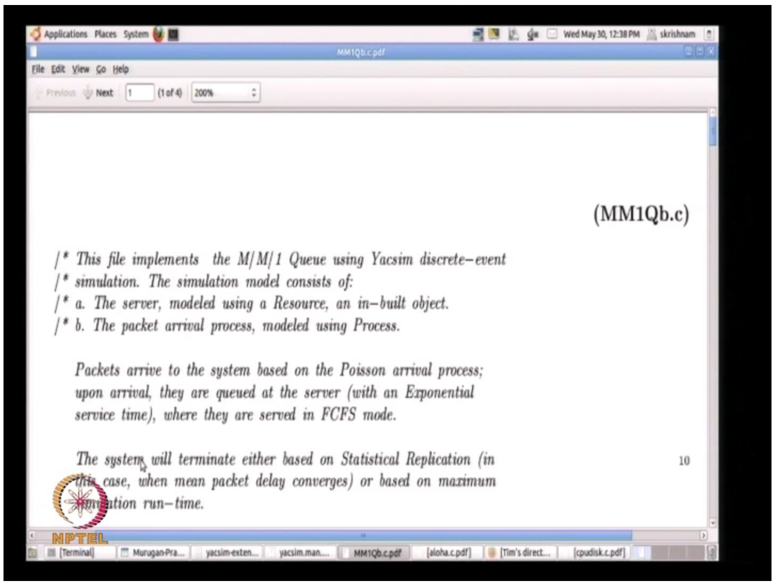
This interrupts the simulator and returns the value 5 to the user.

So, driver run 10.0 which means you can run the driver for the good point 10 unit of time. So, that is the basic thing that will see. If you can reset the driver in between if you want to. That means, you can the start again reset driver and start it again. The driver is basically 0 round zeroed on or bringing down are stratification the statistics record to the original sate and keep quiet going after that. So, that is, these are all the main functions that you will have to require in order to execute the basic stimulation. So, before I proceed, any question from the class? (( )) It is important because you have to think about the So, students who are hearing outside things about learning the outside right And this is a very basic introduction. How does the stimulation time mentioned in the driver run function appear in the real time Sir? That is something that you will have to do the translation right. So, you you have to define one unit of time is equal to say, one microsecond. So, that is something you will have to figure of right. So, that of translation you have to do. And the unless we see an example you wont be able to do it. For example, if you say, that the I am generating eight 30 packets, 30 million packets per second right and correspondingly in that case you will have your…you will be generating , that you will generating, just second as the basic unit of the time. Then your will packet arrival rate, inter packet arrival time will be one over 30 million packets of time, but if you want say that I am not able to make it. And you are going to generate one second interval, you want to you can say that to convert time skill to one unit of the time equal to 10 second or so if you desire decide to do the translation that way.

So, one stimulation time is equal to 10 second that then you will we translate your correspond 30 million packet per second to time 3 million packet per stimulation packet per second unit of time. Because you know, you are condensing the time scale that way contains basic of time skill the page write, that is you will have to try to do that because if you run into systems where we are talking about 1 gbps links and so on. Then what happens is the number of packet that get generating generated is very, very large. And the time between these packets is also very, very small and that leads to that is number of events that is getting created and so on.

So that is, something you have to desire decides at what level of whatever absorption abstraction is okay. Even though is your know system says 30 m b s, it will translate to 3 million packet per unit time. Is that sufficient to give that is that level of accuracy that you are looking for? That is something end you will have to figure out after experimenting and seeing if that is okay. And if at the end you translate that back to. So, do we have to specify this on our code? you to Yes, you would have to specify it in your code, the user parameters terms will be in the parameters of and make up our mega bits per second. It is up to you say lambda says 30 millisecond, in your code you say one unit of time is 10 second. Then in your the system will be stimulation the lambda will be the original lambda is 30 m b s, because your having it on larger gross scale have and it will be just 3 million packet per unit stimulation of time. So, that is on something that you will have to program manually in the beginning of your system.

(Refer Slide Time: 32:10)


(MM1Qb.c)

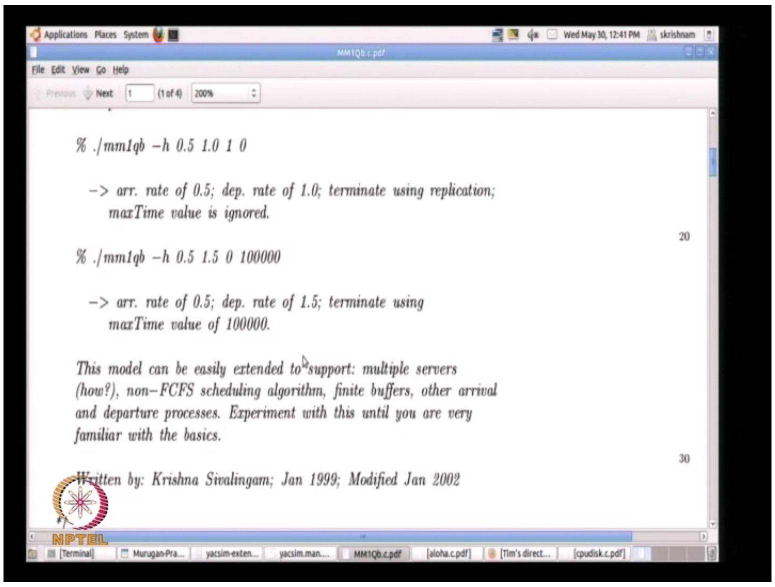
```
/* This file implements the M/M/1 Queue using Yacsim discrete-event
/* simulation. The simulation model consists of:
/* a. The server, modeled using a Resource, an in-built object.
/* b. The packet arrival process, modeled using Process.

   Packets arrive to the system based on the Poisson arrival process;
   upon arrival, they are queued at the server (with an Exponential
   service time), where they are served in FCFS mode.

   The system will terminate either based on Statistical Replication (in
   this case, when mean packet delay converges) or based on maximum
   simulation run-time.
```

So, now, that is the this is basic. Now, let us look at the program right? IS it clear enough? Here it is looking clear enough. You want me to zoom? We can zoom. Is that better? I want the entire screen to show up. So, this the basic M M queue right. Whoever has followed the lecture so far will know what the M M queue is all about right followed You have exponential service time, exponential arrival process, poison arrival process. And there is exactly one server in this particular system. So, how do we build the models? So, you we have seen the theoretical vassals results for M M 1. We know the that delay is one by mu minus lambda right. All those derivation we have seen. How do be we validate that.

We have seen time that there are two days ways of validation; one is where theoretical vassal results are known or none and one is validation with the help of stimulation with real system. We can actually we build a system where the files people come and go and actually pretend that they exponential service times and so on. It is easy actually these days to do it a stimulation based verification. So, this the definition; l Packets arrive on the system are based on the Poison system on the poison arrival process and then they are queued arrive and at the server has So, the service time for the packet is exponential right. And each packet will have exponential service time and we are exponentially servicing packets is in first compression come, first serve mode right. It is simply FCFS or F I F O right as we know that. And this system actually terminates based on two properties; one is run, it will by simply run by maximum unit of time or we can simply converge maximum early on by which I mentioned earlier on.

Looking at the reputation replication method, you can converge. At the time of converge conversion, you start stop the stimulation. We will you not look at convergence initially, converge you we will simply look at stimulation for large amount of time. That is the you specify at that command line.

(Refer Slide Time: 34:02)



```
% ./mm1qb -h 0.5 1.0 1 0

  -> arr. rate of 0.5; dep. rate of 1.0; terminate using replication;
     maxTime value is ignored.

% ./mm1qb -h 0.5 1.5 0 100000

  -> arr. rate of 0.5; dep. rate of 1.5; terminate using
     maxTime value of 100000.
```

This model can be easily extended to support: multiple servers (how?), non-FCFS scheduling algorithm, finite buffers, other arrival and departure processes. Experiment with this until you are very familiar with the basics.

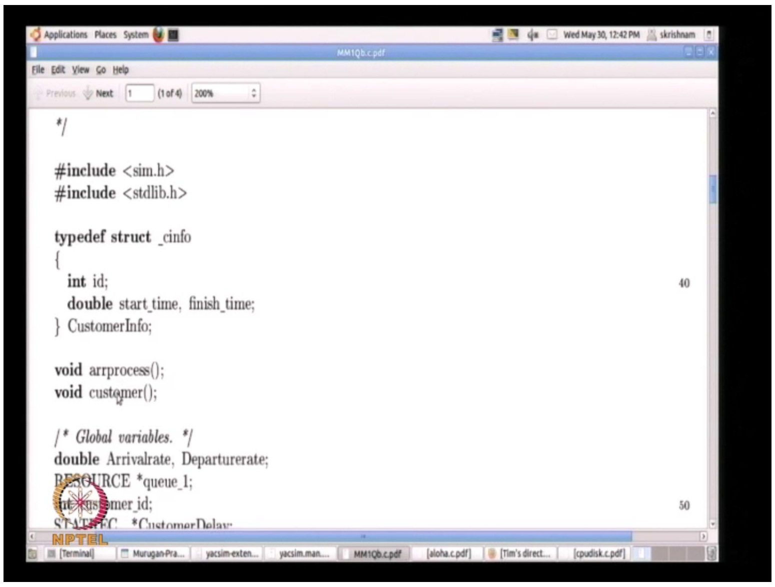
Written by: Krishna Sivalingam; Jan 1999; Modified Jan 2002

So, the basic thing to run this code is as in follows right. So, this is the executable remain minus thing action header the axiom thing which says do not print some headers. Otherwise your axiom prints some useless headers. And parameters are basically 4 parameters. This is the your lambda in your mm 1 for 1. So, this point 5 packets for unit of time whether it this is second or whatever, we do not care. It is point 5 units of time. Service rate or departure rate; this is mu, departure unit This is 1.0 packet per unit time and if I set this to one, it means terminate using replication. are Or if I simply look at the other way, last value will be ignored if where use the same way this to one or if I simply look at the other way, same thing right. Point 5 is the arrival rate point, 1.5 is the departure rate. If the third parameter will is set to 0, it means simply run the this for the process specified command line. So, this is the basic way we are going to run this system and it is off course and it is very easy to extend this to multiple servers which I will post in the assignments later on.

Therefore, I will tell you how to write related on various set of programs. And initially also, this particular information implementation can support finite buffers, infinite buffers and all those variations are also possible. support in finite purpose And also we can easily change this; the duty beauty of stimulation is that, if you want exaction the exponential service time with and some other service time, it very simple. I just need the random number generator to be implemented. And if you look at Raj Jain's book, there are ways of stimulating other kinds of variable, very straightforward. input there way of stimulation And likewise arrival process also right. And sometimes you find that the arrival process is neither Poisson nor exponential. other kind of random variables and like base arrival process also right the sometimes you find the process arrival process exponential In that case, look for, you can simply encode whatever that we you want.

So, that gives you more realistic view of the system it was realize of the basic time,. But you always use the corresponding theoretical resources for mm 1. Because for mm 1 is easy easier to get theoretical results for, significant code off course there is g g 1 for in Raj Jain's book, there is g1 there is mg1. is the All those results are system are available and we find try to combine. And g g 1 right, combine system which is hard ware harder to find. Those specific close formulations right; anyway this is basic program. You can, you should right, try to play around with basic change of basic departure process, arrival process, everything and see how a system can behave run for on time okay. And this is been running for a long time.

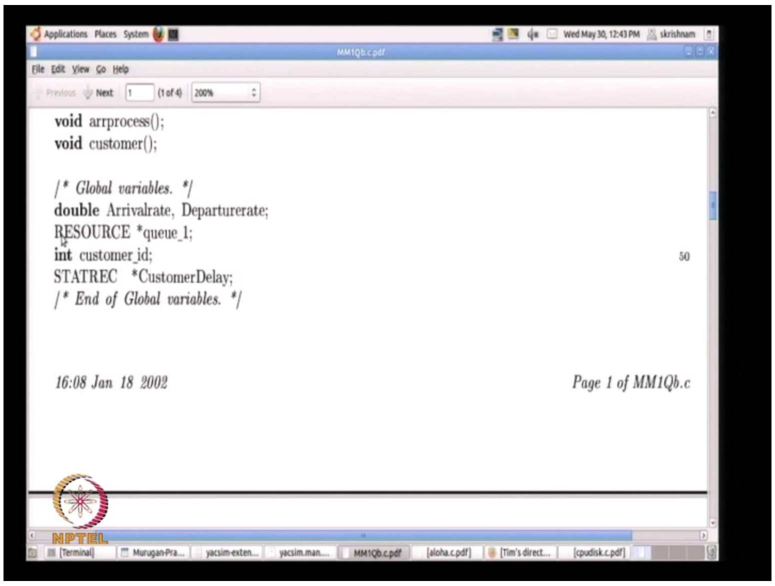
(Refer Slide Time: 36:24)



```
*/

#include <sim.h>
#include <stdlib.h>

typedef struct _cinfo
{
  int id;
  double start_time, finish_time;
} CustomerInfo;

void arrprocess();
void customer();

/* Global variables. */
double Arrivalrate, Departurerate;
RESOURCE *queue_1;
int customer_id;
STATREC  *CustomerDelay;
```

So this is for, the only thing is unique program is in this axiom program, you need to include this, m dot h. This is supplied by or specify supplied by the action axiom library. The sim dot h is the only thing you have to include, is enough everything else is right whatever our you want to do. And by default we create a stature structure, c info that sources stores the start time in to and the finish time. And every time the customer arrives, we give the customer, need id number, token number. We record the customer time that the customer arrived and the time that at the customer finished. That is this want is basically used to store here. Now, the arrival process and the customer; and so this these are the functions that are there going be associated with two different activities. So, the arrival process is going to be associated with the arrival process even that processes is going to be created later related on and customer is simply another logic that is going be associated with the departure event or departure process. So, this is the two prototypes for your two functions. So, you have to simply and default by default it will be voice void related on this function, corresponding to these functions.

(Refer Slide Time: 37:24)

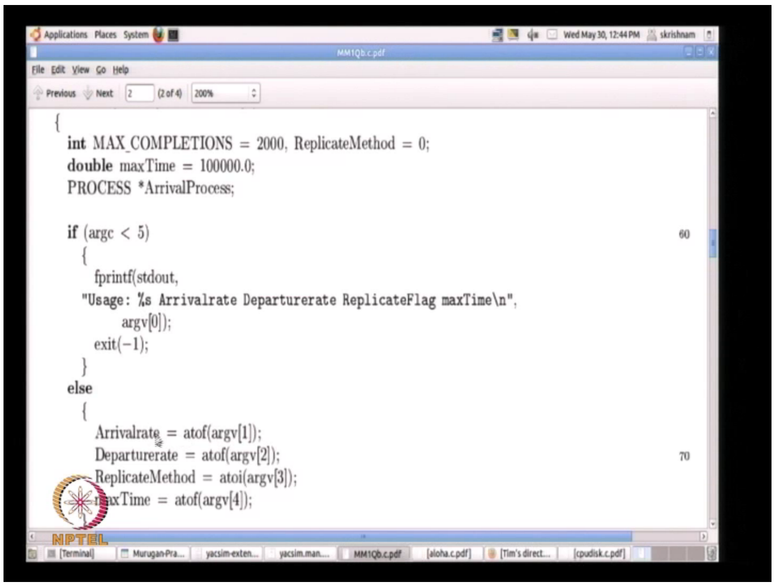


```
void arrprocess();
void customer();

/* Global variables. */
double Arrivalrate, Departurerate;
RESOURCE *queue_1;
int customer_id;                                      50
STATREC  *CustomerDelay;
/* End of Global variables. */


16:08 Jan 18 2002                           Page 1 of MM1Qb.c
```

The global variables, we need not use global variables. But sometimes in when you are sharing data between processes, it is easy as easier to use global variables. But you should remember that, when you rating writing a multi processor simulation and use global variables you have to thing think about synchronization, global variables You would need actually whenever a single variable is shared between the multiple processors, you need remember holes semaphores or locks for synchronization purposes.

This probably thus does not have that but then other simplified complication that is to simplify. Otherwise it is too complicated. Anytime, any particular variable is going to updated by two different processes, you definitely need, the corresponding synchronization process constraints. In the case, arrival rate or departure rate, both the static values. They are not going to be changed. So, therefore we do not need that. Then this resources is again the queue right, again the queue going to be the packets into this particular system; the customer delay, the static variables and the Statrec. So this customer delay is and statistic record that is going to this keep all the track of all the packets that have been executed. So every departure process, we will record the delay taken by the packet or customer in given variables. So, these are this are the four more different variables that are primary necessary. So that is first part of the code this is write there is first part definition.

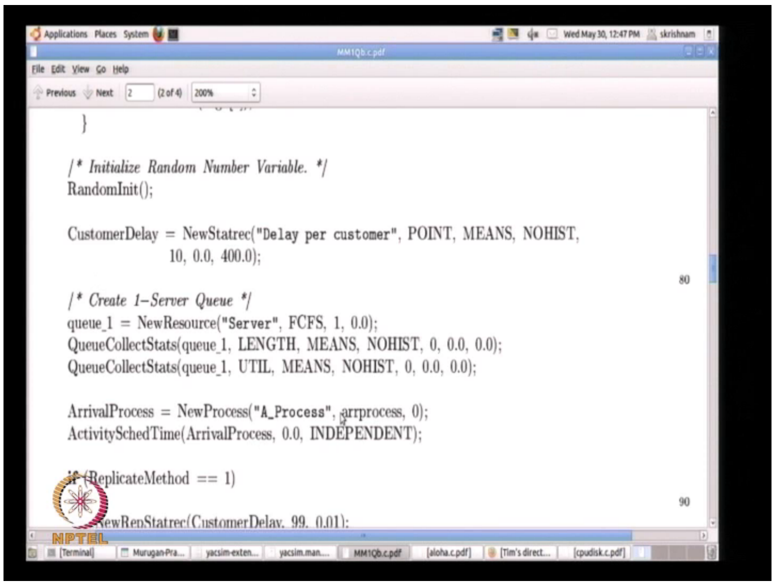
(Refer Slide Time: 38:50)



```
{
int MAX_COMPLETIONS = 2000, ReplicateMethod = 0;
double maxTime = 100000.0;
PROCESS *ArrivalProcess;

if (argc < 5)
  {
    fprintf(stdout,
  "Usage: %s Arrivalrate Departurerate ReplicateFlag maxTime\n",
        argv[0]);
    exit(-1);
  }
else
  {
    Arrivalrate = atof(argv[1]);
    Departurerate = atof(argv[2]);
    ReplicateMethod = atoi(argv[3]);
    maxTime = atof(argv[4]);
```

Let us move on to user main. I mentioned that the user main is the starting point of action program right because main is already defined in the axim in the axim definition. So, there are some simply default values. So far max time is what never we normally use for terminating number of stimulation has you to run. And max time we can update for what we obtain from the command line. So, what we the arrival process is the process I am going to create. These are the command line arguments we are to read. The first argument is going to be arrival rate, second in is departure rate and so on. And max time again a to f. So, that is your standard convergence of ASCII to float or integer ok.

(Refer Slide Time: 39:36)


```
    }

/* Initialize Random Number Variable. */
RandomInit();

CustomerDelay = NewStatrec("Delay per customer", POINT, MEANS, NOHIST,
                10, 0.0, 400.0);

/* Create 1-Server Queue */
queue_1 = NewResource("Server", FCFS, 1, 0.0);
QueueCollectStats(queue_1, LENGTH, MEANS, NOHIST, 0, 0.0, 0.0);
QueueCollectStats(queue_1, UTIL, MEANS, NOHIST, 0, 0.0, 0.0);

ArrivalProcess = NewProcess("A_Process", arrprocess, 0);
ActivitySchedTime(ArrivalProcess, 0.0, INDEPENDENT);

if (ReplicateMethod == 1)

    NewRepStatrec(CustomerDelay, 99, 0.01);
```

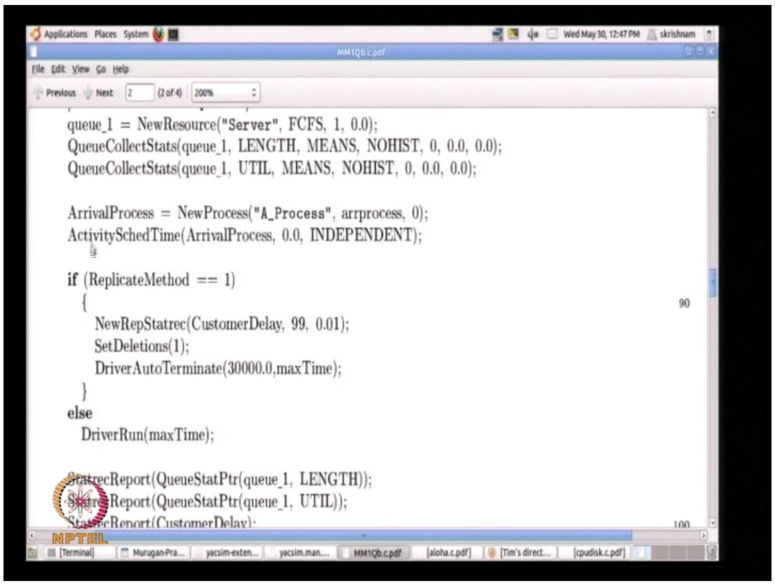
So, this is very important right. The random minute, you need random variables. unique Anytime you run the stimulation right you have to initialize the random variable, you have to call this random variable and so on. Even if you are using you always called SRAM or Srandom, you always call the Srandom. Because when you have multiple runs of same time the same system, you want to make sure that right there are seeds. That initially, the randomness, way of packets are generation generated or the (( )) thus does not affect of the behavior of the system. You want to make that randomness has no significant impact on the system behaviour.

In fact, on system behavior that is why this random minute is there. There are also other associated functions like Set stream is there. But we will not get into that. So, the customer delay is the statistic record that you have will creator created. So, this is the function called, New Statrec, the simply name when we use for debugging purpose or diagnostic purpose. So, that is straight forward. Queue is again, my new resources. So, resource function is called. So, I am creating the server, so the name of that is simply server. Remember that, the scheduling discipling, the queuing discipline in FCFS, I am creating it with only one sever.

And, the last parameter not relevant and this is the slice time that we saw earlier. That is all, that I have a will single server running, FCFS schedule in discipline. Done, I also collect statics on this queue based on the length on net work of the queue. This is particular enqueue in terms of the links of the queue. When you compute that, this is server utilization. What is the fraction of time the sever is being utilized. Actually, be utilities in mm 1 we say row equals lambda by mu, but that is definition. I would actually measure this utilization, how often is this sever to be being utilities utilized. You actually measure that and you record that. And that this called valid is to validate that where utilization that actually equals to lambda by new mu. That is what this is for. And then this is your basic. So, the sever will is being created. One particular statistical record has been created. Now, we come to the main point part of the code. The main part of the code is creating the arrival process. So, remember we saw arrival process, so that is the name of process; and this is the function that contains, the logic for this particular process.

So, again this is only this the definition. If I simply live leave the arrival process in new process; that will simply create a data structure called as the arrival process, the new process. But you also have to schedule that. So, unless it is scheduled it is not going to get executed. So, the scheduling is important. So, that is your activity in your schedule time which simply takes the arrival point process pointer. And then 0.0 means that says that this is going to start write right away. We schedule the process to start running write right away. Independent is the parameter, which will skip for now. So, all that we away say is we have all created the process, which will start running straight away. Then what will happen? The system simulator will create the event at the time 0.0 which will, the event or a process with the process will be created to start executing in the sequins time 0.0, and that will in turn run the code that is given in this particular purpose process, of the arrival process.

(Refer Slide Time: 42:33)



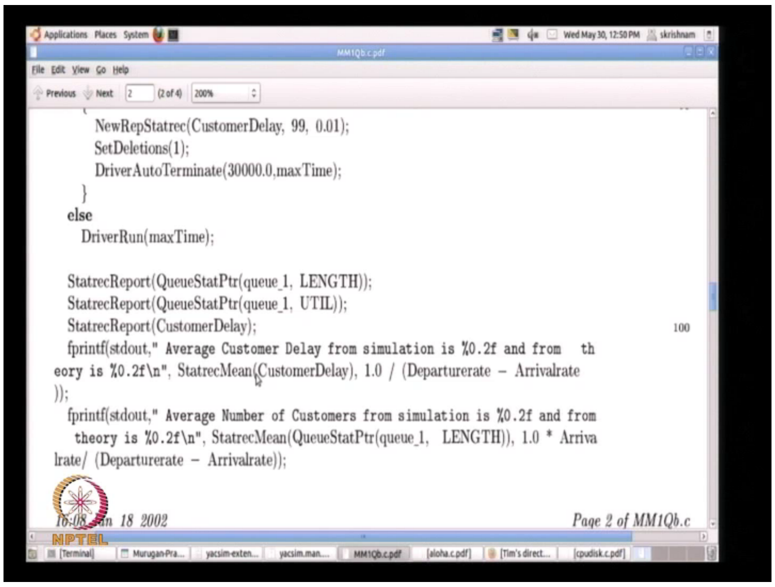
```
queue_1 = NewResource("Server", FCFS, 1, 0.0);
QueueCollectStats(queue_1, LENGTH, MEANS, NOHIST, 0, 0.0, 0.0);
QueueCollectStats(queue_1, UTIL, MEANS, NOHIST, 0, 0.0, 0.0);

ArrivalProcess = NewProcess("A_Process", arrprocess, 0);
ActivitySchedTime(ArrivalProcess, 0.0, INDEPENDENT);

if (ReplicateMethod == 1)
  {
    NewRepStatrec(CustomerDelay, 99, 0.01);
    SetDeletions(1);
    DriverAutoTerminate(30000.0,maxTime);
  }
else
  DriverRun(maxTime);

StatrecReport(QueueStatPtr(queue_1, LENGTH));
StatrecReport(QueueStatPtr(queue_1, UTIL));
StatrecReport(CustomerDelay);
```

Let us ignore the (( )) method for the time being session and then all the day So, now that all that I've wanted is now in place. all object of the day Now, the important point is to get the stimulation run going and that is other driver run. So, the driver run max time, and this max time if you I want to change it from the default value, I can specify max time command line argument. Given, and so it will simply change that.

Now, what happens is, now look at the code, this this is it the right. I have created the code, I have created the objects, I create the process. I have driver run access max time. So, what happens is what happen after that driver run when it returns, it menas that the stimulation of is finished. And it is not very obvious, but that is what is happening. At the end of driver run, when the driver run returns it means that the stimulation is run for max time units.

Then what will we do? We want to print out of all the statistics. So, I simply called stare Statrec report for queue one because all the it has already been collected. it was static I do a statrec report collect on the utilization on on customer delay and so on. This is not needed for all the stimulation. I am using this to show how we can actually invoke these particular things. And then we will then look at finally, how do we print, what do we ant to be print. So, I am looking at customer delay right. There is default function is stare Statrec mean customer delay. This will know, is not automatically compute the main for you. So you do not have to go on figuring all the values as long as we have done the So want as long as corresponding updates correctly. values So, this is what is going to happen.

(Refer Slide Time: 43:55)

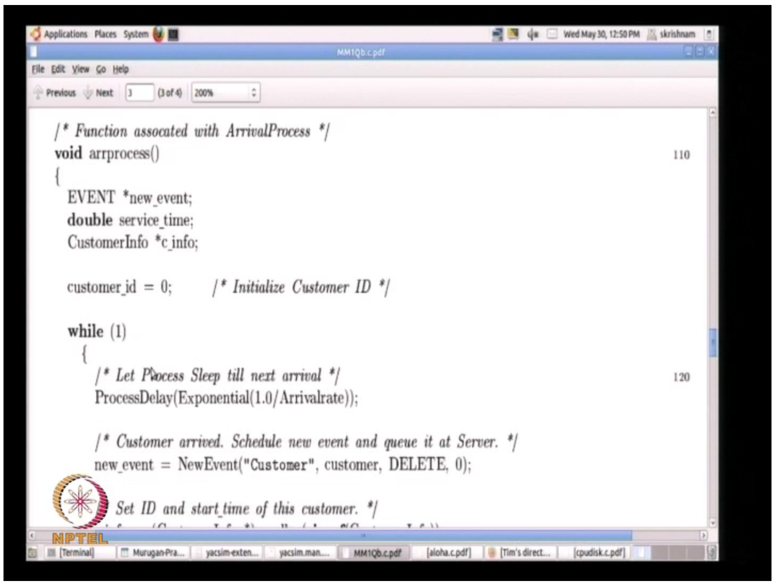


```
      NewRepStatrec(CustomerDelay, 99, 0.01);
      SetDeletions(1);
      DriverAutoTerminate(30000.0,maxTime);
    }
  else
    DriverRun(maxTime);

  StatrecReport(QueueStatPtr(queue_1, LENGTH));
  StatrecReport(QueueStatPtr(queue_1, UTIL));
  StatrecReport(CustomerDelay);
  fprintf(stdout," Average Customer Delay from simulation is %0.2f and from   th
eory is %0.2f\n", StatrecMean(CustomerDelay), 1.0 / (Departurerate - Arrivalrate
));
  fprintf(stdout," Average Number of Customers from simulation is %0.2f and from
   theory is %0.2f\n", StatrecMean(QueueStatPtr(queue_1,  LENGTH)), 1.0 * Arriva
lrate/ (Departurerate - Arrivalrate));

16:08 Jan 18 2002                                              Page 2 of MM1Qb.c
```

Then we try to do compression comparison between what the theory gives us, what is does the theory say? The theory says that mm 1 departure delay is simply one over new mu minus lambda; departure minus arrival rate right. We want are comparing what the stimulation gives us at to the value given by at the theoretical formula. That is the first statement here. This is e of t, the delay value. I can also measure e of n. Average number of customers in the system also can be measured right. And what is that? This is simply is one over sorry this is simply lambda divided by new mu minus lambda right, (( )) formula remember?

Lambda into to delay gives you the number of customer. And and I can also get the number of customers from the stimulation value stare Statrec mean value right. So, this says, this is mean value right; is I am looking at link length of the queue. What is the number of packets queued in that particular queue over time average. So, at every point, every point 0 units of time I look at 0 the length and record that. So, that is why time average link e of n value; this is E of n from stimulation; this is e of n from what theoretical class studies derived in class earlier on. This is the way may be to compare theory to practice. So, what is this stimulation giving you? What you are looking for?

(Refer Slide Time: 45:15)

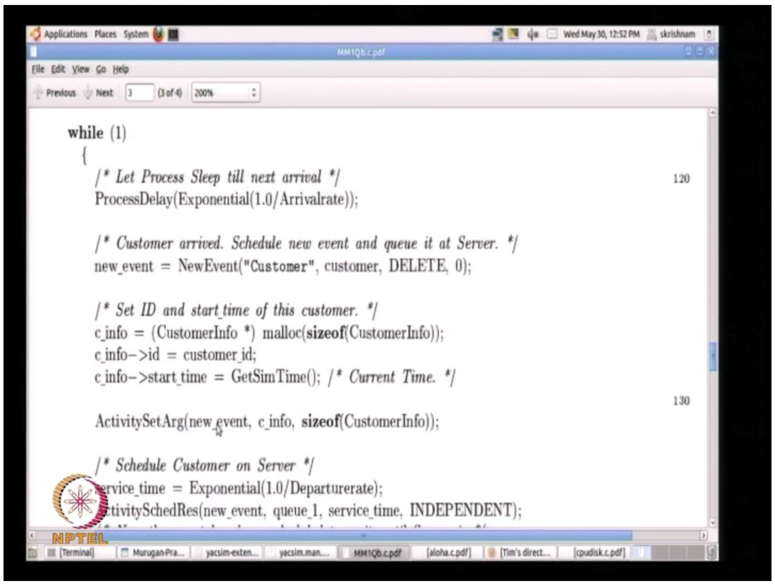


```
/* Function assocated with ArrivalProcess */
void arrprocess()
{
  EVENT *new_event;
  double service_time;
  CustomerInfo *c_info;

  customer_id = 0;        /* Initialize Customer ID */

  while (1)
    {
      /* Let Process Sleep till next arrival */
      ProcessDelay(Exponential(1.0/Arrivalrate));

      /* Customer arrived. Schedule new event and queue it at Server. */
      new_event = NewEvent("Customer", customer, DELETE, 0);

      /* Set ID and start_time of this customer. */
```

Now, let us look at what are arrival process is going to do. So, arrival process is basically a simple process. It is the endless loop. So, the arrival process is the process whose job is to keep on reading generating packets for even ever until the stimulation terminates. So, what we do is So, we the create the first process write right away. But we have special notion of process delay. Remember, we talked about the process delay.

So, one what I do this is, I can delay the process. How long do I want to delay the process? I want to delay it till the process next arrival time. So, inter arrival time of a Poison arrival process is disturbed exponentially. We know some of that from our theory right. So, if the arrival rate is lambda, if the departure sorry inter arrival time is exponentially disturbed with parameters with one over lambda; that is again something with is which we have covered in class right. Therefore, I want delay arrival process by a random arrival of time. And the random the random randomness is defined by this exponentially variable with parameter one over lambda. So, which been that, will with a period of time 0.02 seconds, nothing is happening. There is no packet generating.

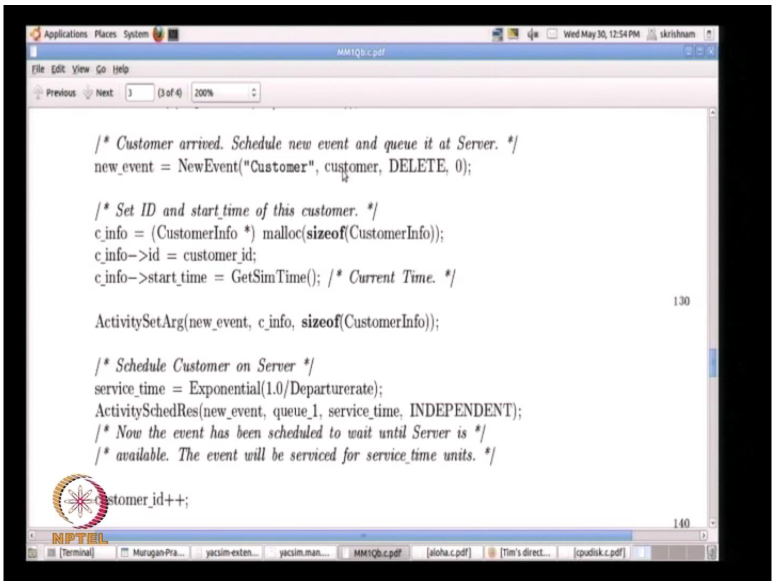
(Refer Slide Time: 46:21)



```
while (1)
  {
    /* Let Process Sleep till next arrival */
    ProcessDelay(Exponential(1.0/Arrivalrate));

    /* Customer arrived. Schedule new event and queue it at Server. */
    new_event = NewEvent("Customer", customer, DELETE, 0);

    /* Set ID and start_time of this customer. */
    c_info = (CustomerInfo *) malloc(sizeof(CustomerInfo));
    c_info->id = customer_id;
    c_info->start_time = GetSimTime(); /* Current Time. */

    ActivitySetArg(new_event, c_info, sizeof(CustomerInfo));

    /* Schedule Customer on Server */
    rvice_time = Exponential(1.0/Departurerate);
    tivitySchedRes(new_event, queue_1, service_time, INDEPENDENT);
```

So, then actually now I am creating a packet and I the packet creation is given again an event. So, now, I am using event examples here right. So, event is I am creating the customers and I am associating this customer function to be executed whenever this event finishing finishes. And I am filling customer id, start time. So, getsim time is a this function defined in the axim and action tells you the current stimulation time, not current time real stimulation time. This is the time when this packet was created and that is what we are recording here. Then we activate Setarg, I am going to all skip that. It is the statement and simply to associate with this data stature structure with this and then particular event. And then So, later I want to retrieve this event, all that I want to, is So, its simply stored in c some convenient place. So, we store this C info as a part of data stature structure and associated with the event.

(Refer Slide Time: 47:09)



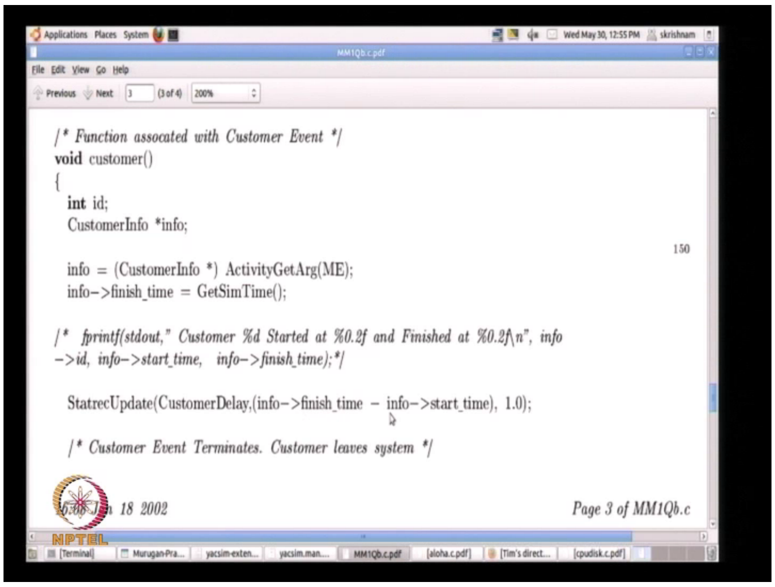
```
/* Customer arrived. Schedule new event and queue it at Server. */
new_event = NewEvent("Customer", customer, DELETE, 0);

/* Set ID and start_time of this customer. */
c_info = (CustomerInfo *) malloc(sizeof(CustomerInfo));
c_info->id = customer_id;
c_info->start_time = GetSimTime(); /* Current Time. */

ActivitySetArg(new_event, c_info, sizeof(CustomerInfo));

/* Schedule Customer on Server */
service_time = Exponential(1.0/Departurerate);
ActivitySchedRes(new_event, queue_1, service_time, INDEPENDENT);
/* Now the event has been scheduled to wait until Server is */
/* available. The event will be serviced for service_time units. */

customer_id++;
```

So, now, the event will be creator has been created. The as packet or customer has arrived or the event get is now getting created, has been created. And what is next thing do? It is to schedule this event. Where do I what is the schedule event? The schedule on the server queue one that I created right. So, the service time again for this packet is exponential and out of from our definition and we saw that the departure rate is mu right. new Now, the service time one over mu is the parameter that is passed to this exponential variable. Now, the service time is also going to be automatically exponentially generated. Now, I know what the service time is; now I automatically schedule the event right to run on queue one with service time which is this is the basically the length of the packet time of service for the customer.

So, I am have created the this particular customer and I schedule the this particular customer. Now, the queue one is empty; what happens? This particular customer will get serviced right away. And the customer finishes the service at the time after the service time for the particular customer is over, this customer function that we defined here that is going to get called okay. That is be that is the basic idea. Again, as soon as this packet is finishing the service, at the end of finishing the service, this event will actually take place. So, I am creating an event that does not take place right away. I am simply putting queue of the server. When the sever selects the event or this packet of for service, and after that finishing service of the packet, at the time this event is actually said to occur that is way of action how this axiom definition is…

So, at the time of occurrence of that event, you called the corresponding event handler which is defined in this function called customer. So, this customer function will contain all the things need for recording end the departure of the particular customer. All are our statistics are complete when the customer with lead leaves the system right, entire (( )) simply store storing the start time. Halt time, this is basic logic given in this arrival process and then this goes on. The, what happen after this customer will is generator generated? When is the next customer going to generator generated? Again, based on this random inter arrival time. So, this is the loop that we have. So, every And this is every inter arrival time is basically exponential distributor. So, you will be packets uppering appearing in the system at periodically periodic points in time. They all get queued in sever. That is all we try to do.

(Refer Slide Time: 49:35)



```
/* Function assocated with Customer Event */
void customer()
{
  int id;
  CustomerInfo *info;
                                                              150
  info = (CustomerInfo *) ActivityGetArg(ME);
  info->finish_time = GetSimTime();

/* fprintf(stdout," Customer %d Started at %0.2f and Finished at %0.2f\n", info
->id, info->start_time,  info->finish_time);*/

  StatrecUpdate(CustomerDelay,(info->finish_time - info->start_time), 1.0);

  /* Customer Event Terminates. Customer leaves system */

16:00 Jan 18 2002                                           Page 3 of MM1Qb.c
```

Then, what is the customer? The customer thing is very simple. In the info data stature structure associated with the particular event, I store the finish time, which is basically the time will that this packet finished service. That is given by GetSim time. So, again this is actually then again something image this is something you have to imagine right. This is not a sequential program. These are all things happening in different points in time right. And At some future point in time, this packet will finish service. Therefore, GetSim time some time will be the time when this packet has finished service. And then simply apply update the customer delay, Statrec delay chat with update with info finish time minus info start time. That is all. Wait is weight again ignored above. They are more complicated. This simply says, the sample value for this new sample value is info finish time minus info start time. The customer event terminates; the customer leaves the system. Now, on the automatically, this is de queued from the server; all those things, we do not have to worry about. All the packet has arrived, the packet has been queued, will queue the packet as been serviced and we have only captured the departure services statistics in this particular system. That is all. That is the main part of the code.