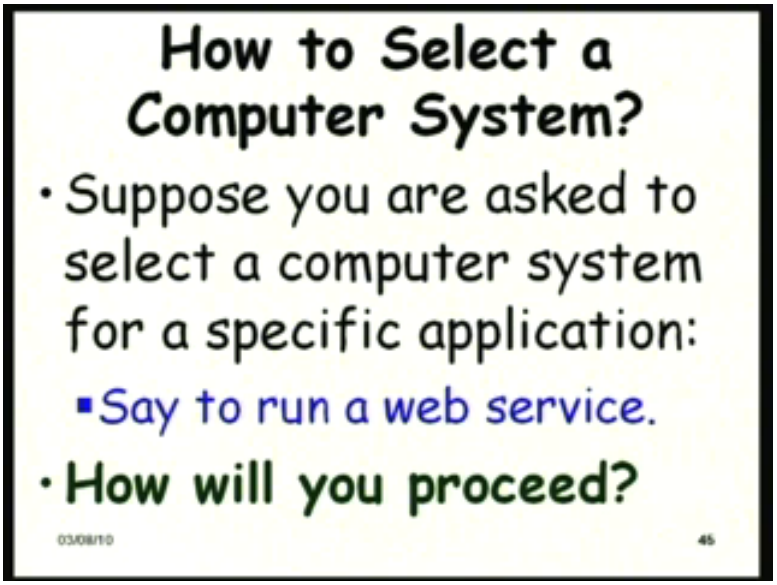
Real-Time Systems
Prof. Dr. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture No. # 29
Benchmarking Real-Time Computer and Operating Systems

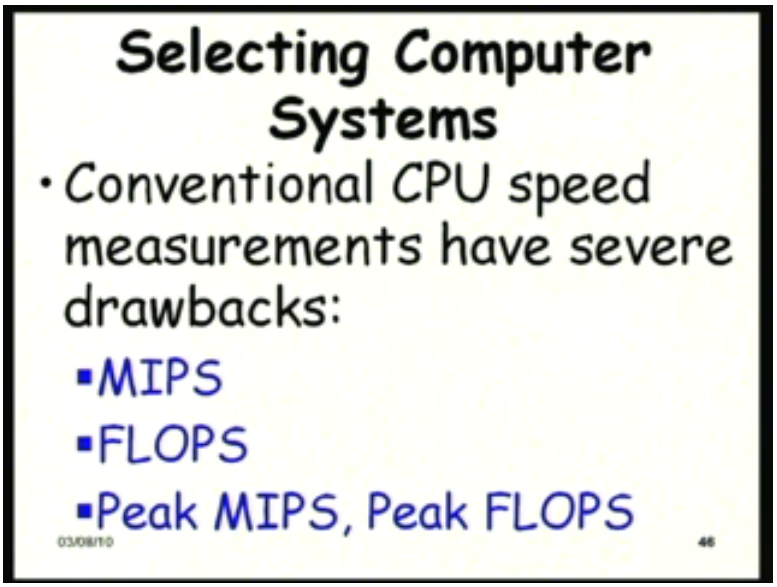
Good morning; so, we will get started. So, far we have discussed about some basic concepts in real-time operating systems, and then we have examined the available operating systems, both open source and commercial. So, today we will examine that for a given application, how we select a computer for real-time application or if the hardware is fixed, how you do select an operating system, out of several available operating systems to meet the requirements of our application. So, let us proceed with that.

(Refer Slide Time: 01:09)



So, we had just started this last time that if you are asked to select a computer system or given application mean mind, the computer will be used for specific purpose. How do you go about selecting the computer, because there are several vendors, several types of systems with different configurations and so on. So, I have posed this question yesterday; and we had given the example of a web service that what do you do to select a computer to host a web server for an organization.

(Refer Slide Time: 01:51)



And then some of you had said that MIPS, and flops rating, etcetera can be used to distinguish between different systems. But then we had said that there is a big problem with the MIPS and flops rating; these are misleading.

Why are this misleading?

(( )) correctly represents the… (( ))

Not clear

Correctly represents CPU users means what?

(( ))

Anyone else?

(( ))

So.

(( )) take different thing… (( ))

So, what he was saying is that, the different instructions can take different times to execute, but then what is your problem.

(( ))

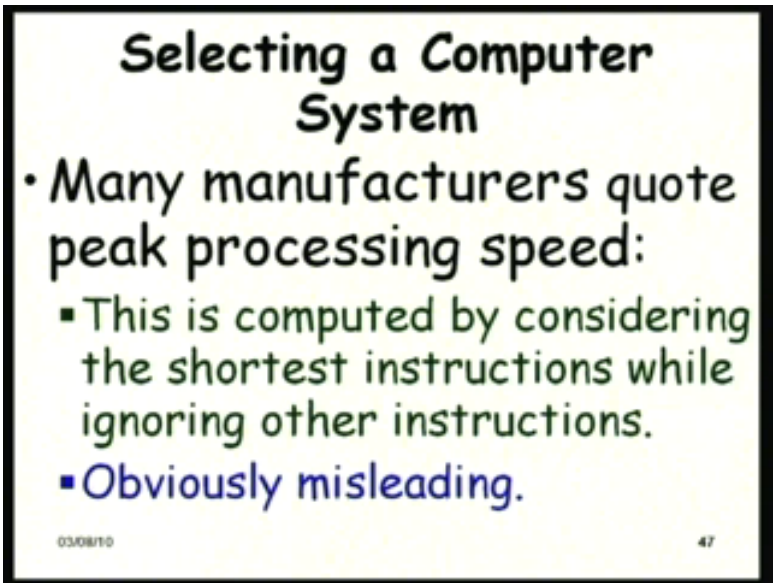
What do you mean by standard?

(( )).

So, the MIPS rating are usually quoted by vendors with respect to the shortest instructions. Because after all, they will take some programs and execute or maybe they will just take the average of all instructions depends, and how to compute? Because MIPS; how to evaluate MIPS? That is no standard definition. So, somebody can even take the shortest instructions, and see how many instructions can be executed per second But actually, when you bring the computer, and try to run your application, your application will have a different mixed up instructions; may be it does lot of IO; each time your web page is loaded needs to do IO, and also networking right. So, lot of I O is involved, and very less ALU operations.

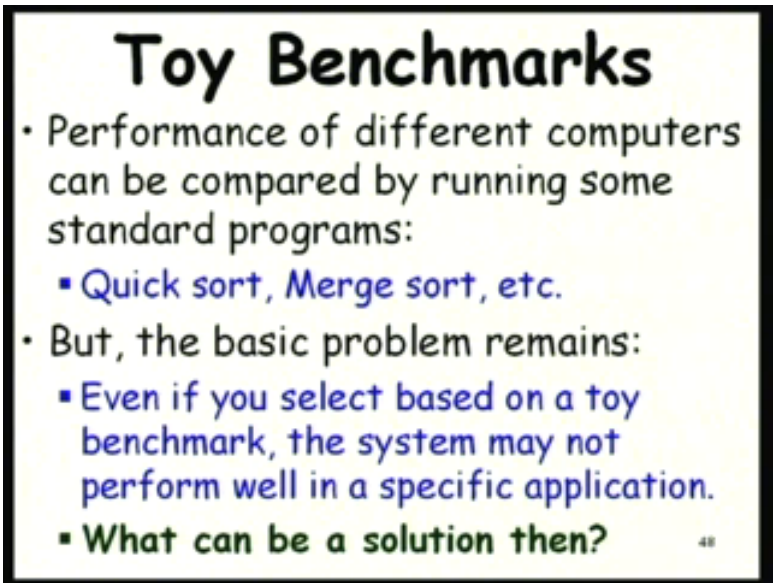
But if the rating is computed, the shortest instructions are ALU operations. So, in that case, it will be highly misleading. Do you agree with that? So, later the peak MIPS and flops were quoted by vendors; implicitly with the understanding that, they are expressing the performance of the computer with respect to the shortest instructions, and the instructions that execute in the shortest time. But still that does not help. I mean, it says that, it will has some peak performance with respect to the shortest instructions. But as far as your program is is concerned, we are not going to use those instructions. right.

(Refer Slide Time: 04:51)



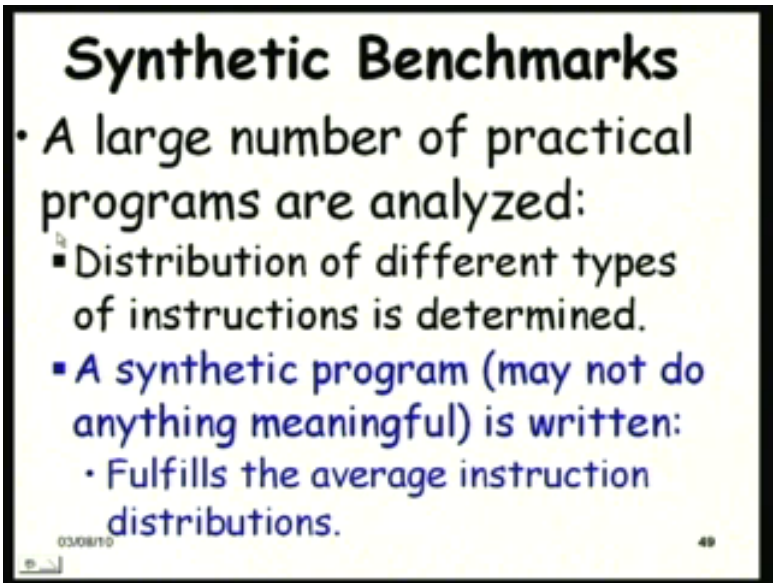
So, even if you compare the peak, MIPS or peak flops, it hardly helps as far as your selection for a web server application is concerned. is that ok. Let us proceed. So, this exactly you just saying this; that the peak processing speed computed by the shortest instructions, and other instructions are ignored ION; those instructions and it becomes misleading. Later I mean, once this was known that MIPS is a problem, everybody agreed that MIPS is a problem. So, the first thing that one would think of is that, why not run one program? Some program? And then say, whether it performs well with respect to program; that program. And that was the concept of a toy benchmark.

(Refer Slide Time: 05:14)



In a toy benchmark, some of the standard programs are run. For example, quick sort, merge sort operating on some large amount of data; they can quote that, how much time it takes to quote 100, 1000 integers using quick sort in that on that computer? right. So, that gives some indication of the performance, but again the basic problem still remains; it just tells, how will or how first, it computes merge sort or quick sort; it does not tell, how well it will run as a web server, because the web server the web service will require a different mix up instructions, as compared to a quick sort or a merge sort.

(Refer Slide Time: 06:36)



So, the solution is possibility to use the synthetic benchmark. See the best thing would be to actually get the computer. I mean all the computers that you were trying to select; and then run web service; and then measure the performance. But that is practically impossible. We cannot do that; we cannot get all the vendors to get all their computers here or operating system. And then you evaluate performance; it is impractical, because they would not give you the computer to experiment; it'll be too expensive; and also to install and run, it will be not a practical solution.

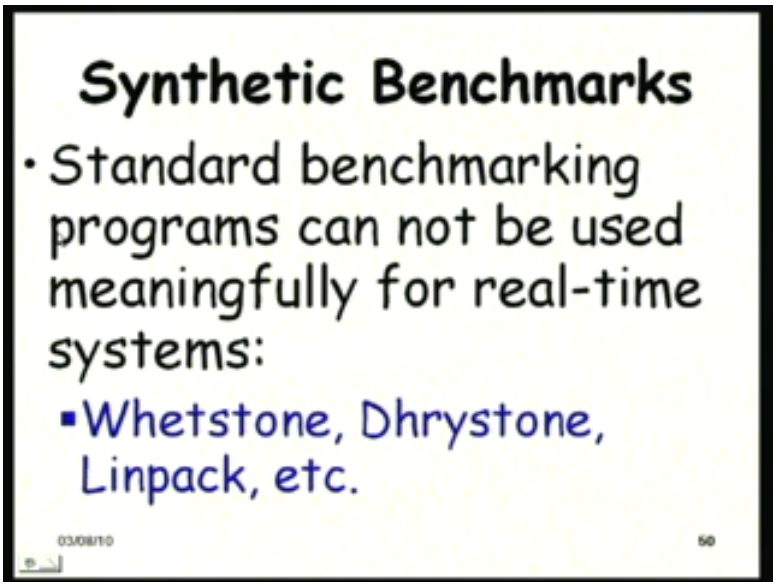
So, the solution that was considered by the synthetic benchmark. So, we were just discussing this yesterday, that in a synthetic benchmark, the instruction distribution over a large number of programs is analyzed. So, what is the difference of frequency of different instructions that occur in variety of programs? And then you take the average, and then find out in a typical program; that is the average program. How many ALU instructions, How many IO instructions, etcetera - different categories of instructions, you can find out for a typical program; that is a average program right.

And then you can synthesize a program or write a program, which has that many percentage of ALU; let us say, it has 12 percent ALU operation; so, what you can do is to give first 10 percent or let us say, 100 ALU instructions; right that is twelve percent. I mean as the distribution is, then write the IO statements, and remember that this program does nothing It is not doing anything meaningful, But the only thing is that, it has the

required distribution of the instruction set, as an average program. So, it'll give you an idea of how an average program will run on this computer.

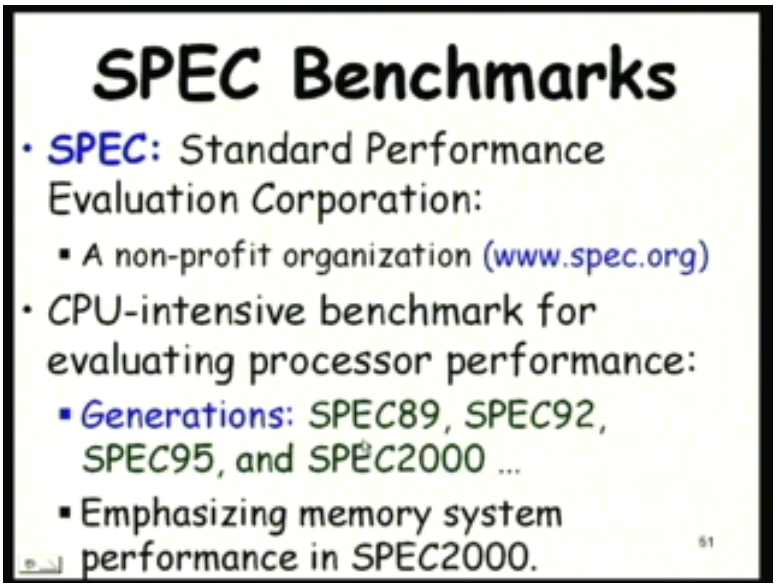
So, helps to some extent, but not really, isn't it? Because web server is not a average program; it is a specific program whose instructions distribution will be different from an average program, but still, it'll give a good indication. Now, let us see what are the other solutions? So, let us look at the synthetic benchmark; the synthetic benchmark is written with which fulfills the average instructions distribution. And there are many synthetic benchmarks which were popular: The Whetstone, Dhrystone, LINPAC, etcetera; these were popular those days about 25 years back, but this had a problem, that we were discussing yesterday.

(Refer Slide Time: 09:08)



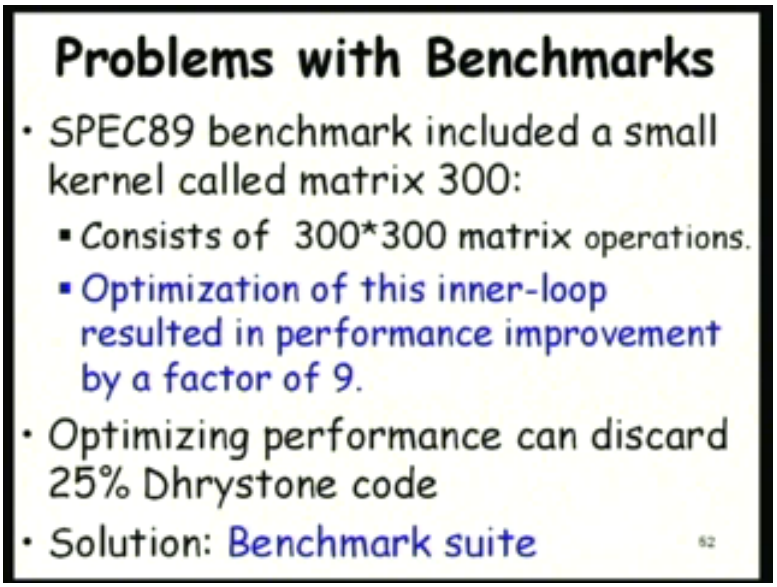
The problem was that the manufacturers, they optimized their system with respect to this benchmark. Because they know, finally they will have to quote with respect to a Whetstone or a Dhrystone benchmark. And these are the source code of this is available Whetstone, Dhrystone, etcetera. And you can easily analyze, what kind of instructions they use; what exactly they do and so on; and you can write a compiler which will either generate instructions, that takes the shortest time or it can make some optimizations; simplifications of a expression evaluation, and that kind of thing. So, that runs well for that program, optimized to run that program well.

(Refer Slide Time: 10:32)



So, that is your problem, isn't it? Because again, it depends on how well the compiler is optimized. Based on that, the performance will be reported. So, then what can be done? The SPEC benchmarks have become very popular over the last 15 or 18 years. SPEC stands for Standard Performance Evaluation Corporation; it is a non-profit organization; you will get information from www.spec.org; you can download the benchmark programs the source code also. And there are various types of benchmark programs; we will just look at them very briefly; for example, I think SPEC 89 was the first one and then SPEC 92, 95, 2000, etcetera.

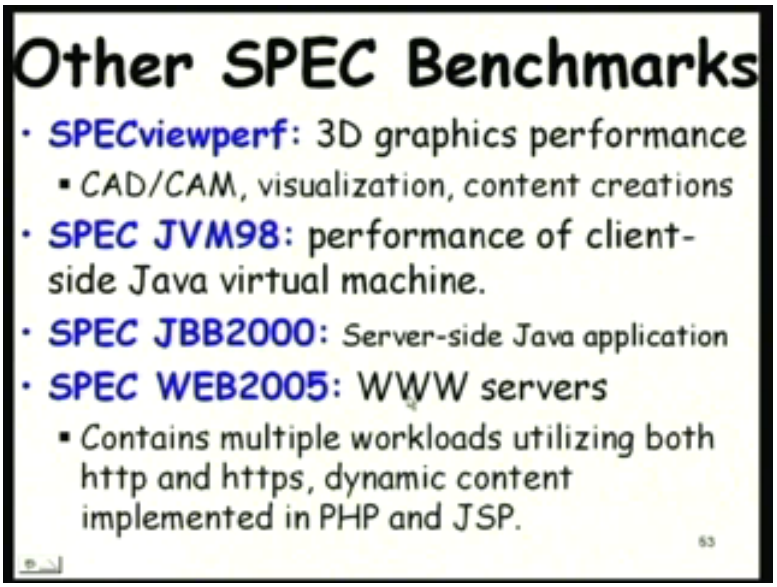
(Refer Slide Time: 11:18)



So, the SPEC89 had a problem; the same problem that Dhrystone, etcetera had. It included a small kernel called as matrix 300, a small part of the code which is executed as the inner loop; right it is executed maximum number of time which consists of a 300 by 300 matrix operation. So, that is carried out many times. And by optimizing the inner loop, somebody can improve the performance by a factor of 9; just see by writing the compiler carefully, and just targeting that inner loop, somebody can improve the performance by a factor of 9.

And also for the Dhrystonek, somebody can discard 25 percent of, if you look at the Dhrystone benchmark code, you can see that many of the code can be just the compiler can discard it without executing; for example, as giving an example of, if one do something, right that always evaluates to true. So, that means that, instruction will anyway be executed. So, why do we need a few instructions? And then the benchmark suites were used. Here, it is not just one program, based on which the benchmarking is carried out; It is a suite of different programs. So, you cannot just optimize one part of the code.

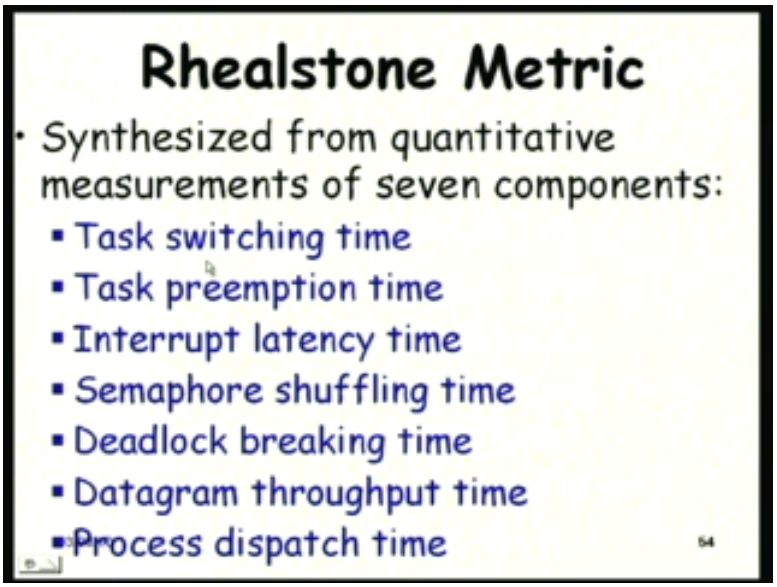
(Refer Slide Time: 12:54)



And because you want to choose computers for different applications, there are, if you look at the SPEC website, you'll see various benchmark programs. For example, SPEC view perf for graphics performance 3d graphics, visualization, content creations and so on. JVM 98 for client side virtual machine; JBB 2000: server-side java application; web 2005: web servers. See, possibly this is the one, you should use. This benchmarking would ask your computer vendors to quote the performance with respect to web 2005 benchmark. And then it will give you a fairly good idea about, how the computer system is going to perform, when used as a web server?

If you look at the program here, the source code; if you download from SPEC site and examine, you'll see that, there are multiple workloads, utilizing the HTTP, HTTTPS and also dynamic content PHPJSP, etcetera Again, there should not be a precise measure of the performance, because your organization might finally be using a static web pages; and then this will be, you know slightly of the benchmark. I mean the actual performance that a computer will show.

(Refer Slide Time: 14:35)



But still, it will be fairly close. Now, let us see, how do you benchmark real-time computers? You want to select computers for some embedded real-time application, what do you do? You have to first of all, consider identify some parameters, that are important for real-time applications. What are the parameters for that are important for a real-time application? Can anybody say some few parameters?

(( ))

No, scheduling algorithm is your choice.

(( ))

Timer.

(( )).

So, you mean, you the finer the timer, the better it is.

Not really.

See your application might need the 10 millisecond timer or a 100 millisecond timer. What do you do with a nanosecond timer?

(( ))

Not really, let us see, what are the parameters that we can consider? The task switching time, task preemption time, interrupt latency time, semaphore shuffling time, deadlock breaking time, and datagram throughput time. So, these are some of the important parameters; we will see that. And later, see this we are talking of a rhealstone metric; we will look at different metrics; and then we will see that, there are other parameters also, not included in the real-time, rhealstone metric which are important.

Yes

(( )) significance of that metric; it is all whetstone dhrystone (( ))

See, I mean he is asking an interesting question. He says that, see whetstone Dhrystone etcetera, and this is rheal stone. So, is there a reason? See, the only reason, one can think of is that, you go to a goldsmith and ask him that, how what is the quality of this gold? He will take out the stone; he will rub it; right and he will tell you, what is the quality of the gold, and maybe I think so, that these, I do not know. I mean, it is not written anywhere, but possibly that is the reason; that this is like a stone with a goldsmith. You take the stone, and try to benchmark different computers possibly; that is the reason.

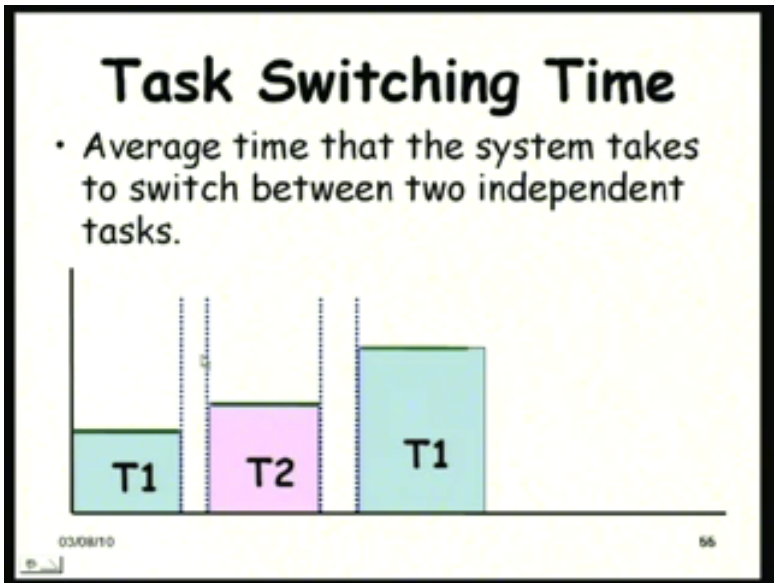
(( ))

I do not know. I do not know why it is extra? Sorry any other question.

Sir (( )) there will be seven components, but we have listed six.

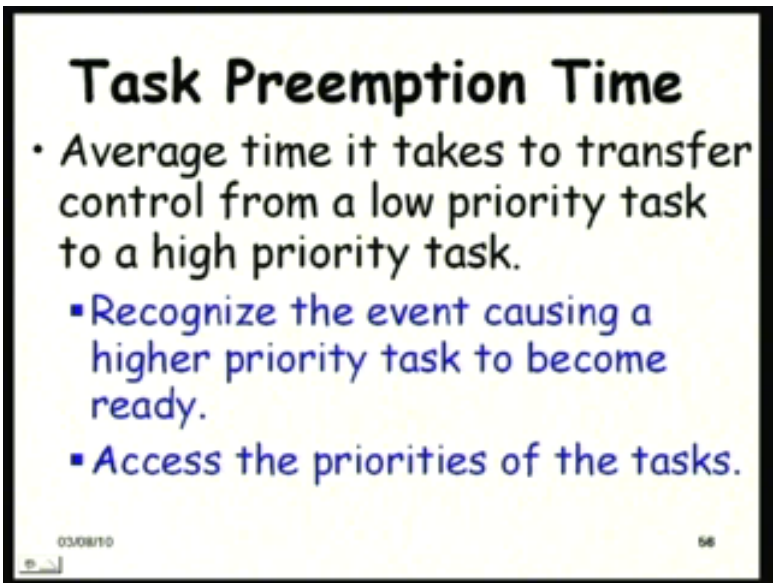
So, maybe I just missed here. We will just see the details of this. yeah right. So, he says there are seven, but him there is six. He has counted it here. So, I think one, I have missed here; I think the process preemption time. No, the task preemption time is, there; we will we will see. I think there is one more there. As we proceed, I will just tell you. But this is not really the jostled set of metrics, that we can use to measure real-time performance; we will see that there are many more important parameters, that are not really mentioned; we will we will discuss about that in different metrics. I mean, this one has proposed this set. So, here it is there; I did not show it, the process dispatch time.

(Refer Slide Time: 18:20)



So, let us see what this mean? The task switching time is the time, the system takes to switch between two independent tasks. So, let us say T1, T2, T3, sorry T1, T2. Now, let us say for some reason, T1 blocks need to start T2. So, how much time does the system take here to do this switching from T1 to T2; context switching, basically.

(Refer Slide Time: 18:56)



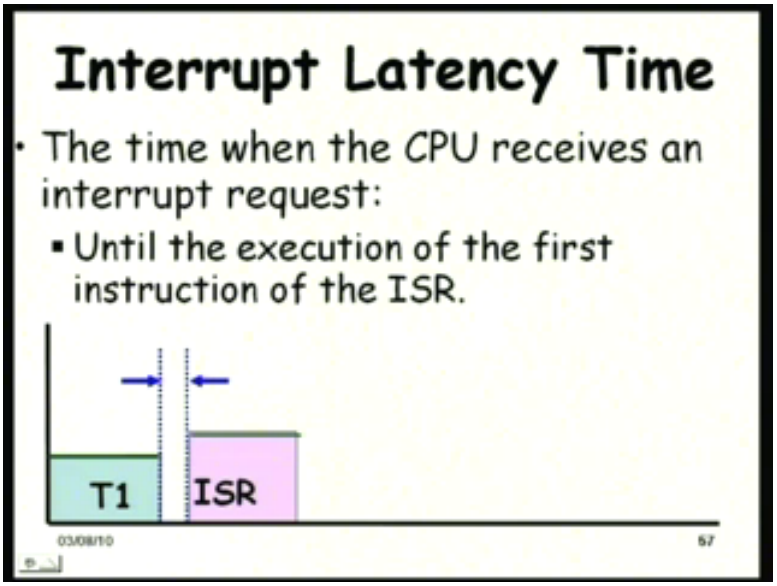
First, we will see the times; I mean the parameters. Later, we will consider how to really evaluate them? I mean can you write a application or a program which will evaluate them, because if you can do this, write your own or we will see that, the programs that

we are saying, these are just examples; the way you will compute this; these are examples; not that, that is the only way, we can do that. May be you can think of a better way of measuring benchmarking real-time computers and you can publish that, because not much result. If you look at the literature, not much results are available on benchmarking real-time programs. But still, it is a very practically important problem.

So, let us see, what are the parameters? And what are the ways we are suggesting? I mean, the literature suggests that these should be evaluated. And then you can see that, there are many other ways, you can do it. So, it can be a good exercise, if you try that; let us proceed. So, the task preemption time; So, here if a high priority task has become enabled or has rest arisen, it should get the CPU, is it not? Because it is a preemptive priority based system that we are discussing.

So, the task that is involved here, the work that is involved is a recognized, the event causing a high priority task to become ready. So, the event occurs with a sensor or something, and then it takes time for the event to be registered by the processor. And then the operating system has to access the priorities of the tasks. See that, this is a higher priority task, and then it takes time for context switching. So, context switching time or the task switching time will be less than the task preemption time, is not it? Because there are other activities involved here.

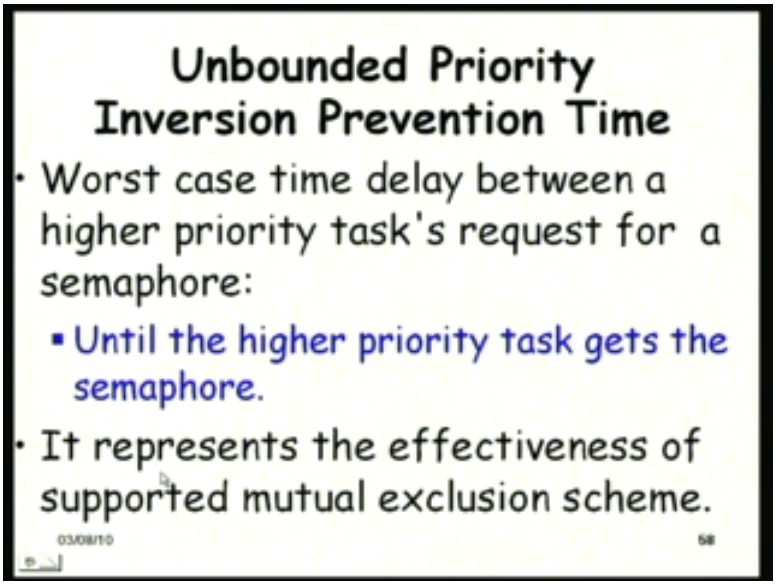
(Refer Slide Time: 21:19)



The interrupt latency time; this is the time, when the CPU receives an interrupt. So, from the time, it receives an interrupt to the time it executes the first instructions of the ISR is the interrupt latency time. So, let us say, it was T1 was executing and at some point, the interrupt occurred. And it takes certain time before the interrupt service routine starts executing; the first instruction starts executing. So, the time it takes between the events causing the interrupt event to the first instruction of ISR getting executed is the latency time.

And of course, this is the important criterion as far as real-time processing is concerned, because if a task needs to complete after the occurrence of an event within certain time. Then if too much time is wasted here, in the as interrupt latency time, then we have very little, that we can do to run the task and meet it is deadline. This has to be low; the interrupt latency time has to be low; that is the interrupt latency time.

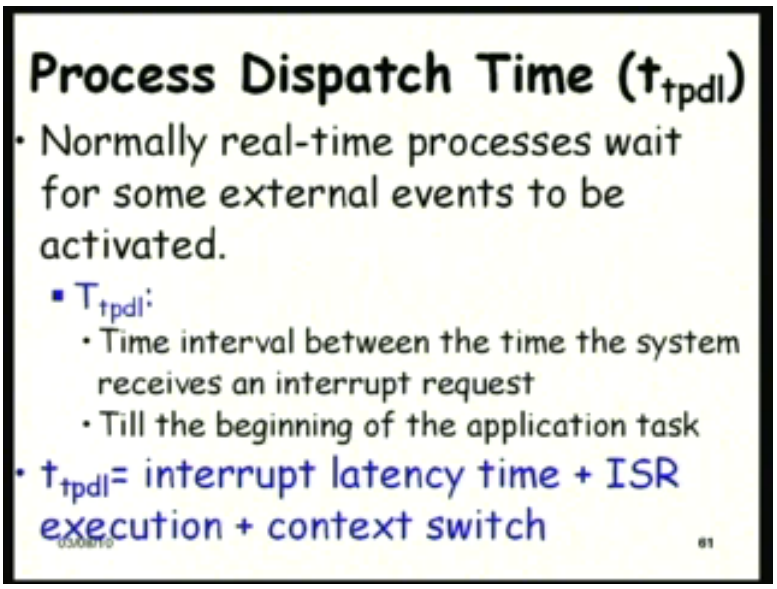
(Refer Slide Time: 22:58)



Unbounded priority inversion prevention time; whenever high priority task requests a semaphore shared resource, then what is the worst case time delay, that can occur? So, the worst case time delay will occur, when a low priority task is holding the resource. And intermediate priority tasks are not allowing the low priority task to complete; unbounded priority inversion will occur, and it will take long time. So, this parameter represents the effectiveness of the supported mutual exclusion scheme or what is the scheme used to avoid unbounded priority inversion? Is it just a inheritance mechanism?

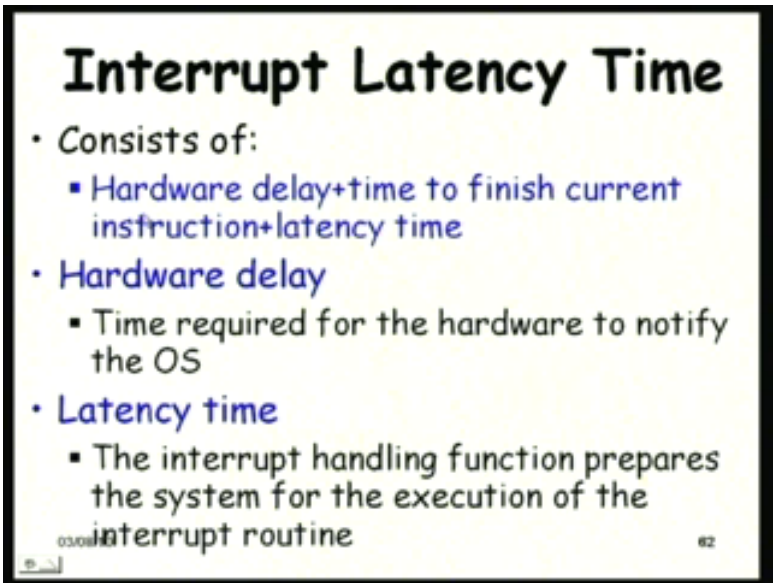
Is it a ceiling protocol, locker protocol, etcetera or is there is just an ordinary operating system, does not do anything specific, as far as request by a higher priority task is concerned.

(Refer Slide Time: 24:28)



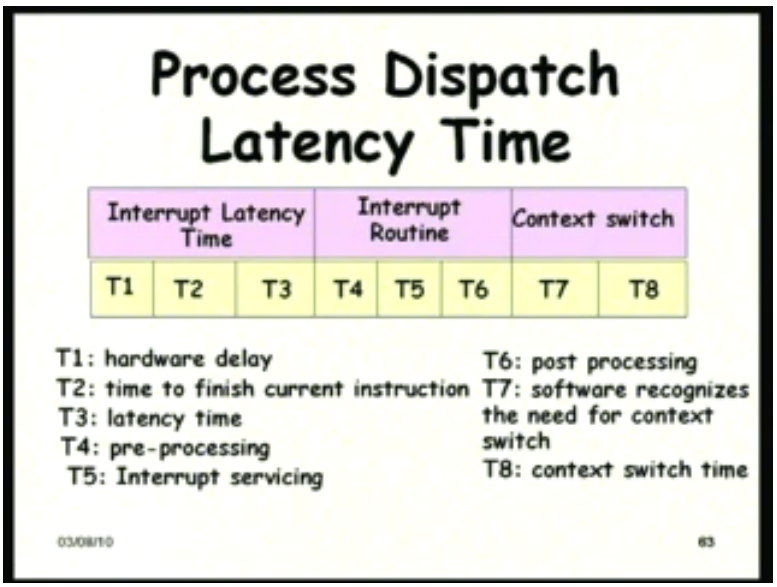
So, some problem with the diagram; we will just look at those two parameters little later. The dispatch time; this is the time, the real-time process wait's for some external event to be to be activated So, the time interval between the system receiving an interrupt request and beginning of the application task. So, just see, this is different from the latency time, isn't it? It includes latency time, in fact. See after an event occurs which is suppose to start a task, for example, let us say temperature became high, and task handling that event needs to execute; then first, the interrupt latency time will be incurred; and then the ISR will start executing; and then the context switch will occur; and then the task will start running

(Refer Slide Time: 25:38)



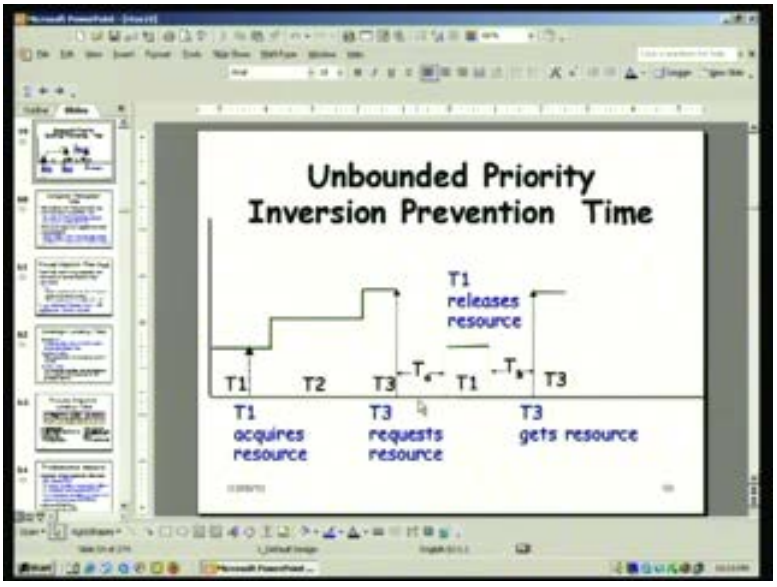
If we look into details of the latency time, we will see that it is the hardware delay; the hardware delay, basically the sensor recognizing the event. And the event reaching the computer takes time for it to be reported, and the time to finish the current instruction and then latency time. So, the hardware delay is the time required for the hardware or the sensor to notify the processor; the latency time is the interrupt handling function, prepares the system for execution of the interrupt routine; basically save the context and all those.

(Refer Slide Time: 26:30)



So, if we see here, the different components of the process dispatch latency time; we will see that, there are three components; one is the interrupt latency time; and then interrupt routine execution; and then the context switch. So, the interrupt latency time is the initial hardware delay for the sensor to recognize the event, and then to be reported time; it takes till it is registered with the C P U; the time it takes to finish the current instruction T2; T3 is the latency time; and T4 is certain pre-processing, that is done in the interrupt routine sorry the ISR, and then the ISR executing; then some post processing that needs to be done; and T7 is that, once the ISR completes the post to recognize the need for context switch; and then the final context switch time. So, let us just see with some two parameters, that we left out.

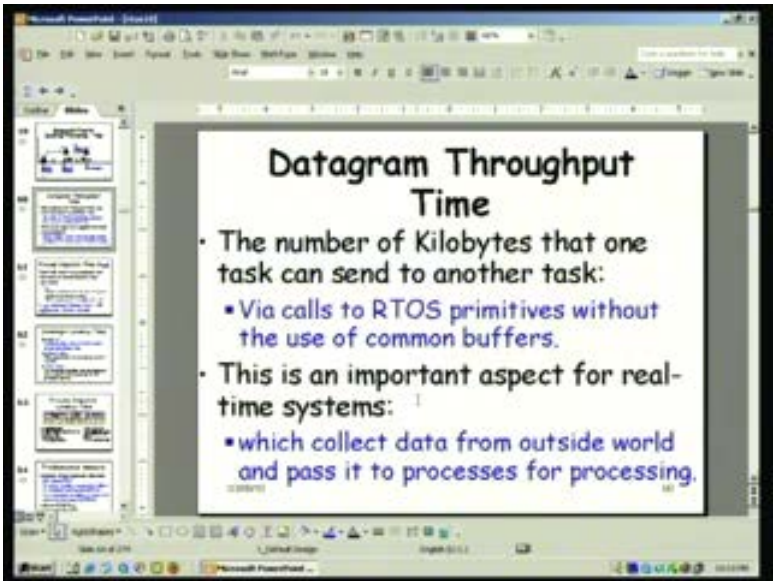
(Refer Slide Time: 27:58)



So, let us see in a reduced size. So here, see the task T1 is a low priority task on the y axis. We have the priority of the task. And then the x axis, we have the time line right. So, the T1 task was executing; and at certain time, T1 acquired a resource; and then a high priority task T2 started executing; and then another task T 3 started executing and so on; and the high priority task T3 requests the resource, that T1 was holding the same resource. So, naturally there will be a context switch, but T1 would not immediately start executing. Because T2 was preempted by T3, and in between there may be several other tasks that might come in.

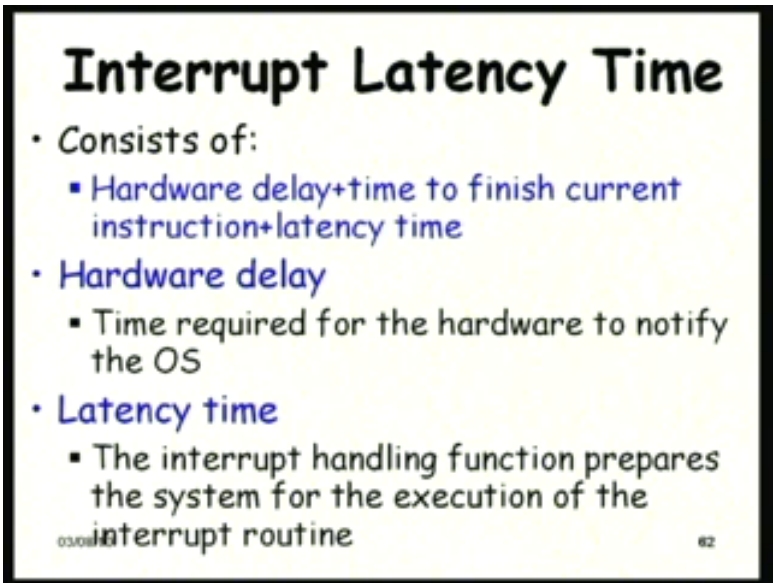
So, there is not a proper prevention mechanism or priority inversion; it will take long time. And finally, T1 will get a turn to its turn to execute; and then complete; and then it takes certain time for T3, basically the context switch time for T3 to start executing. So, the priority inversion prevention time is basically the sum of T a plus T b, right and if there is not a proper mechanism in place, T1 will take long time.

(Refer Slide Time: 29:41)

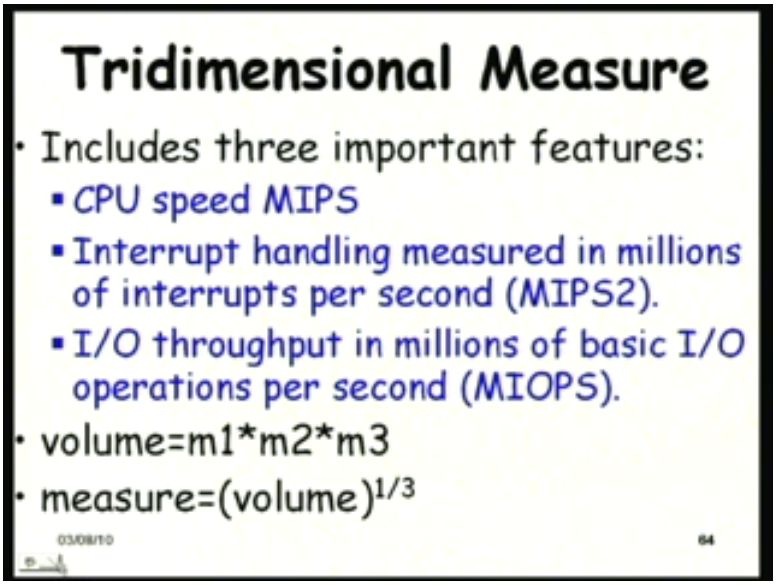


Another parameter, we did not see is the datagram throughout time; as we are saying that, many applications need to transfer data between one process to another. So, this is the time that takes for sending certain kilo bytes of data from one task to another by using the system call, rather than writing data to some shared memory, and then another task reading it. So, let us proceed with our discussion. And there are many synthetic benchmarks which were popular: The Whetstone, Dhrystone, LINPAC, etcetera; these were popular those days about 25 years back, but this had a problem, that we were discussing yesterday.

(Refer Slide Time: 30:35)



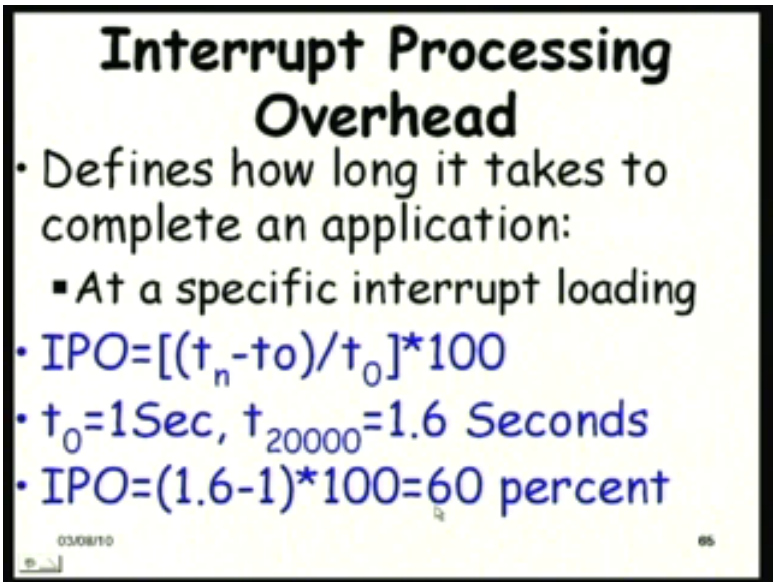
(Refer Slide Time: 30:59)



There is also another one which is called as the tridimensional measure; it says that, see there are three things we need to be need to consider; one is the MIPS speed, and we know that MIPS speed has problem; the second is millions of interrupts per second which is MIPS; two MIPS, one MIPS, two. And then the IO throughout in millions of basic I O operations per second, and then the volume is computed as m1 m2 into m3. and this measure. The tri-dimensional measure is volume to the power 1 by 3. So, basically it takes not only the MIPS rating into picture, I mean.

If you have only MIPS rating available, then you can at least consider the interrupt handling per second; number of interrupts handle per second and the IO throughput another factor that is considered is the interrupt processing overhead, and this can be easily computed through an experiment. So, it basically defines how long it takes to complete an application at a specific interrupt loading. So, let us say interrupts occur, for example, network transfer is occurring at certain place or let us say, disc IO is occurring or page fault by other applications is occurring. So, that time one task takes how much time to execute or a sensor is keeping on giving interrupts. So, how much time does a certain task take to execute?

(Refer Slide Time: 32:02)

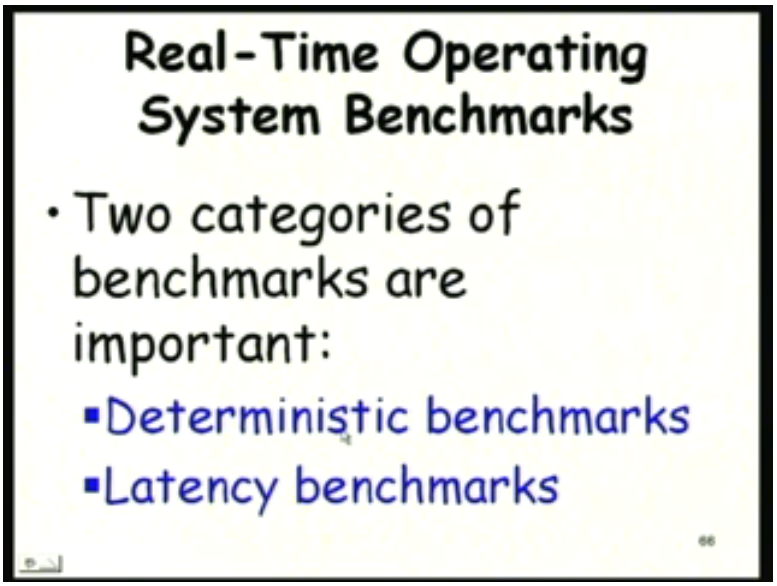


So, the interrupt processing overhead the interrupt processing; overhead is t n minus t 0 divided by t 0 where t n is the time the application takes, when there are n interrupts per second; and t 0 is the time it takes, when there are no interrupts divided by t 0 into 100. For example, if the application takes when there are no interrupts takes one second, and when there are 20000 interrupts occurring per second. See, here we are talking of the application, which is not needing this interrupts; these interrupts are occurring, and the system is just recognizing them and they are not processed.

And for this task, the high priority task which is not affected by the interrupts takes 1.6 second; then what will be the interrupt processing overhead? Will be 1.6 minus 1 divided by 1 into 100, which is basically 60 percent. So, for this specific computer which runs

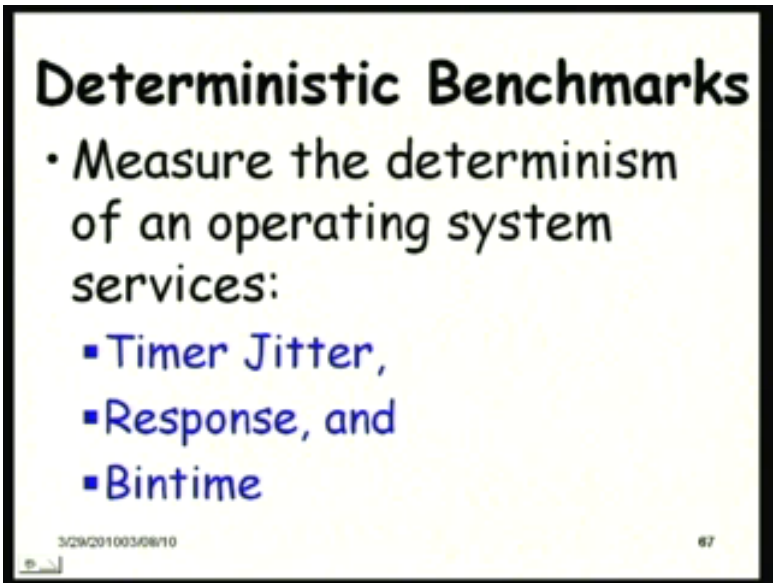
the task in one second, when there are no interrupts, and it runs as a task same task at 1.6 second; when there are 20000 interrupts occurring per second, then the interrupt processing overhead will be 1.6 minus one into 100 is 60 percent. And if we find that, there is another computer which for which the interrupt processing overhead is 80 percent for similar situation; obviously, this one would be better for a real-time application.

(Refer Slide Time: 34:41)

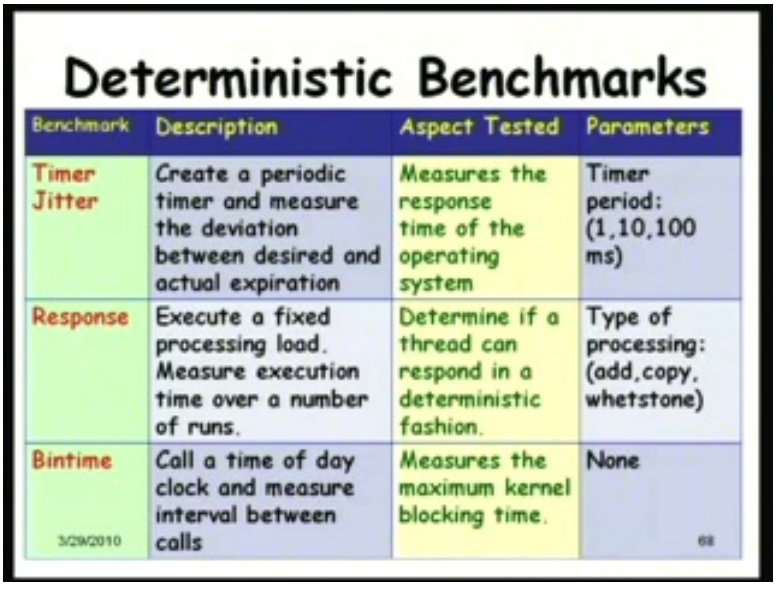


Now, let us see some operating system benchmarks; we will consider two main benchmarks; one is called as a deterministic benchmark; and the other is a latency benchmark; we will we will just see what are those. The deterministic benchmarks, they measure the determinism of the operating system services; basically, whether there is a jitter in the timer. So, when a timer is set to expire, does it give precisely? The time at which it is give to be given or is there a variation in the timer time that is reported to set. A one millisecond timer does it report at 1.01 millisecond or does it report at 0.99 millisecond or does it give at exactly one millisecond. We'll look at another parameter; the response and also the bin time.

(Refer Slide Time: 35:02)



(Refer Slide Time: 35:48)



So, let us see these parameters. So, here we have described; what exactly this involved in a timer jitter parameter? And how we can measure? In the timer jitter, we create a periodic timer; set a periodic timer basically, and then measure the deviation between the desired and actual expiration reset it for 1 millisecond, 10 millisecond and 100 millisecond. And then check; keep on checking, what is the maximum time by which it differs from the expected time?

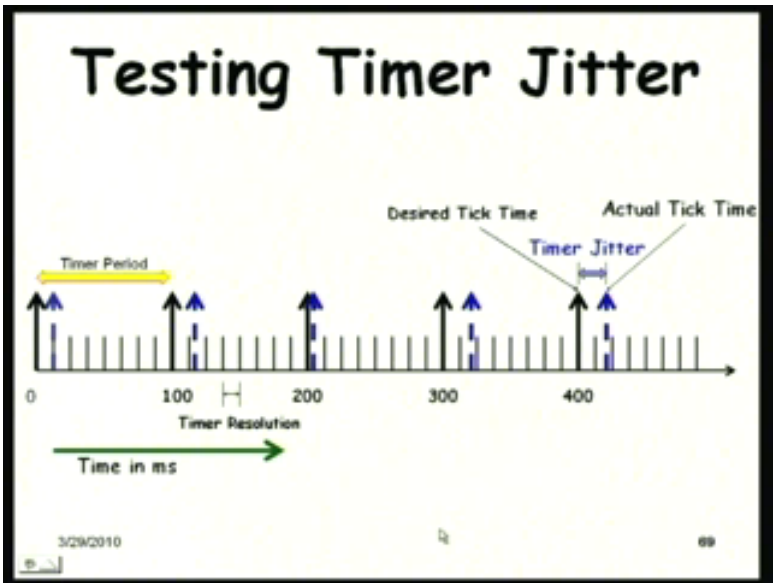
So, look for some 100 or 200 or may be 1000 readings of this timer, and then check, what is the maximum discrepancy that occurred? But then you would ask me that, how do you find the discrepancy? Because the timer you have set for 1 millisecond or 10 millisecond, how do you find the discrepancy? So that, we will just see shortly, how do we find the discrepancy? So, here it measures the response time of the operating system; basically, the timer will report the event; right it is not that the timer did not work; the timer has reported the event, but it is a operating system which has taken time to respond or take is taken different amount of time; each time the timer has given a interrupt.

We'll also consider the response parameter as a deterministic benchmark; execute a fixed processing load, and then see how much time does it takes; each time does it take the same time or is there a variation of the time, that it takes the typical processing load that we might consider; may be Whetstone may be some benchmark; SPEC benchmark. It may be simple hard programs or copy simple ALU programs. The idea is that, when you run a thread, will it complete in a deterministic fashion? And remember that, we are considering a real-time priority; the highest priority, it should not happen that, the kernel is doing something, internally handling the timer response or something.

And then depending on how many reports or may be interrupts occurring for other tasks, it just taking time to handle those. And in the meantime, your task is getting delayed; sometime is completing early sometime; and then we will look at another parameter - the bin time. Here we will call the time of the day clock to report, what is the current time, and then measure the interval between calls. So, basically how much time it takes to complete the bin time. Here we are just notice here that, we are just keep on invoking as system call. See the difference between this and this; the response and the bin time is in the response.

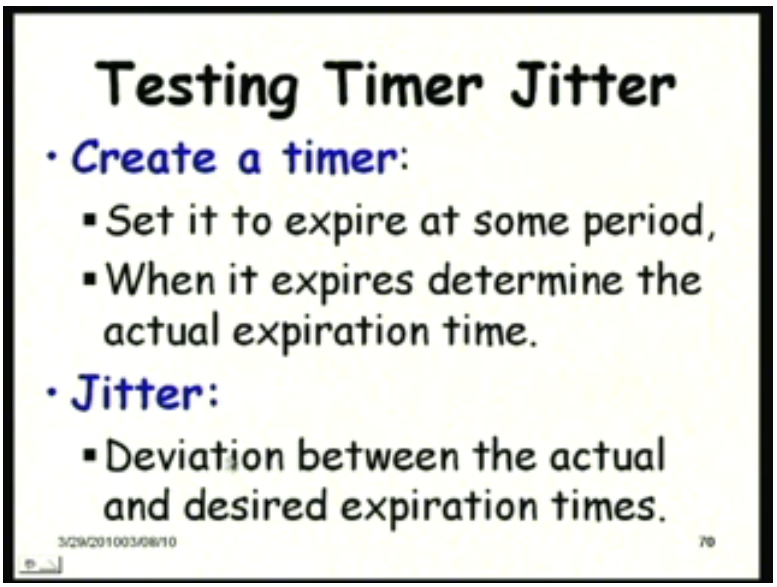
We are trying to execute a thread; that is a user thread; but here we are trying to execute a system call; and then trying to see how much time it takes to report that the time. And here we are measuring the maximum kernel blocking time. So, this is about testing the timer jitter. See, we have set a periodic timer. So, the timer period is between the time; it was set from time 0 to 100, 200, 300, 400 etcetera. But see at different times, the timer is being reported at different places right. See, here it is almost nearly 200; here it is three 320 or so, and so on.

(Refer Slide Time: 39:34)



So, the timer jitter is the maximum variation. So, here the maximum variation that has occurred; here the variation is less for each of them; here the variation is maximum. And that we are calling as the timer jitter. Let us see, how to setup an experiment to measure the timer jitter. First is to create a timer through system call. And while creating the timer, we set it to expire at some time; may be 1 millisecond, 10 milliseconds or 100 millisecond; basically, at few different values that are comparable to the tasks that we were or application that we were considering, we have to set it.

(Refer Slide Time: 40:29)



And then, when it expires, determine the actual expiration time. But how do you determine the actual expiration time? Say, set it to you know expire at one millisecond. How do you measure the actual expiration time? Yes, anybody would like to comment anything

Sir,

See the problem; here the problem that we were asking is that. See you are you are set up an experiment; you written a small C program. See, it can be a good small project also. You are trying to benchmark different real-time operating systems. So, what you have done is first parameter? You are trying to check is that, what is the accuracy of the timer interrupts? right You set a timer periodic timer, and what is the accuracy at which the it is reporting the events? right And here, you have just created a timer, and then trying to measure whether it has exactly given at the same time. But the question is that how do you measure?

See, you have set at 10 millisecond; let us say. So, it has given you timer interrupt at sometime right. So, then how do you know that whether it has given you exactly a 10 millisecond or it is 10.1 millisecond or 11 millisecond; how do you know? So, that is the question, we are asking. Yes, anybody would like to answer? Anything small experiment you can do it very simple experiment all all of these benchmarking programs; we will see simple experiments.

(( ))

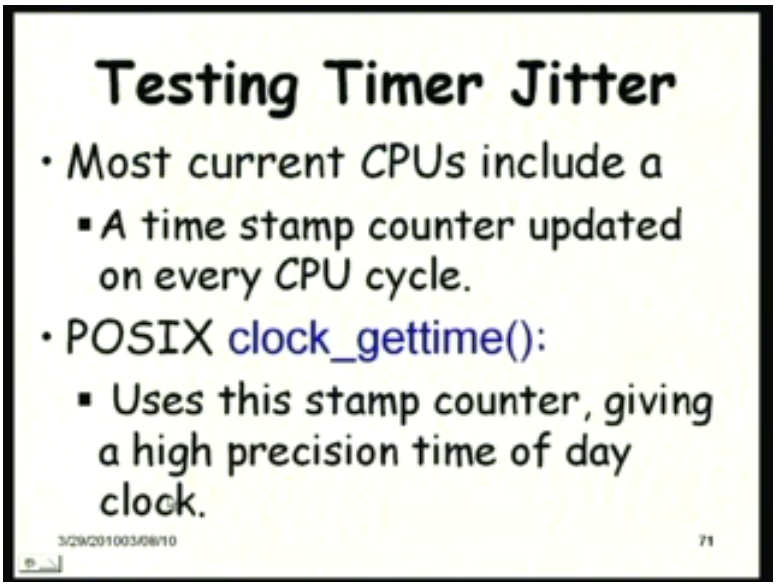
No, how do you compute the jitter? See, what he is saying is that, set it to expire at million number of times, and see what is the time at the last instant. And then see; No, that does not help, because see, it is each time, the timer gives a interrupt. How much time it takes to handle the interrupt? I mean report the interrupt. The interrupt is occurring. See, if you do that, the last one will also be not very far from the reading. And then you divide it by million; you'll get 0; not very hard; basically, you'll have to read the time value, isn't it?

So, most of the operating systems; they either have a hardware clock where you can read the time value directly or you can [re/read] read the time value the precise time value or

the clock reading. You can read the clock by through a system call or directly. Let us see that. So, the jitter is the deviation between the actual and the desired expiration times; it should be the maximum; the jitter is the maximum deviation. So, to compute jitter, we will have to consider all deviations, and then select the maximum among them. So, if you take just 2, 3 readings, then you may not notice the maximum.

You have to take 1000 of readings; 1000s of deviation values, and select the maximum jitter. So, all real-time operating systems, they would include a time stamp counter, and even in the posix R T; posix real-time, we have the clock get time system call. See, either you can directly read the counter or you can give a call; the clock get time call and this will give the high precision time of the day, which will be a large number. You can even try on your Linux machine; you use the clock get time; it'll report you a large number; basically, the number of clock readings that have expired, since the clock was booted; may be the clock reading right.

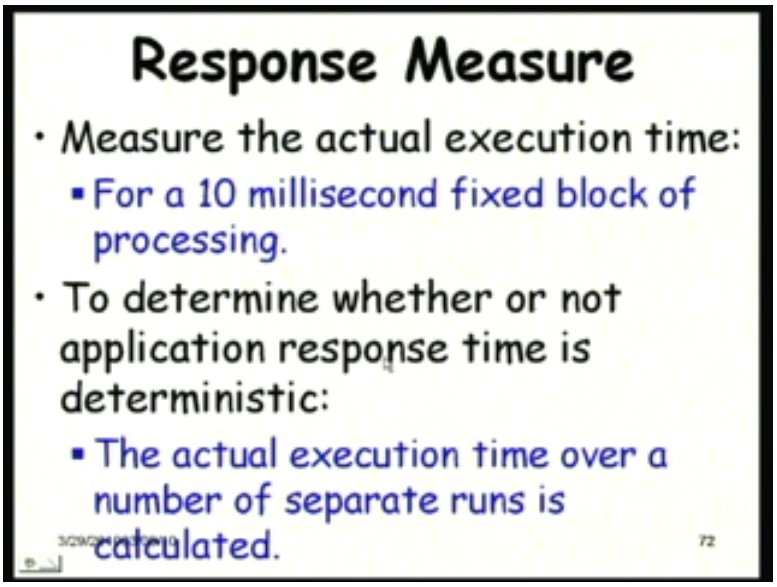
(Refer Slide Time: 44:39)



But what you are interested here is the difference between two times; it is not the clock time; it may be a large number not bothered about that, you have to take the difference. First take the clock reading, and then keep on taking the readings; each time the interrupt occurs, take the difference and that'll tell you how much time has expired, since the interrupts has been reported right. Now, let us see how to measure the response? So, here we have fixed block of processing. Let us say, some 10 millisecond or something which

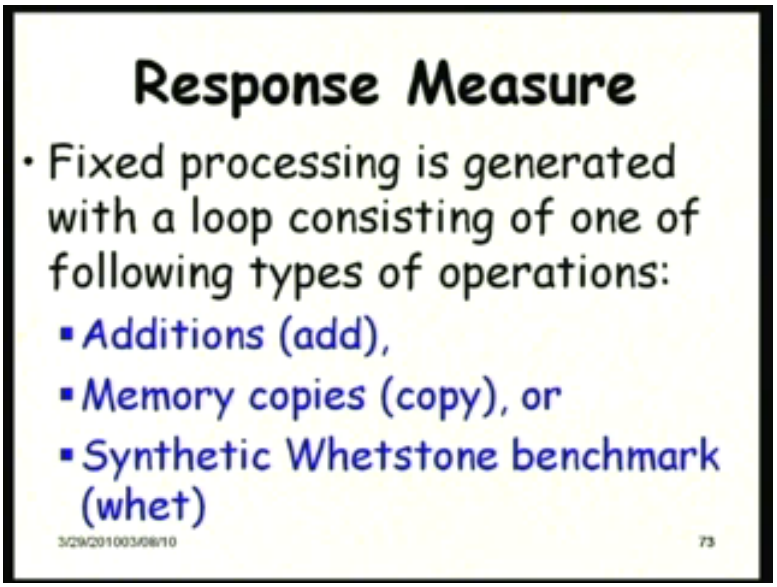
may be a you have written a small program with some ALU operations, (( )) etcetera or it may be some benchmark program, SPEC or may be a Dhrystone or something.

(Refer Slide Time: 46:01)



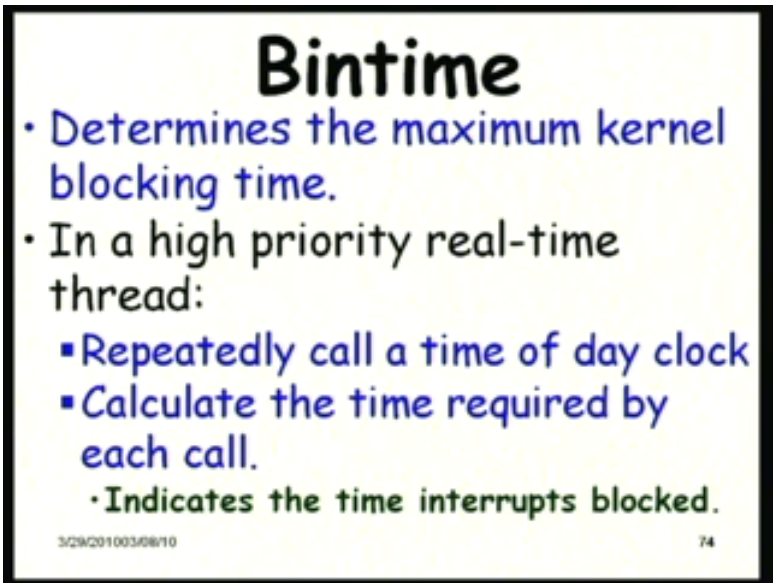
So, here we are trying to determine whether the time that it takes to execute is Deterministic or not? And remember that the task that you are running here is a real-time priority task; and a real-time priority task should not take different times at different instances of run; otherwise, know your scheduling routine you cannot really assume that, it takes that much amount of time. You have to take the maximum. In that case, sometimes it takes less time; sometimes it takes more time than your scheduling algorithms. In the analysis scheduling analysis, you'll you'll have to take the maximum time. So, here you will run it several times, and each time use the time of the day clock clock get time and measure how much time it takes?

(Refer Slide Time: 47:33)



So, you can try with different types of tasks; some may be some additions, subtractions, etcetera; another may be involving memory copy, data transfer and so on; another may be synthetic benchmark. Let us look at the bintime. So, the bintime, you you can use to find out, how much is the Kernel blocking time? It said that you must every system sometime or other the interrupts are disabled. And as long as in the kernel mode, it takes different amount of times, depending on what it is doing in the kernel or for systems that do not even consider this, how much time it is in the kernel mode maximum time. So, it may be in the worst case

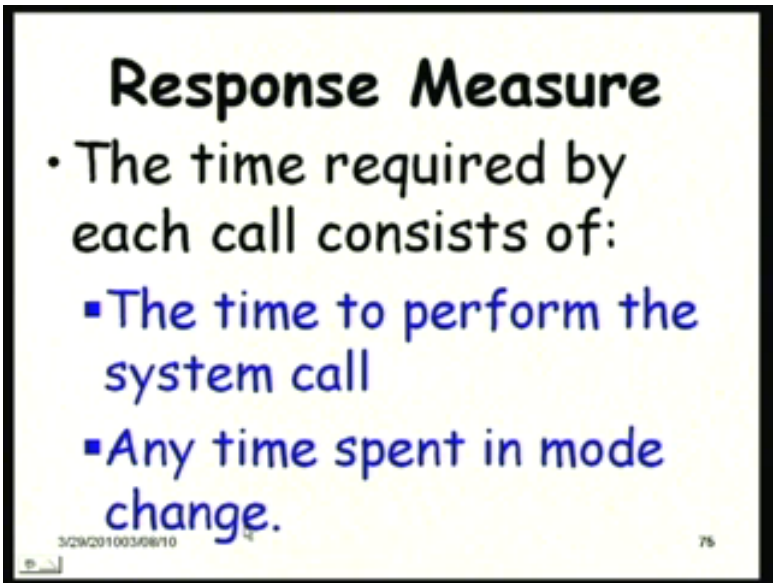
(Refer Slide Time: 47:59)



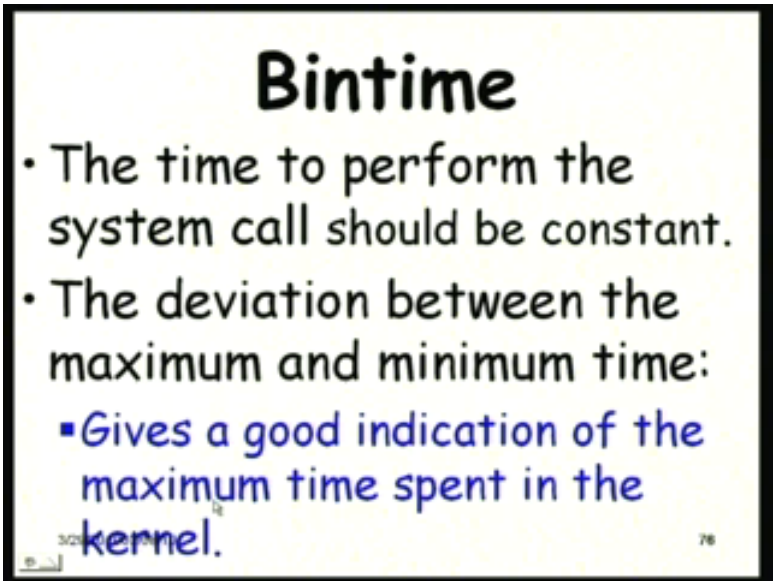
If you once you make a system call or another task is making a system call, how much time does it take to complete? So, repeatedly call the time of the day clock, and calculate the time required in each clock. And it would indicate the time the interrupts are blocked. So, each time you are just trying to execute something; a fixed routine and that too as a at the real-time priority level, and then you are finding that at different times, it is taking; the same one it taking different times And obviously, the kernel is getting blocked by some other system calls; of course, one thing that is remained unsaid here is that, you keep on calling the kernel with from another task; keep on giving system calls; otherwise if there are no system calls by another task, this may not show variation.

So, let us say there is a very low priority task, which is keeping on giving system calls or maybe it is doing IO, disc I O or something or networking. And then, this high priority task which is running and trying to give system calls. What is the variation in the time? So, here of course, if we decompose these times, it'll be the time to perform the system call, and the time to do mode change, if required. So, the deviation between the maximum and minimum time will give an indication of the time spent in the kernel.

(Refer Slide Time: 50:09)

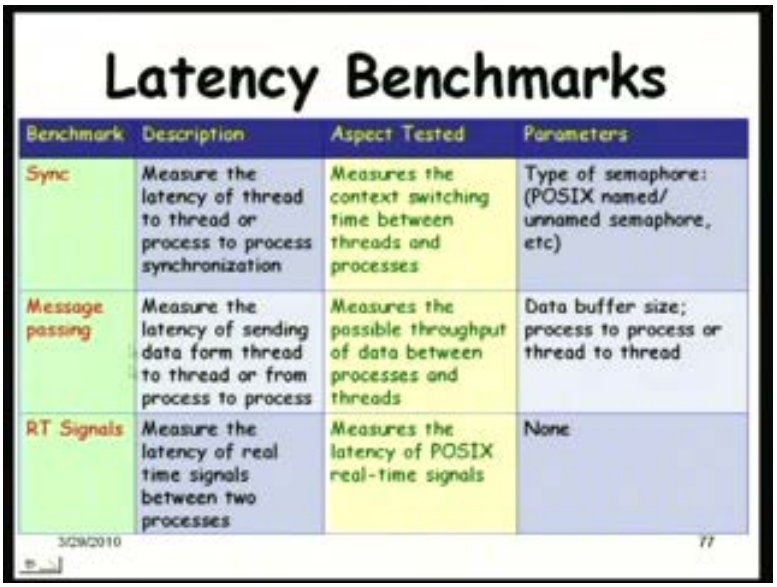


(Refer Slide Time: 50:24)



So, far we looked at the deterministic benchmarks, which basically tell us that whether timer runs deterministically; it reports the timer deterministically reports the events whether a thread runs deterministically, and whether each time you make a system call, there will be a variation in the time. Now, let us look at the latency benchmark; the latency benchmarks are sync, which is the which measures the latency of thread; thread to thread or process to process synchronization.

(Refer Slide Time: 50:36)



So, how much time? How much overhead is there in synchronization? Because resource sharing will be required, and if it the operating system has too much overhead in doing synchronization, then you'll have to factor in that time in your scheduling, and where we just assume that, it is almost nothing the synchronization time, is it not? I think, when we are doing the examples earlier, we had considered something on context switch and so on. Synchronization time - we had almost ignored this, while doing the analysis of resource sharing, but if it is a significant time the operating system implements a very inefficient synchronization mechanism; then, of course, our scheduling analysis will be wrong.

So, here it considers the context switch, and handling the resource. See basically, when it requests for a resource, it undergoes a context switch, and we will measure that time. So, the type of semaphores that will consider, I mean that can be considered is the posix, recommended named and unnamed semaphores; then the message passing which measures the latency of sending data from one thread to another thread. So, this is a simple parameter to measure. We will see how to measure all of these. So, this measures the possible throughput of data between processes and threads, and we will use different data buffers size.

And then we will measure the latency of real-time signals between two processes. So, once a real-time signal occurs, how much time does it take for the signal to be reported?

Because the different tasks will communicate different events through signals; so, the latency of the posix real-time signals will be measured. I think we were discussing about the posix real-time signals in the context of posix real-time; we had said that, the requirement, I mean the difference between a standard signal and the posix real-time signal; does anybody remember, what is the difference?

We said that the posix real-time standard, the signals has something extra compare to standard signal; so that is your home work just recall it that, how is the posix real-time signal different from standard posix signal. So, we will use the experiment; we use an experiment to measure the real-time signal latency. So, we will stop now, and we will continue from this point onwards in the next lecture; thank you