Real-Time Systems

Prof. Dr. Rajib Mall

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Lecture No. # 30

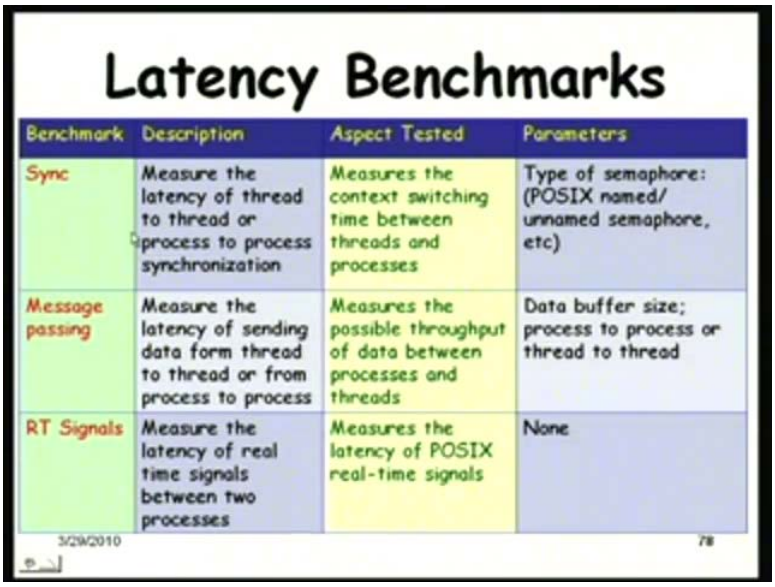
Benchmarking Real-Time Computer and Operating Systems (Contd.)

So, let us continue with what we were discussing last time; we are trying to discuss some ways to select a computer system or a real-time application, embedded real-time application, how does one go about. So, basically the task consists of finding the parameters that are of interest, and then, evaluating different systems based on those matrixes.

Unlike general computer systems, where the, we have a benchmarking program, and the manufacturers, they quote with respect to those benchmarking programs; here it is not so simple; we have to consider several parameters, and then, check the performance of the computer with respect to those parameters, and we had just discussed about determining the static parameters, I mean, the deterministic static parameters, and then, the latency parameters.

The deterministic parameters we are saying that, whether the system behaves deterministically across different calls, timers are deterministic, etcetera, and then, we are just starting to discuss about the latency aspects; so, let us proceed from that point.

(Refer Slide Time: 02:01)

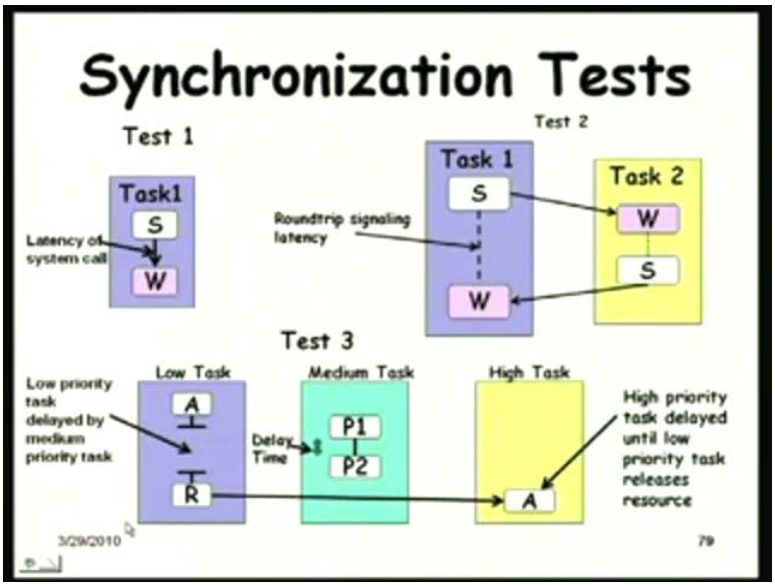


So, we just start to discuss this one, the latency benchmarks, where we will consider three parameters the Sync, Messaging passing, and RT real-time signals. So, the Sync, here we will measure the overhead due to synchronization or the latency, that is introduced due to the synchronization primitive, the simplest case will be, of course, that when a process request for a resource, which is set declared as a semaphore.

So, how much time does it take to acquire it? We will also consider Message passing, which measures the overhead of sending data from one thread to another or from one process to another, depending on the operating system, because some operating systems support only threads on a process.

And then, there are multiple processors, and also threads, I mean, data from thread of one process to thread of another process, and here the important parameters that we need to consider or size of the data, data buffer size, and then, the real-time signals which measure the latency of the signal, once a signal is generated by one task, how much time does it get take for the signal to get reported at the recipient.

(Refer Slide Time: 03:48)



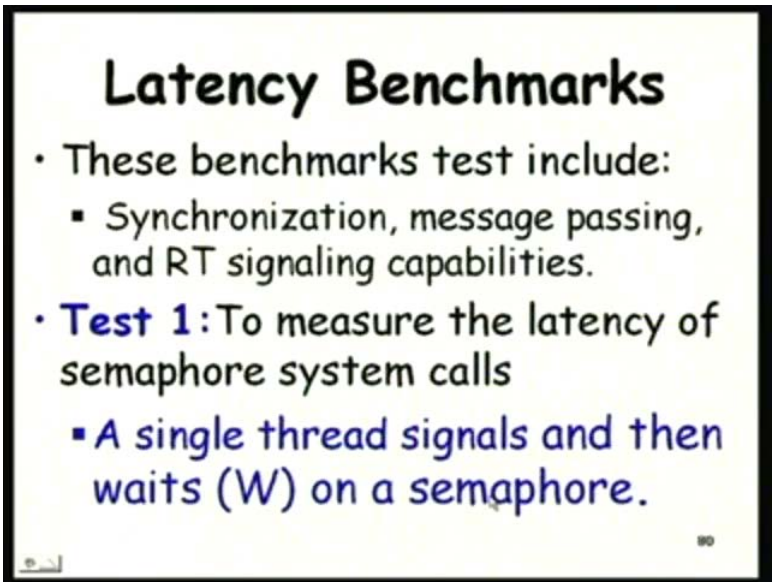
So, these are the three tests, that will discuss, one is that task requests for a resource, and then it waits for the resource, the resource is being used, and it undergoes a context switch. So, through a system call a task, requests a resource, and how much time does it take for the context switch to occur.

And here in the test 2, what we will do is that, we will have two tasks, and task 1 request a resource being held by task 2, and task 2, we will request for a resource held by task 1. So, this is basically the round trip signaling latency, as you can see that task 1 is already holding resource, and task 2 is holding a resource, and it just calling that, and the third test that we will do is a low priority task, is as acquired a resource here r, and then a high priority task has requested for the resource.

So, how much time the high priority task will take to acquire the resource, the high priority task will be delayed, until the low priority task will release the resource of course, but what about the medium priority tasks, if there are many medium priority task coming out, which do not need the resource then will it vary, so we have to test that.

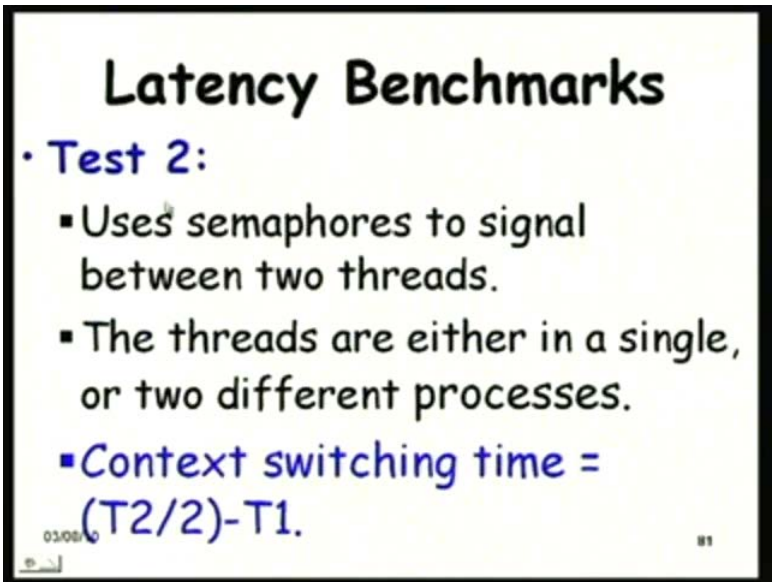
So, this third test is we have a low priority task holding a resource, and then high priority task requesting the resource, and in between there will be medium priority tasks which are executing, and what is the impact of that on the high priority task getting the required resource?

(Refer Slide Time: 06:12)



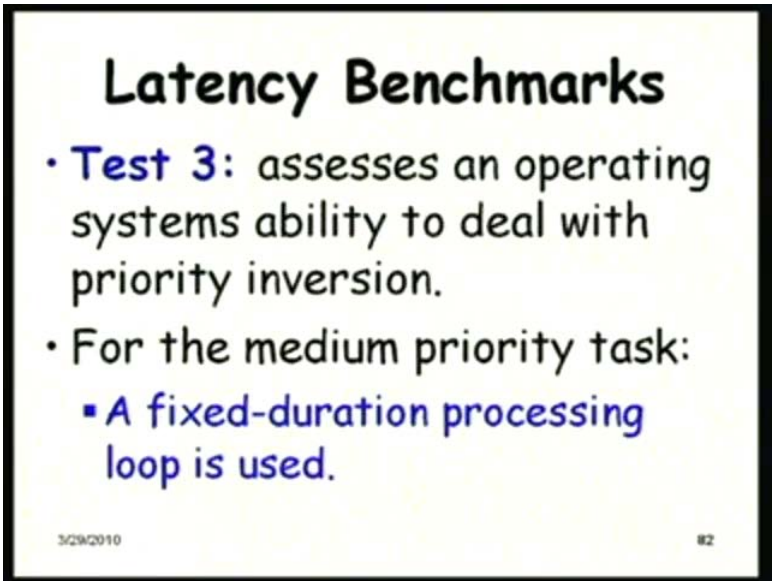
So, in the test 1, we measure the latency of the semaphore system call. A single thread or a process waits for the semaphore; it makes a system call and waits for the semaphore. So, basically, if you see here a single process makes a system call, and it is not available, it just waits for it, and naturally, it will undergo a context switch. So, how much time does it take for the context switch to occur? The test 2 uses two semaphores to signal between two threads, and here the threads are either in a single or two different processes.

(Refer Slide Time: 07:08)



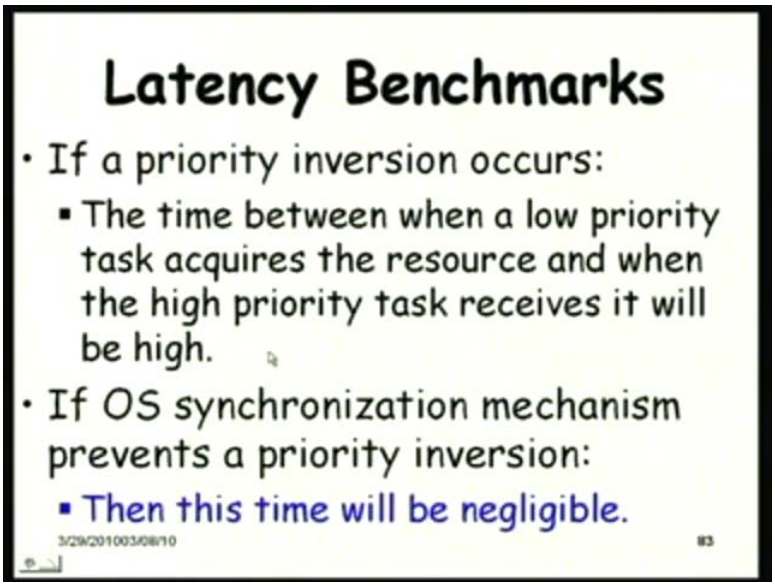
So, here if we consider the test 2 as T 2, the time taken for the test 2 is T 2, then the context switch time is basically T 2 by 2 minus T 1, because this is a, there are two, there are two calls involved here, there is a round trip signaling latency is involved here, and on the test 1, we are just it had requested one and then undergone the context switch.

(Refer Slide Time: 08:10)



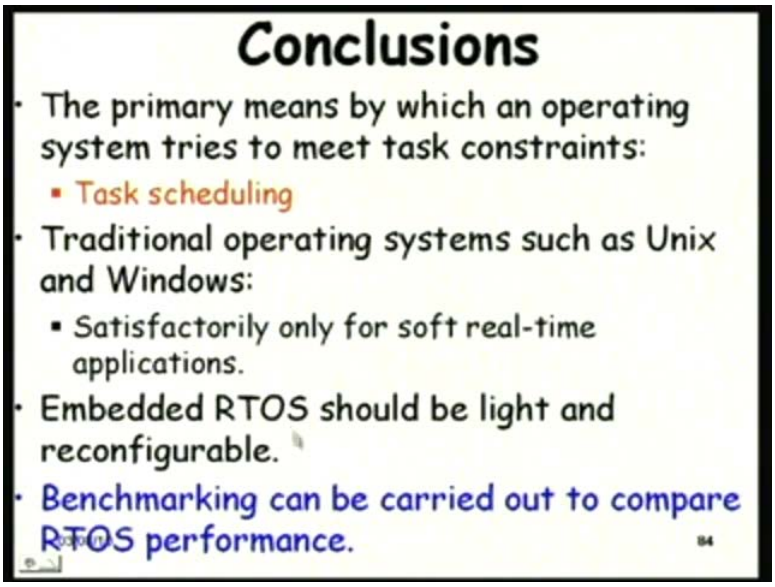
The test 3 it assesses the operating systems ability to deal with the priority inversion, and we had discuss this one, that it just low priority task is holding the resource, and high priority task is requesting it, and in between there are medium priority tasks which are generated. We can use a fixed duration processing loop to generate all this medium priority tasks, and then check whether there is a difference in the time with respect to the high priority task getting the resource, because the low priority task is takes a constant amount of time for its processing, and the high priority task should take that much time only to get the resource, but if we find that it varies depending on how many medium priority tasks run etcetera, then we know that, it is not handling the unbounded priority inversion.

(Refer Slide Time: 09:18)



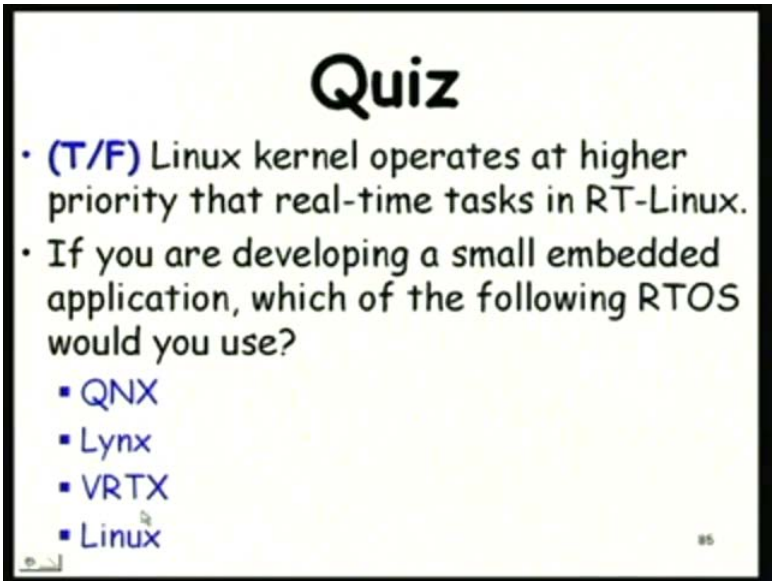
So, if priority inversion occurs, the time between the low priority task acquiring the resource, and the high priority task receiving will be high. If unbounded priority inversion occur, then the time of the high priority tasks, the time at which the high priority task receives the resource the delay it incurs, getting the resource will be high, but if the operating system as a mechanism to prevent priority inversion, basically the inheritance mechanism or the ceiling protocol or something, then the time will be very small. So, this also we can measure through a simple experiment.

(Refer Slide Time: 10:02)



So far we have been looking at the operating system issues, we will looked at scheduler task scheduler is the primary means through which a system tries to meet the constraints the time constraints, and then, we had seen that traditional operating systems are for soft real-time applications, but for hard real-time applications, we cannot use Unix and windows traditional Unix, and windows, we need to use real-time operating systems, and then, we had seen various types of real-time operating systems, and then, we had also mentioned about operating systems required for embedded applications, and then, we had discussed about how to compare the performance of real-time computers and operating systems through benchmarking.

(Refer Slide Time: 11:02)



Now, just look at a few quiz, before we proceed to the next topic, let us, let me just ask you few questions. The first question is a true false question, the question statement is that, the Linux kernel operates at a higher priority compare to real-time tasks in RT Linux.

(( )).

What, what do you think.

(( )).

Why, why is that, so, what about others.

False.

False, why is that.

(( )).

What is the justification?

(( )) because then only the real-time task and (( )).

Any, any other answer.

(( )).

So, the way RT Linux runs is that, once on a Linux kernel you install RT Linux, it gets all the interrupts reported to it, is not it, and also the traditional Linux runs says the low priority task, and the real-time priority tasks run at a much higher priority, and it ensures the preemption of Linux priority, a Linux kernel by this arrangement.

So, irrespective of whether the actual code is preemptible or not, it becomes a task of the RT Linux, and therefore, it just can preempt it anytime, even if the kernel might be masking the interrupt, whatever it may be doing, it need not change the Linux kernel, because it runs with the task of the RT Linux, it loses all its power to get the interrupts reported, it is reported by a RT Linux is that ok.

Now, let us look at the next question, if you are developing a small embedded applications, which of the following real-time operating systems would you use one is QNX, Lynx, VRTX.

(( )).

Linux, so, why VRTX?

(( )).

You mean the size of the operating system.

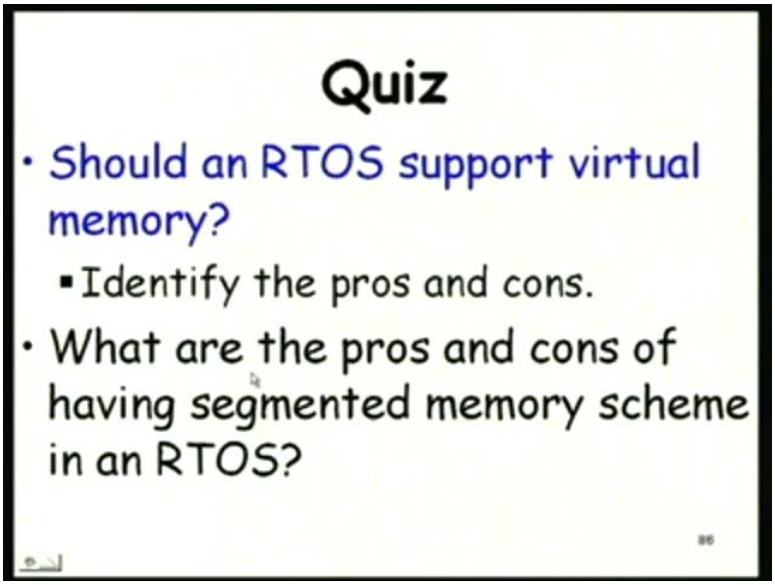
(( )).

Any, any other answer size is the answer, but see here VRTX is the host target operating system, QNX, Lynx etcetera, these are self host systems.

So, we cannot really run them on the target board right, QNX you cannot run it here. So, if it is a target board you are considering, then you cannot test them with respect to QNX, one is you cannot load it, and you also does not support a target board testing is not it.

So, here VRTX is a host target system, where you develop the application run it on a host system, download it into an embedded board or a system, and chip, and then test it. So, both size, and also whether running it on a target board and testing is supported.

(Refer Slide Time: 14:55)



Now, see the real-time operating system support virtual memory, we had seen that many real-time operating systems, they support virtual memory, some do not, when we looked at different operating system, the open source in the commercial we had noticed this, but what do you think, I mean, what would be the advantage of having a virtual memory support on a real-time operating system, vis-a-vis what would be the disadvantage of having a virtual memory support in a real-time operating system.

(( )).

No, multi, what do you mean by multi programming.

(( )).

No, that its virtual memory is not a necessity for multiprogramming, no, sorry ,that is not a correct answer.

(( )) the memory require by (( )).

I mean, at what cost, you need a smaller ram size at what cost.

(( )).

No, no, what is the, I mean, it just not just the ram size, you need something else is not it.

(( )).

Let us just address his question.

So, is it just you reduce the ram size by using a virtual memory system or is there something else you need, when you use a virtual memory system, you need a hard disk is not it, you need a hard disk just having a less memory will not do on a, so, you are not reducing anything.

Sir, scoring the programs, we are already have (( )).

No, no, no, see normally none of the embedded applications have a, I mean, none means almost none, very few may have hard disk; usually they have a small flash drive for something, you cannot implement, I mean, on a small flash drive, you cannot implement a virtual memory.

So, that is not a correct answer that.

(( )).

So, one positive is that memory protection between the kernel processors and the processor among themselves is provided through virtual memory, true, that is one, one of the advantages, many advantages may be there. Yes, anybody would like to answer, any, any other points as far as advantage is concerned.

(( )).

So, when you have soft real-time applications or let us say compiler debugger, let us say self host system, it has to have virtual memory support for consider in a self host system, because in a self host system, we had looked at two different categories of real-time operating system, the self host systems and the host target systems. If it is running only on the target board, virtual memory is not a necessity, but if it in a self host system, the application is also develop than it, and also soft real-time tasks, if your system is required to handle that is basically larger applications large real-time applications, then you need a virtual memory support, because compilers debugger or let us say email, networking and world wide web, all those things can run only if you have a virtual memory support, but if you are considering a very small application, then virtual memory is not a necessity, that is also a point larger applications need virtual memory support, because there may be non real-time tasks and soft real-time tasks.

Now, what about the disadvantages?

(( )).

So, one is that the memory access time is not deterministic, of course, that can be reduce through memory locking, he says that, you know his answer is that, when we use a virtual memory, because the page for its may occur. The memory access time can vary greatly depending on whether it is in the ram or it is in the hard disk. So, of course, whenever a virtual memory based real-time operating system is considered, and its posix compliant memory locking is provided, but what about other any other disadvantages.

(( )).

For many applications, virtual memory is not required just like small embedded systems, you cannot really have the cost of supporting virtual memory there, you cannot have a hard disk, and all those processor supporting the virtual memory use a very simple processor microcontroller or something, it would be very difficult to implement a virtual memory in a microcontroller.

So, for very small applications cannot use a virtual memory, any other points, anybody would like to answer. So, let us say look at the next question say, questions slightly

different from the previous one, what are the advantages and disadvantages, what are the pros and cons of having segmented memory scheme in a operating system.

Yes, anybody would like to answer.

(( )).

So, the overhead of translation etcetera is reduced, virtual to physical translation etcetera is reduced.

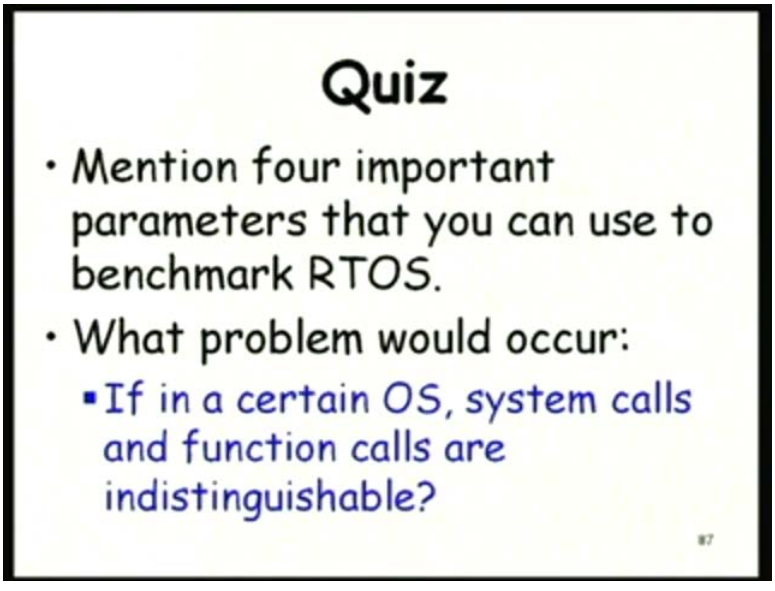
Sir, the number of processors will be limited to those (( )).

So, the number of processors may be limited to the segments, and any, any other points.

Sir, like (( )) into slow the (( )) which segment has been allocated to (( )).

So, may be the implementation of how the segments are implemented that will also determine, and also (( )) of memory in a segmented space, right segmented addressing. So, those are the cons, but usable for small applications cannot use for large applications.

(Refer Slide Time: 23:00)



Now, let us look at another question, mention four important parameters, that you can use benchmark a real-time operating system. So, trying to select real-time operating

systems from various vendors, now what parameters you can use to conductive of benchmark (( )).

Interrupt latency time task switching time.

Data (( )) throughput time.

Data (( )) throughput time, anything else.

(( )).

Dispatch time, and also you can use the deterministic benchmarks, and the latency benchmark, for example, how much time it remains in the kernel mode, and non preempt able, and whether the timers are deterministic.

So, there many aspects we considered, so, just only four asked here, but there are dozens of the aspects which you can consider to distinguish between real-time operating systems depending on what is important for our application. If the, your applications requires timers of one millisecond precision, the same (( )) consideration, then, of course, you will like to consider timer visitor.

Now, there is another question here, what problem would occur if in a certain operating real-time operating system, the system calls and function calls are indistinguishable, operate in the same way.

(( )).

There is no mode change nothing, so system call is the same as a function call what problem can occur?

(( )).

Not clear what protection features.

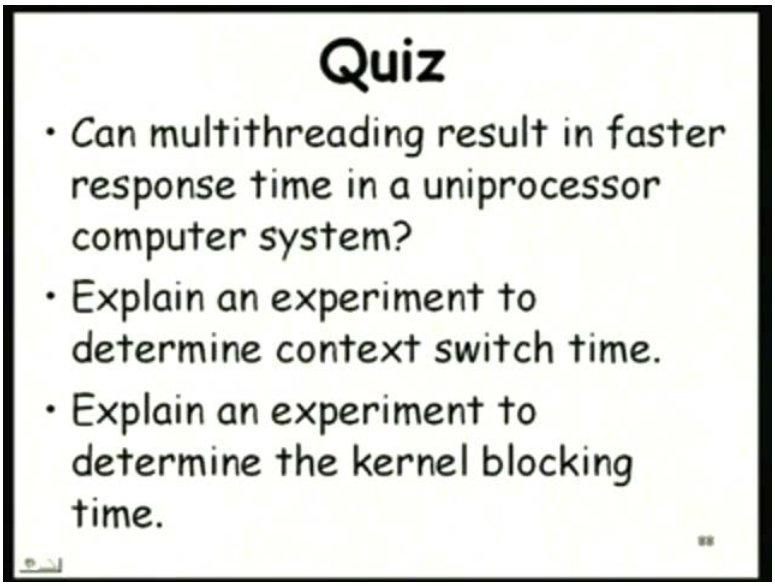
(( )).

So, when there is a system call and a function call, there is it means that, there is only one mode, because the system calls are able to handle devices etcetera in the user mode

itself, see when it is used as a function call, it is still in the user mode, and its able to do all the operations required, then there is only one mode.

And therefore, the kernel can be destroyed, I mean, the kernel can get corrupted while debugging or malicious programs and so on. So, it is basically only the kernel mode exists, no user mode, only the kernel mode exists, then only the system call and function call will be indistinguishable, and in this case developing programs will applications, will become very difficult, and also malicious programs and so on.

(Refer Slide Time: 26:15)



So, let us look at another question, if you have a uniprocessor system can multithreading result in faster response time. See only one processor is there, and if you use multithreading will it result in any faster response time?

No.

So, some are saying yes, some are saying no, let us just think about it little more.

We can have concurrent execution of (( )) mode of code.

How see only one processor is there know, a single threaded processor is there.

(( )), response time.

Yes.

(( )) different, it is the response means, it is not about the completion, it is only about the (( )).

Response, so basically if you have a program running as a multithreaded processor cum program versus a ordinary program.

(( )) it can be (( )) time, because sir if we have a situation (( )), a program has a (( )), and from the initial threads is (( )) due to some.

(( )), something.

So, then, if it was (( )) the response time would have, would (( )).

It will context switch, exactly see the thing is that, the question possibly we need to elaborate a little bit is that, there are concurrent events, that need to handled, just like you are saying that let us say you are doing a processing concurrent processing, where it just undergone a blocking, one thread undergone a blocking, and then another thread can immediately start running, if you did not have any threads, then the entire process will block is not it.

So, here definitely the time, that it takes to run will be much smaller, because otherwise the entire thing would have blocked is not it.

(( )).

No, no.

(( )) throughput not the response.

Same thing.

(( )).

Same thing response time is basically throughput how fast it completes a job, and the job might involve several events and conditions. So, so your, let me just analyze your

answer. So, the answer that you are trying to give there is that, the average response time will improve, but not the worst case response time, is it, is it.

Yes, true, but again see a possibly the question is bit ambiguous here, we could have also described the situation under which we are trying to measure the response time, is it just the task running, and doing disk I o or is it just doing a simple arithmetic computation. So, the question is bit ambiguous is correct, and your answer also, that it can only the worst case response time, I mean, it the worst response time will still remain the same.

Only the otherwise response time improve in a multithreading, because of the blocking etcetera is looks reasonable, but of course, see if you have another situation, where we either implement through processes or though threads, if that was the question, then of course, threading will take less time, if we implement a equivalent solution using processors, threading response time will be faster, is that with you.

So, the question has little bit of ambiguity that is why we have so many different answers for this; now, this is a question explain experiment to determine context switch time.

(( )) the system clock (( )), whenever a (( )) gets slow after before (( )) should be in the (( )), but how, how will that happen, I mean, getting, I mean, how do you determine that when it gets blocked, I mean, just making IO request is it.

(( )).

Making IO request or something, and then how do you know the, for the other task to start running. So, you have two task is mentioning an experiment, where you have two task, one task just reports the time, just before it issues a request for some semaphore or something, and then another tasks as it starts it reports the time, yes possible, but just think of some more experiments, that may not be very accurate time measuring.

(( )).

Sorry.

(( )).

Report, it can be just written to a memory or something, which can be read by another task.

Read the system time, and (( )).

Read the, see basically writes into a memory location one task, another task writes into memory location, a third task reads written reports, because if it prints it, that will take long time, you can just store in the memory location.

Explain an experiment to determine the kernel blocking time, yes, anyone, how, I mean, how good is the operating system with respect to real-time performance or how much time, maximum time it can block in the kernel mode.

(( )).

Just tell me what is the experiment?

(( )), as a system path.

Yes.

(( )).

So, before we showing a system call.

Read the (( )).

Read that time.

Time.

Clock, get time yes and then.

After the (( )) the system call is (( )).

Ok.

(( )), the system time ok.

And then take (( ))

And then.

Take (( )).

But that takes.

(( )).

But that takes the time includes the time the system call takes to complete is not it, see if you just make a system call, and then know before making the system call get the time after system call, you get the time, then it.

(( )).

It also know, it includes basically the how much time the system call as taken to complete, but we are interested in kernel blocking time.

(( )), we can write the kernel (( )).

You mean, you want to modify the kernel.

No, sir like (( )) we can (( )) kernel functions (( )).

No, not clear, the answer is not clear what are you saying.

So, like sir (( )) low level systems (( ))

Yes, yes.

So, why we are, so, scope of modifying the boot loader and that kernel (( )).

No, as showing that see we are not going to change the kernel to measure the kernel blocking time, assume that you have been given operating system.

(( )).

You are running the operating not modifying it, you are not looking at the code, you are running the operating system.

Sir, when we are looking a system (( )).

Yes.

System call there was make a system call, like i o system call.

Yes.

And that is a kernel (( )).

No, no, no I mean how does it.

(( )) kernel mode itself, but it cannot do other job because (( )).

(( )) suppose, you, you are saying that give a time reading, before making IO call and then at the end of the IO call is it.

(( )).

But then it just measure how much time it took to do IO, there is no kernel blocking reported at that time.

(( ))

Sorry.

(( )), before blocking, (( )) after blocking.

No, how do you know in the kernel blocks, see if you had this codes source, code for kernel etcetera, then you can check blocking by finding largest section etcetera, you know that is ruled out, we running the kernel, and then you just describe an experiment using, which you can measure the kernel blocking time.

(( )) (( )).

No, no, no not that complicated, see simple thing we had just discussed few slides back, that you make a system call, keep on making system call, and from another task which keeps on making system calls by the time this, on make system call.

So, the completion of a call will depend on whether the other system call is being processed, if it is blocking, say low priority task is making system call, and you have a high priority task trying to make some simple calls.

Now, if the kernel is blocking, then of course, there will be a variation in the time, sometimes when there is no system call to handled, it will complete in the shortest time because this is the highest priority task.

Now, if it was in the kernel mode, due to the other task calling see, other task in a loop is just keeping on calling the system call. So, if you notice a variation for the high priority task to complete the system call to complete, then you know that the variation is due to a low priority task having made the system call, and that is the difference between the lowest time it completes, and the highest time it completes gives the blocking time, just think of it.

Low priority task is (( )).

Yes.

(( )).

Because the system call makes it go into the kernel mode right. So, if it is a…

But (( )).

If it is a blocking kernel.

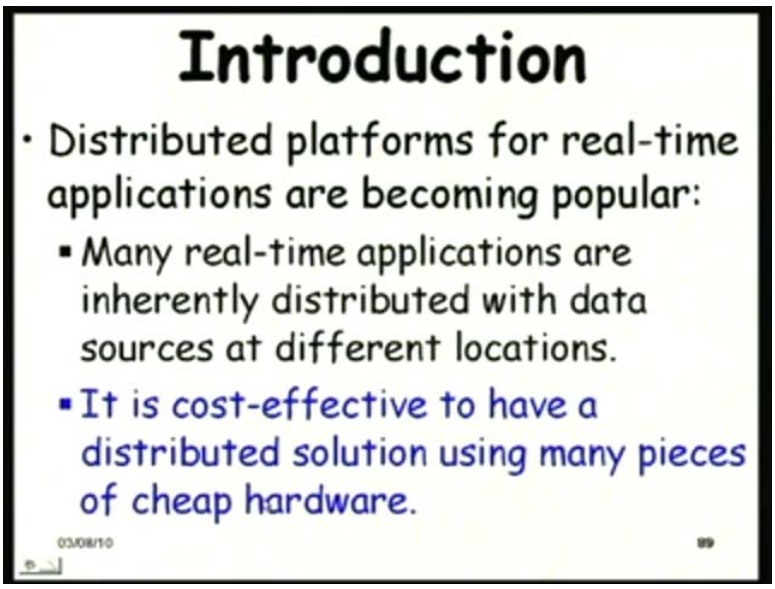
But only highest priority task is (( )) currently so.

No, no the highest priority task is before the highest priority makes the call, this one is keeping on making the call, and the highest priority task, once in a, while it makes a call and then you measure, and then after sometime, you make another call. The highest

priority task is once in a while making a call, and then reporting the time, low priority task is running in the background just making system calls in a loop.

So, just think of the experiment, you can refine the experiment think of other experiments also, how you can measure the kernel blocking.

(Refer Slide Time: 38:35)



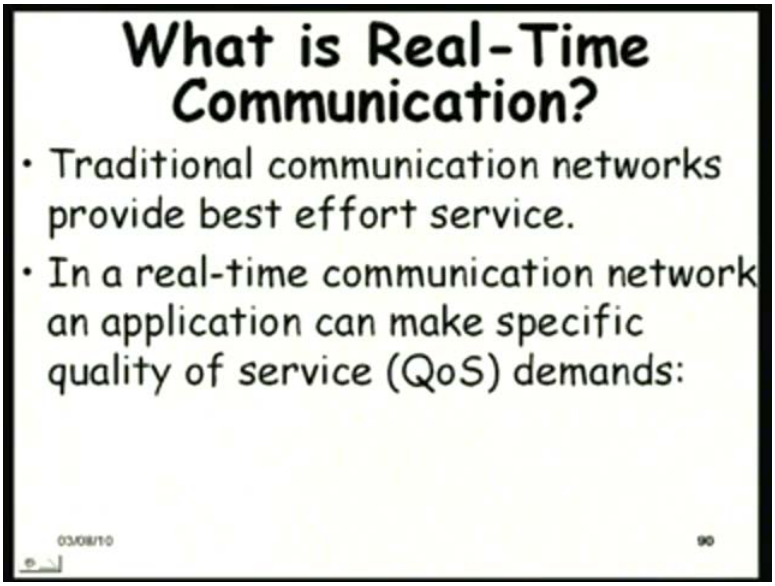
So, we will start a new topic, which is real-time communications. So, you have seen so far some ideas about real-time operating systems seen about the scheduling policy resource having commercial operating system, real-time posix, the features that need to be supported by a commercial operating system, and few commercial operating systems; now, let us have some idea about real-time communications.

Nowadays we have applications that are becoming more complex, sometimes in distributed platforms, and sometimes they just need to communicate with another application somewhere, and many applications are inherently distributed, because the data is getting generated somewhere, at different places the data is getting generated.

And then, the actuator is at some other place or a different places, and the data needs to be process locally, because otherwise, we will have to transmit too many, too much of data, the data needs to be process locally send to a controller, and the controller send some signal to the actuator, which is also a processor, and then it does some computation and drives the actuator.

So, in those situations, controller and many other situations, we can have a naturally distributed system, and then, it also becomes cost effective to have many pieces of cheap hardware rather than having one centralized system. So, these also have a factor which is driving this distributed solutions as far as the business sense is concerned, cost is concerned, it is much more cheaper to have several small pieces of hardware rather than one large piece of hardware.

(Refer Slide Time: 38:35)



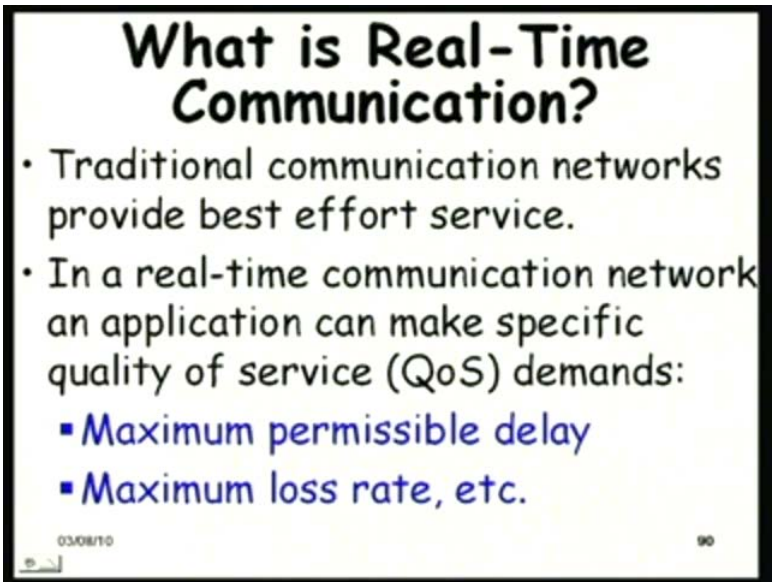
Now, let us try to find out what this real-time communication is about what are its characteristics how do we define and so on, and how is it different from a traditional communication, because we know from a first level, of course, some basic ideas about computer communication.

A traditional communication network provides best effort service; the best effort service is that, it gives no guarantee on the time, when it is the, the communication will occur for example, let us look at Ethernet.

So, it just tries to access the media, and there is no guarantee, when it will succeed in accessing the media, if the load is high, then it will take time for it to get the media, and transmit, because of the collision, basic collision, and link principle takes variable amount of time depending on the load, but we are talking of a communication mechanism, where the maximum delay, and the variation in delay are guaranteed. So,

that is the simplest definition, we will see more elaborate definition of what is real-time communication.

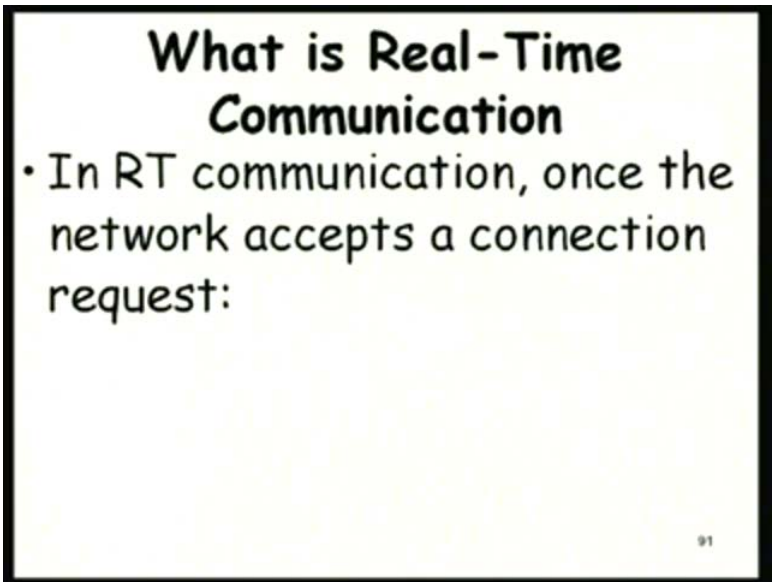
(Refer Slide Time: 42:24)



So, in a real-time communication network, an application can makes specific quality of service demands, see in a traditional network, whatever is the best the network supports, it just gives that at that moment depending on the load and so on.

It provides the best service, that is possible under those situations, but here in a real-time communication, the application the real-time application, we will make specific quality of service demand, for example, what is the maximum delay, it can terror it, what is the maximum loss rate etcetera. So, these are the different quality service, I mean, there are many more here just examples of quality of service; so, that is a very basic definition in a traditional network.

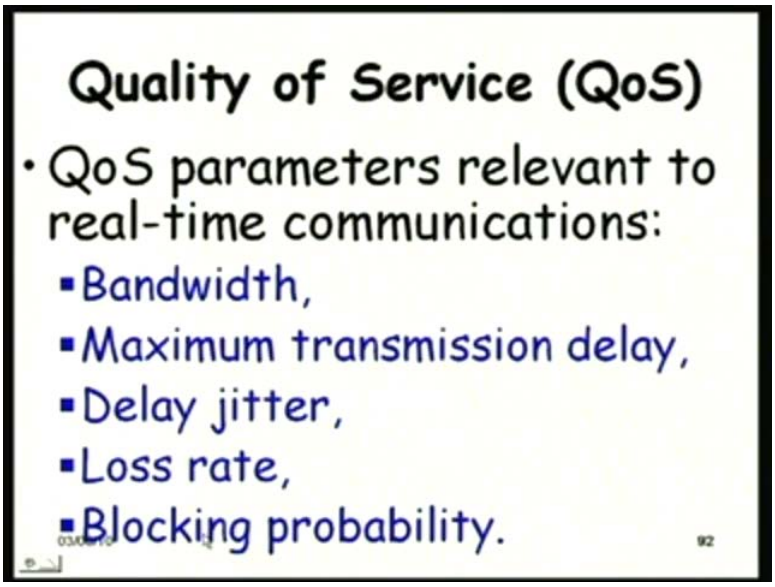
(Refer Slide Time: 43:21)



The application has no control over the kind of service that will be deliver to it, whatever best effort the network can provide, it just delivers that, whereas in a, if we are talking of a real-time communication being supported the application (( )) to request for a specific service or a service with a specific quality, and then the communication system should be able to support that quality of service, then we have a real-time communication.

And we will elaborate this definite on as we proceed, we are saying that the quality of service can be about maximum delay, it can be with respect to the loss, how much loss is tolerated. In a real-time communication, once the network accepts a connection request with a specific quality of service requirement from the application, it has to guarantee the requested service quality, but let us elaborate this term service quality, what do we mean exactly by service quality? We are given some example of a service quality, in terms of maximum delay with respect to the loss rate that can occur.

(Refer Slide Time: 44: 44)



We will consider four parameters for quality of service in our later discussions, in the next and subsequent classes; one is the bandwidth that would be available to the communication, so how much data it can transmit per unit time.

The maximum transmission delay, the jitter that is, whatever delays is the maximum transmission delay, that is would be constant preferably for some applications especially, because sometimes if it transmits in no time very fast, and sometimes it takes the maximum transmission time, then for some applications we will have problem. So, the difference between the minimum delay and the maximum delay should not be, should be within certain bound that the application need, of course, the loss rate.

So, let us look at some ideas about this, and the kind of applications that might make such requests, and also the blocking probability, for example, you are trying to initiate a call, but it could not be accommodate by the system, by the communication system, because it does not have resources to support, see you have said that certain delay bandwidth is required, and then it might, so happen that the system does not have those resources.

So, then, your call will block, and if it blocks too often, then your application will suffer right for applications, the blocking probability may be almost zero for some applications,

some application tolerate no blocking probability, some other applications might tolerate some amount of blocking probability.

Refer Slide Time: 46:58)

Let us contrast first with the definition of a real-time communication with a traditional communication. Let us look at a traditional application like a FTP, email etcetera, we know that, they have reliability requirement is not it. If you do a FTP or email, and on the other side, you receive a corrupted document, nobody will accept, that is not it, FTP, it is a one gigabyte file, and then find that part of it is corrupted, then it is as good as not downloading that file is not it, and similar with respect to email, you sent a large email, and then found that (( )) of many parts, you would not like that network.

So, for soft real-time applications also reliability can be a concern, we had said that reliability is also a concern for hard real-time communication; actually, if you look at the network protocols, much of the complexity of the traditional network protocols arise from need for loss free communication. So, all that traditional communication whether we talk of TCP protocol. The idea is to provide loss free communication, so how do they ensure loss free communication

(( )).

Sorry, because the media is (( )), there can be a no at the physical level, there may be a, let us say lightning occurred or let us say electric spark occurred, and the data got corrupted, but finally, it has to be ensured, that it is a loss free.

So, how do traditional protocols ensure loss free communication?

(( )).

Error correcting, error correcting.

It is not possible (( )).

Is not really usable, see he said error correction mechanism, but normally error correcting code has too much of overhead think in the first level course, you would see that a error correction mechanism is very expensive, in terms of the amount of overhead in terms of data, that you need to transmit for correction.

Correction is extremely expensive and difficult, but what can easily be done is error detection and retransmission. Correction codes are expensive not usable for a typical communication. So, what typically done in a traditional network is error detection and retransmission, that is how it works, at various levels there are subject the physical level, at the packet level and so on, because packets may be lost, even though the lower level ensures that the data is sent, let us say physical level you have checked through correction, finally, you find that the packet is lost, but how can that happen, that are the lowest level, the physical level you have ensured that, and no correction, sorry, no, no corruption, and there has been retransmission the lowest level.

So, how can at a higher level at a packet, a network level, you find that the packet has been lost.

(( )).

No, all those corruptions, they have taken care at the lowest level, whenever there is a corrupt data, that is retransmitted lowest level physical level.

(( )). So, certain extra (( )) are added, so for the (( )).

That is a packet definition, but how can packets get lost, because then idea access control the Mac protocol will ensure that the packets have been, you know the data, that is send the frames, basically the frames have been transmitted successfully that is ensured, but at the packet level, there may be loss in packets how can it happen, very first level course, I mean, question that I am asking at the frame level, you find that frames have been delivered correctly, but at packet level, there is a loss, you would not again have a detection and correction mechanism at the packet level, you know that TCP etcetera, they keep count of the number of packet, you know, they have a serial number is not it. So, why is that necessary, because lower level, you have ensured that there are no errors?

(( )).

A reordering, reordering is means not a problem, say very first level question, the packets might have been dropped intermediate nodes; this is an end-to-end communication. So, there are many intermediate nodes, that might be involved, and somewhere, because of the load considerations packet might have been dropped on intermediate nodes, and then finally, at the destination, even though at the lower level, you have ensure that, there are no corruption at the end-to-end, you see that packets are missed.

So, we will not really diverse that extent, and even if you look at soft real-time communications, FTP etcetera, even the packet delay, and average throughput are also important parameters, because if you takes long amount of time, and the throughput is very low that wont also be acceptable for FTP or email.

(Refer Slide Time: 53:56)

But what are not important for soft real-time communication is the worst case packet, delay the jitter, and the worst case throughput what we are concerned is the average case delay, and throughput even jitter is not a concept for soft real-time communication. So, the worst case delay throughput are not a concern for a soft real-time application, what is of concern to a soft real-time communication is the average packet delay average throughput, and jitter is normally not a concern for soft real-time application.

On the other hand, if we have a television transmission occurring, and as a non-interactive manner, which are typical televisions here, there are strict bounds on the jitter, but insensitive to delay do you agree with this.

So, in a typical television, see nowadays television have become interactive might have seen that, but let us not consider the interactive television, but when the transmission is occurring one sided, if these are, there are bounds on the jitter, but insensitive to delay do you agree with this.

(( )).

That is what I am asking basically, that why should the traditional television the insensitive to delay, but they do not tolerate the jitter.

(( )).

Not.

Sir, like (( )), soft real-time right now (( ))..

(( )).

See, see if all the, let us see you are looking at a movie or something, so it does not matter, when exactly it was transmitted from the antenna, and when you received or let us say, when it was support, suppose to be broadcast, and when you exactly receive it as long as it is no within few seconds or minutes.

(( )).

So.

(( )), if all the frames are having the same (( )).

Exactly.

Suppose, one frame has coming in time (( )).

No, that is jitter, so, we are saying that, it does not tolerate jitter, where some frames are coming very fast, some are not coming first, but if all the frames are equally delayed.

Then (( )).

Then the viewer will not feel anything, that is what we are saying, same thing I just written down, that in a traditional television, we requires stringent bounds on jitter, that is different frame should not incur different amounts of delay, but the maximum delay is not a problem, I mean, the amount of delay as long as it is constant, it is not a problem, it can be one second, it can be five second does not matter, but the jitter, even, if it is few milliseconds, 50 millisecond, you will immediately notice the picture getting disturbed.

So, we will stop here and we will continue in the next class from this point. Thank you.