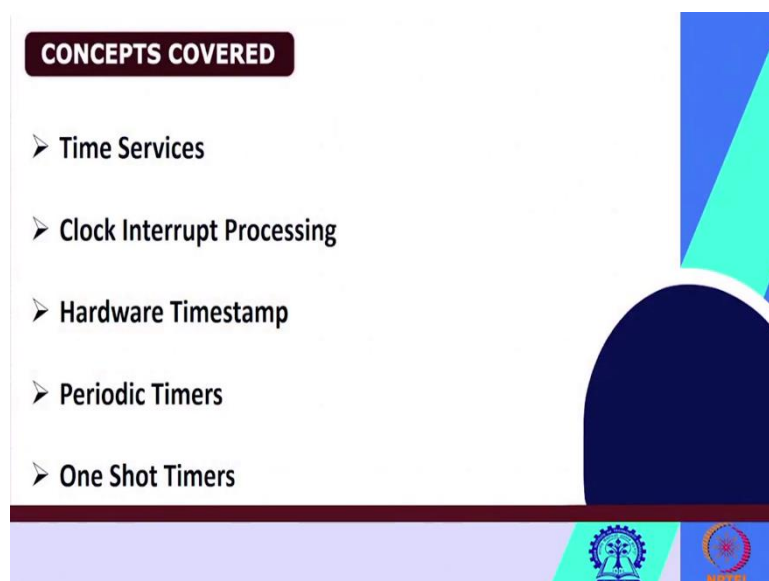


Real-Time Systems
Professor: Durga Prasad Mohapatra
Department of Computer Science and Engineering
National Institute of Technology Rourkela
Lecture 38: Time Services

Good morning to all of you. Today, now, let us continue where we have left from the last class. Last class, I have already told you, the requirements of real-time systems. We have seen so many basic requirements, such as, priority level and resource sharing, it should support resource sharing protocols, it should have, what, it should provide schedulers and the preemption some time, interruption latency and so on.

So, one of the requirements also there it was mentioned time services. So, real-time operating system should provide better time services. So, they should have timers, because that is the most important concept of real-time operating systems. So, today, we will take up about the time services in real-time operating systems. How the time services are provided in real time operating systems?

(Refer Slide Time: 01:19)



KEYWORDS

- System Clock
- Clock Resolution
- Clock Interrupt
- Jitter
- Timers

The slide features a dark red header with the word 'KEYWORDS' in white. Below it, a list of five terms is presented with right-pointing arrowheads. To the right of the text is a circular video inset showing a man in a suit. The slide has a decorative background with blue and green geometric shapes and logos for IIT Bombay and NPTEL at the bottom.

We will see, how time services are provided in the real-time operating systems. We will see, a little bit about the clock interrupted processing, some basic concepts of hardware timestamp, then, what are the different timers available in real-time operating systems. Basically, two kinds of timers are available in real-time operating systems. One is, periodic timer another is one shot timer. We will see, about terms like system clock or clock resolution, clock interrupt, jitter, timers so those kinds of keywords, we will use here.

(Refer Slide Time: 01:54)

TIME SERVICES

- RTOS provides clocks and time-services to programmers.
- Time services provided by OS are based on software clock known as **system clock**.
- **System clock is maintained by OS kernel based on the interrupts received from hardware clock.**
- **System clock should have fine resolution to handle the issue of timing constraints.**
- Currently, the resolution of hardware clocks is finer than a nano-second.

The slide features a dark red header with the word 'TIME SERVICES' in white. Below it, a list of five bullet points is presented. The second, third, and fourth points are highlighted in blue and red respectively. To the right of the text is a circular video inset showing a man in a suit. The slide has a decorative background with blue and green geometric shapes and logos for IIT Bombay and NPTEL at the bottom.

Now, let us see the first one, that is how, the time services are provided in real-time operating systems. Real-time operating systems they provide clocks and time services to the programmers. This I have already told you, so every real-time operating system they provide

clocks as well as time services to the programmers. So, now, let us see how the time services are provided.

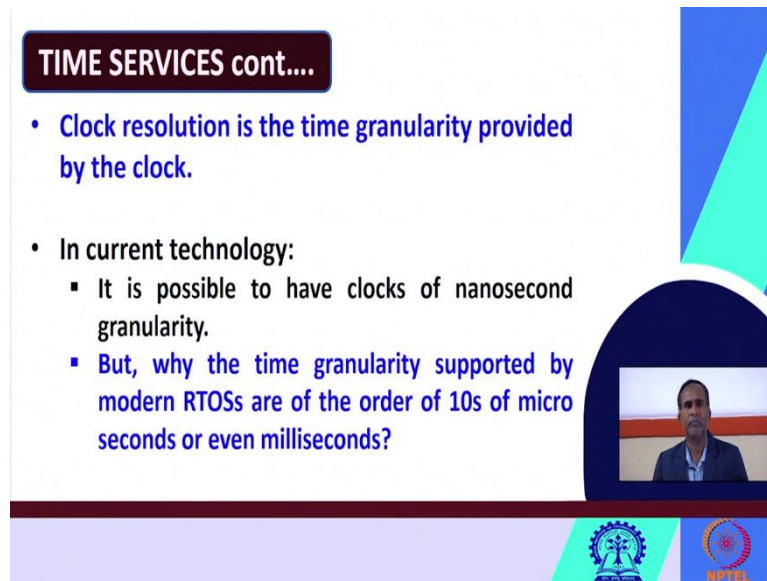
The time services provided by the operating systems they are based on a software clock, which is known as system clock. Who maintains that system clock? You know that every operating system has a kernel. The system clock is maintained by the operating system kernel based on the interrupts, which are received from the hardware clock. So, in every computer, there is a hardware clock.

And now, I have already told you the operating system, it provides this time services through a software clock, known as system clock. So, this system clock is maintained by the operating system kernel. Based on what? Based on the interrupts received from the hardware clock. The system clock should have fine resolution to handle the issue of timing constraints.

In real-time systems, very important is, how to achieve the timing constraints. There are three key timing constraints, you might have known. Three Ds, deadline, delay and duration, these are the timing constraints in real-time operating systems. So, in order to handle the issue of these timing constraints, such as, deadline, delay and duration this system clocks should have a very fine resolution, the system clock should have a very fine resolution to handle the issue of the timing constraints.

So, if we will see, what the resolution of current hardware clocks currently or they present the computer system, they present the computer systems, competently present the computer systems. The resolution of a hardware clock is finer than a nanosecond. I hope, you know about these terms second, millisecond, microsecond, nanosecond. Please see those things yourself the formula for them, the conversion of them. So, currently, the resolution of the harder clocks is even finer than a nanosecond. What, about the worth, the resolution of the software clock or the system clock? Let us look at.

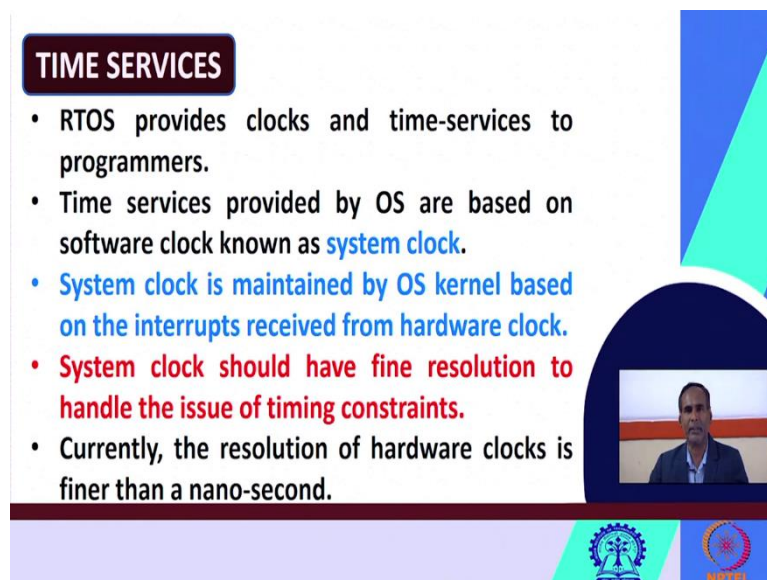
(Refer Slide Time: 04:10)



TIME SERVICES cont....

- Clock resolution is the time granularity provided by the clock.
- In current technology:
 - It is possible to have clocks of nanosecond granularity.
 - But, why the time granularity supported by modern RTOSs are of the order of 10s of microseconds or even milliseconds?

The slide features a video inset of a man speaking, a decorative blue and green geometric shape on the right, and logos for IIT Bombay and NPTEL at the bottom.



TIME SERVICES

- RTOS provides clocks and time-services to programmers.
- Time services provided by OS are based on software clock known as **system clock**.
- **System clock is maintained by OS kernel based on the interrupts received from hardware clock.**
- **System clock should have fine resolution to handle the issue of timing constraints.**
- Currently, the resolution of hardware clocks is finer than a nano-second.

The slide features a video inset of a man speaking, a decorative blue and green geometric shape on the right, and logos for IIT Bombay and NPTEL at the bottom.

The clock resolution. What do you mean by clock? By this time, as I have already told you, the resolution of hardware clocks is finer than a nanosecond. But what do you mean by resolution, clock resolution? Clock resolution is, the time granularity provided by the clock. What is the granularity? So, clock resolution means, it is the time granularity provided by the clock.

In current technology, it is very much possible to have clocks of nanosecond granularity. I have already told you the hardware clock their resolution is even finer than a nanosecond. So, in current technology, it is possible to have clocks of nanosecond granularity. But what happens here? While the time granularity supported by modern real-time operating systems they are of the order of 10s of a microseconds or even milliseconds.

They are not of the order of nanosecond. In hardware, the hardware clocks they are having finer resolution maybe of the order of nanosecond. But the modern real-time operating systems, their clock resolution is of the order of 10s of microseconds or even milliseconds. Why? Why they are not achieving the granularity of nanosecond? Why do not have a clock resolution of the order of nanosecond? Let us see.

(Refer Slide Time: 05:25)

TIME SERVICES cont...

- **The clock resolution available in modern RTOS is order of several milliseconds or worse. Why?**
- Hardware clock periodically generates interrupts (known as **time service interrupts**).
- After each clock interrupt, the kernel updates the software clock and performs certain other functions, e.g. updating time execution budget.
- A task can read the time:
 - Using the POSIX function `clock_gettime()`.

So, I have already told you, the clock resolution available in a modern real-time operating systems is order of several milliseconds or even if worse not of the order of nanosecond. Why? Let us see the reason. Why it is not able to provide, what resolution of the order of nanoseconds? Let us analyze the reasons.

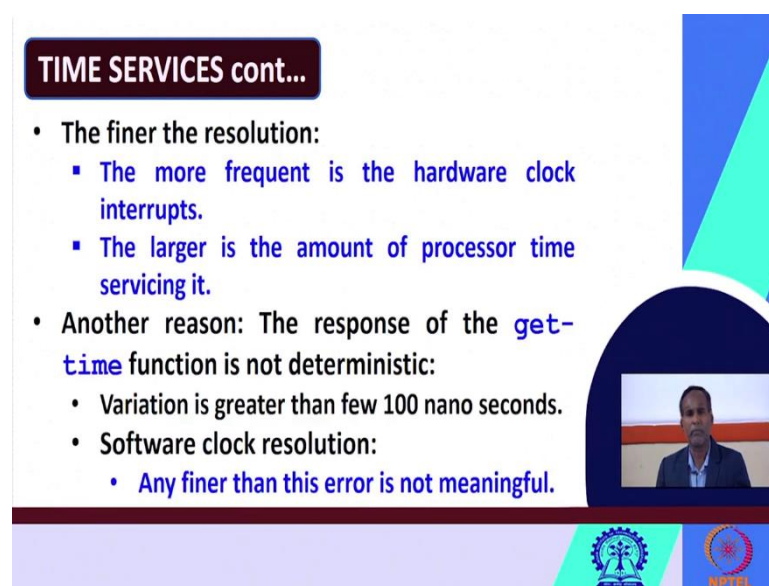
You know that the hardware clock, it periodically generates interrupts, is it not it. The hardware clock it periodically generates interrupts, those interrupts are known as, what, time service interrupts So, the interrupts provided, the interrupts which are periodically generated by the hardware clock they are known as time service interrupts.

Now, what happens, after each clock interrupt, the kernel updates the software clock. After each what clock interrupt may be which is generated by the hardware clock then the kernel of the operating system it updates the software clock. And it performs certain other activities, certain other functions, such as, updating the time execution management or updating the ready queue, those kinds of things they are what processed.

Now, a task can read the time. How? In real-time operating systems how a task can read the time? A task can read the time by using the POSIX function what is POSIX, we will discuss in the next class. So, a task can read the time using a function which you call as using a POSIX function which we denote as a clock gate time.

So, a task in the real-time systems can be read, a task can read the time using the function clock get time. Now, what happens, since, it is using a function called clock get time let us see how it is affecting the resolution.

(Refer Slide Time: 07:09)



TIME SERVICES cont...

- The finer the resolution:
 - The more frequent is the hardware clock interrupts.
 - The larger is the amount of processor time servicing it.
- Another reason: The response of the **get-time** function is not deterministic:
 - Variation is greater than few 100 nano seconds.
 - Software clock resolution:
 - Any finer than this error is not meaningful.

The slide features a dark blue header with the title 'TIME SERVICES cont...' in white. The main content is a bulleted list in blue text. A small video inset on the right shows a man speaking. At the bottom, there are logos for a university and 'NPTL'.

The finer the clock resolution, the more frequently the hardware clock interrupts. So, how much finer you will make the resolution, the more frequently the hardware clock interrupts. And if there will be, the frequency of the hardware clock interrupts will be more then there will be huge amount of processor time will be spent.

The larger the amount of processor time servicing it. The finer the resolution, the more frequent will be the harder clock interrupts. And if the more frequent will be the harder clock interrupts, then the larger will be the amount of processors time servicing it. This is one reason, why the software clocks they do not have what finer clock resolutions.

Another reason, is that, I have already told you that the task can read the time using this function clock get-time. So, the response of the get time or the clock get-time function is not deterministic. It may vary, the variation may be greater than a few 100 nanoseconds, that variation may be greater than 100 nanoseconds.

So, any software clock resolution. So, the software clock resolution, anything finer than this error, that means, I have already told you, this difference is of the order of 100 nanoseconds. So, any finer resolution than this errors, this difference this variation is not meaningful at all. So, software clock resolution, any finer resolution than this error may be of the order of this 100 nanosecond this different, this error is not meaningful at all.

So, that is why the clock resolutions in real-time operating systems, they are not of the order of nanoseconds, they are of the order of milliseconds or even microseconds.

(Refer Slide Time: 09:04)

TIME SERVICES cont...

- In fact, every system call invocation has some associated jitter. (Jitter is defined as the difference between the worst-case response time and the best-case response time.)
- Jitter is caused partly on account of interrupts having higher priority than system calls. During interrupt calls, system call processing is stopped.
- The problem gets aggravated in such a situation.

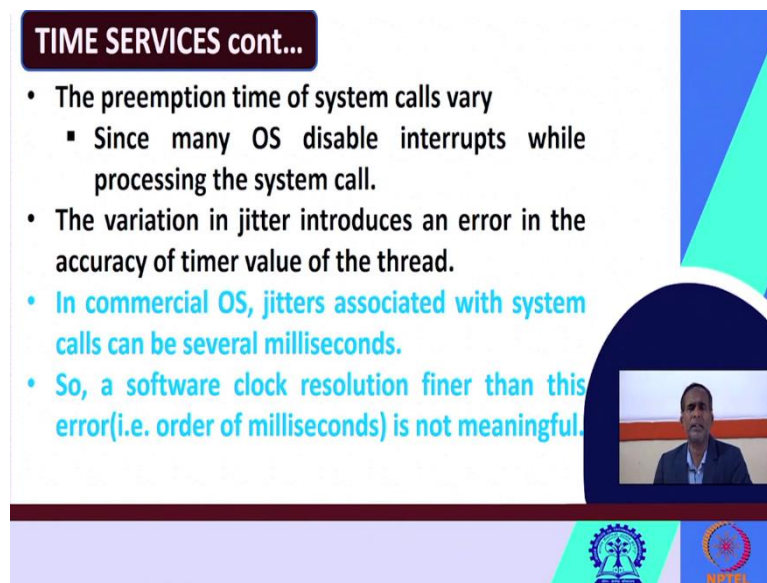
The slide features a video inset of a man speaking, and logos for IIT Delhi and NITEL at the bottom.

In fact, every system call invocation has some associate jitter. So, whenever you are invoking any system call, it has some data. So, in fact every system call invocation has some associate jitter. So, what do you mean by jitter? Jitter is what? It is defined as the difference between the worst-case response time and the best-case response. Now, this detailed how it is caused? This jitter is caused by, the jitter is caused partly on account of the interrupts having higher priority than the system calls.

So, if there are some interrupts, which has higher priority than the system calls then, of course, jitter may happen. So, a jitter is caused partly on account of the interrupts which are having higher priority than the system calls. But during the interrupt call system call processing it stopped. So, whenever the interrupt call is there, system call processing it stopped. So, this problem, the problem gets aggravated in such a situation.

That means, when, what will happen, these interrupts are having higher priority than the system calls. And during the interrupt calls the system call processing is stopped. The problem gets aggravated in such a situation. So, these are some of the reasons, why the clock resolution in case of the real-time operating systems it is not such much finer, they are not of the order of nanosecond, they are of the order of the what even milliseconds or microseconds.

(Refer Slide Time: 10:44)



TIME SERVICES cont...

- The preemption time of system calls vary
 - Since many OS disable interrupts while processing the system call.
- The variation in jitter introduces an error in the accuracy of timer value of the thread.
- In commercial OS, jitters associated with system calls can be several milliseconds.
- So, a software clock resolution finer than this error (i.e. order of milliseconds) is not meaningful.

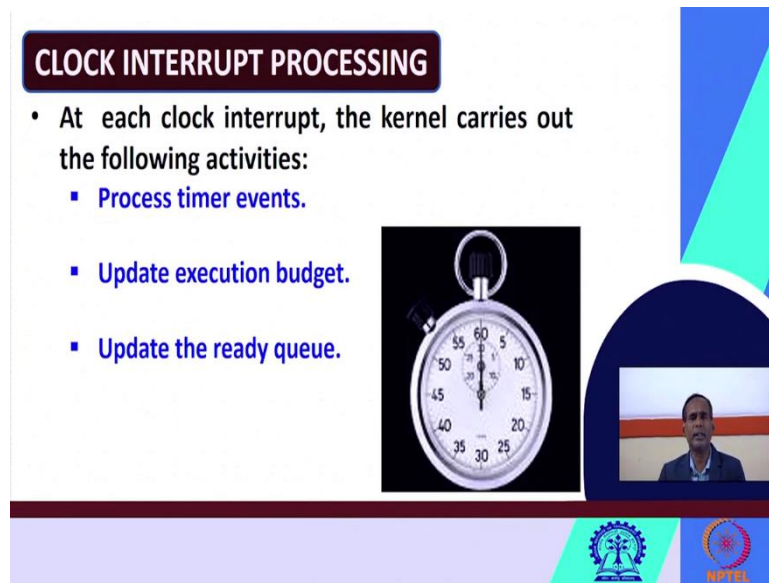
The slide features a dark blue header with the title 'TIME SERVICES cont...' in white. The main content is a list of four bullet points. The first two are in black, and the last two are in light blue. A small video inset in the bottom right shows a man speaking. The slide is decorated with a blue and green geometric shape on the right and logos for IIT Bombay and NPTEL at the bottom.

The preemption time of system calls vary. So, if we will see, the preemption in the real-time system you have already known what is preemption. The preemption time of the system calls may vary, since many operating system disable the interrupts while processing the system calls. So, what happens, many operating systems they disable the interrupts while they are processing the system call, and hence, the preemption time of the system calls may vary.

The variation in jitter introduces an error in the accuracy of the time value of the thread. So, whenever there will be variation, the variation in the jitter it introduces an error. Where? It introduces an error in the accuracy of the timer value of the thread. So, some errors will be introduced in the accuracy of the timer or in the accuracy of the timer value.

In commercial operating systems, the jitters associated with the system calls can be of several milliseconds. So, if you will see, some of the commercial real-time operating systems, the jitters associated with the system calls can be of the order of several milliseconds. Hence, any software clock resolution finer than this error, that means, of the order of several milliseconds is not at all meaningful, that is why, the software clock resolution in real-time operating systems is the order of milliseconds or microseconds, not of the order of nanoseconds.

(Refer Slide Time: 12:07)



CLOCK INTERRUPT PROCESSING

- At each clock interrupt, the kernel carries out the following activities:
 - Process timer events.
 - Update execution budget.
 - Update the ready queue.

The slide features a stopwatch image and a small video inset of a speaker. Logos for IIT Bombay and NPTEL are visible at the bottom.

Now, next, we will go to the next item, that is, clock interrupt processing. So, let us see, what is happening during this clock interrupt processing. At each clock interrupt the operating system kernel it carries out some activities. What are the activities? It carries out the following activities. So, it processes the timer events, then, it updates the execution budget, and it also updates the ready queue. In this way, this will happen. Let us see, how does it happen?

(Refer Slide Time: 12:46)

PROCESS TIMER EVENTS

- The timer queue:
 - Contains all timers arranged in order of their expiration times.
- At each clock interrupt:
 - The kernel checks if any timer event has occurred.
 - Kernel processes all timer events:
 - Queues the specified actions.

The diagram shows a horizontal bar representing a timer queue with segments of red, green, and blue. Below the bar are several colored circles (green, red, blue) labeled 'Handlers'. A small video inset shows a man speaking. The slide footer includes navigation icons and logos for IIT Bombay and NPTEL.

So, first, let us see processor time events. So, how to process the timer events. Let us see. So, the timer queue, I have already told you so, now, these timer events or the timers the timer events they will be arranged in a queue, the timer queue, it contains all the timeouts arranged in order of their expiration time.

So, since in the operating system, you will see several times the timers are there, these timers will be arranged in the form of a queue, we call as a timer queue. And how they will be arranged? They will be arranged in order of their expiration times. When? Which timer will expire when? So, in based on that expiration times, in order their expiration times, all the timers are in the queue, and that we call a timer queue.

So, the timer queue it contains on the timers arranged in order of their expiration times. Now, at each interrupt clock interrupt what happens? At each clock interrupt, the kernel it checks if any timer event has occurred. So, whenever any clock interrupts occurs the operating system, it got the operating system kernel, it checks if any timer event has occurred.

The operating system kernel processes, all timer events and queues the specified action. So, the kernel it processes all the timer events, and then, it puts in a queue. And it queues all the specified functions, all the special actions. Where it will put? In a queue. It queues all the specified actions. So, let us see how does it look like? It will look like this as shown in the above figure.

So, you see. So, here, these boxes, they represent the expiration times because I have already told you, the timers are not in the order of their expiration times, so these boxes are the expiration times. The timers are arranged in a queue, so each timer has a handler. See, each timer has a handler.

This is a handler, this is handler, and that might be handler 1, handler 2, like this. So, all the timers are arranged in the timer queue in the order of their expiration times. Whenever each clock interrupt occurs, the kernel checks if any timer event has occurred or not. Then the console processes all the timer events, and it gives the specified actions. It puts all these specified actions in the queue. It queues them.


(Refer Slide Time: 15:01)

UPDATE READY QUEUE

- Some tasks may have become ready:
 - If a ready task has higher priority than executing task:
 - The executing task is preempted.
 - Otherwise, it is inserted in the ready queue.

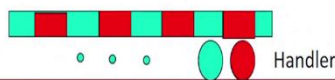
CLOCK INTERRUPT PROCESSING

- At each clock interrupt, the kernel carries out the following activities:
 - Process timer events.
 - Update execution budget.
 - Update the ready queue.




PROCESS TIMER EVENTS

- The timer queue:
 - Contains all timers arranged in order of their expiration times.
- At each clock interrupt:
 - The kernel checks if any timer event has occurred.
 - Kernel processes all timer events:
 - Queues the specified actions.



The diagram shows a horizontal bar representing a timer queue with alternating red and green segments. Below the bar are several small circles: three green, one red, and one red. To the right of these circles is the word 'Handlers'.



Next, we will go to the second activity. That is, the update the ready queue. So, some tasks may have become ready. Actually, what happens, so this is a basic concept of operating systems, you might have read already earlier. So, some tasks may have become ready, but what happens, some other tasks are arriving and its priority is more than the current task.

So, if a ready task has higher priority than the currently existing task, what will happen then, the currently existing task is preempted. Otherwise, what will happen? Otherwise it is inserted in the ready queue. If that ready task has less priority, then what will happen, that task is inserted in the ready queue. So, who updates this ready queue? Who updates this ready queue? I have already told you, that at each call clock interrupt the operating system carries out the following activities.

That means, the operating system kernel, it processes the timer events, the operating system kernel, it updates the ready queue and finally operating system kernel it updates the execution budget. So, we have seen here how the operating system kernel it processes the timer events, then we have told you how the operating system kernel it updates the ready queue. And finally, we will see, the last activity that is how the kernels it updates the educational budget.

(Refer Slide Time: 16:16)

UPDATE EXECUTION BUDGET

- After each clock interrupt:
 - The scheduler decrements the remaining time slice (budget) of the executing task.
- If the task is not complete and the slice (budget) becomes 0:
 - The task is preempted.
 - Scheduler is invoked to select another task to run.

The slide features a video inset of a man speaking, and logos for IIT Bombay and NPTEL at the bottom.

After each clock interrupt, what happens, the scheduler it decrements the remaining time slice available of the executing tasks. So, after each clock interrupt, the scheduler what it does, it decrements. Decrements what? The remaining time available or the remaining time slice available or the remaining time budget of the executing task.

If the task is not complete, and the slice becomes zero, what happens, that means, still, the task is not completed, it will require more time, but its time slice, its budget is 0. So, what will happen, the task has to be preempted, and the scheduler is enable to select another task to run. Is it not it? And then, this preempted task, it will be executed later on when time will be available.

So, if the task is not complete, and the slice, time slice or the time budget it is it becomes 0 then the task is preempted, the currently executing task is preempted and the scheduler is invoked to select another task from the ready queue to run. And this currently preempted task it will again run when later on when time will be available.

(Refer Slide Time: 17:34)

PROVIDING HIGH CLOCK RESOLUTION

- We have already seen that there are 2 difficulties in providing high resolution timer.
 - The overhead associated with processing the clock interrupt becomes excessive as the time service resolution gets finer.
 - The jitter associated with the time look up system call (`clock_gettime()`) is often of the order of several milliseconds.

UPDATE EXECUTION BUDGET

- After each clock interrupt:
 - The scheduler decrements the remaining time slice (budget) of the executing task.
- If the task is not complete and the slice (budget) becomes 0:
 - The task is preempted.
 - Scheduler is invoked to select another task to run.

So, who does this updation of the execution budget? So, this, who does? This operating system kernel, it updates the execution, time execution budget. Now, let us say, how to provide high clock resolution? I have already told you that what happens, there are two difficulties, two major difficulties in providing high resolution timer.

One, the overhead associated with processing the clock interrupt it becomes very much excessive. The time service resolution gets finer, this we have seen. Second, I have already told you, that the jitter associated with the time lookup system call. What is the system, that function or the system called clock get-time.

It is often of the order of the several milliseconds not nanoseconds. So that is why, any error beyond this milliseconds it will not be meaningful. So that is why the software clock is having clock resolution of the order of milliseconds or microseconds not what such finer like nanoseconds.

(Refer Slide Time: 18:45)

PROVIDING HIGH CLOCK RESOLUTION

- Therefore, it is not useful to provide a clock with a resolution any finer than this.
- But, some R-T applications need to deal with timing constraints of the order of a few nano seconds.
- **Is it possible to support time measurement with nano second resolution?**

PROVIDING HIGH CLOCK RESOLUTION

- We have already seen that there are 2 difficulties in providing high resolution timer.
 - **The overhead associated with processing the clock interrupt becomes excessive as the time service resolution gets finer.**
 - **The jitter associated with the time look up system call (clock_gettime()) is often of the order of several milliseconds.**

So, therefore, it is not at all useful to provide a clock with a resolution any finer than this. I have already told you these are the two major difficulties in providing high resolution timer, therefore, it is not useful to provide a clock with the resolution finer than this. But, still, there are some real time applications they need to deal with timing constraints of the order of a few nanoseconds. Then what will happen?

Then the question is, is it possible to support the time measurement with a nanosecond resolution for these types of real time applications? Let us see the answer.

(Refer Slide Time: 19:19)

PROVIDING HIGH CLOCK RESOLUTION

Solution:

- A way to provide sufficiently fine clock resolution is mapping a hardware clock into the address space of applications.
- An application can then read the hardware clock directly through a normal memory read operation without having to make a system call.

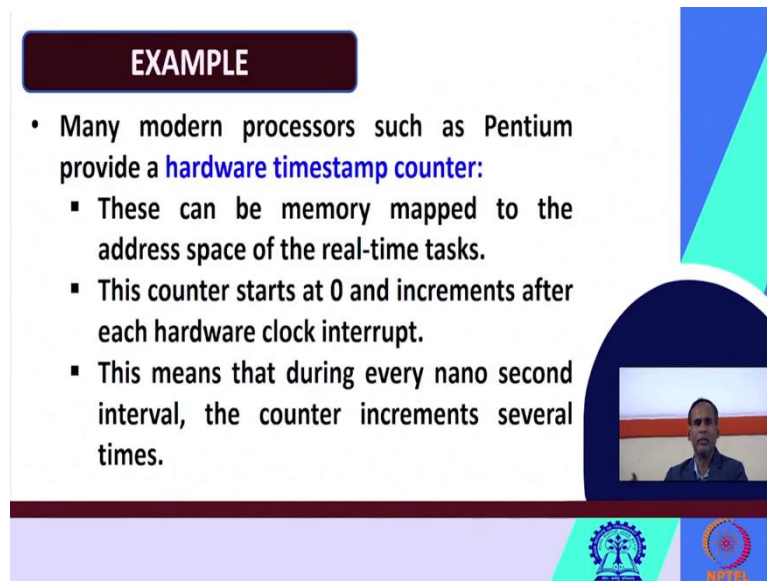
The slide features a dark blue header with the title in white. The main content is on a white background with blue text. A small video inset of a speaker is visible on the right side. The bottom of the slide has a purple and blue gradient bar with logos for IIT Bombay and NPTEL.

The answer is, yes, can be done. What is the solution? How we can provide high clock resolution for those types of real-time applications? So, what you can do, let us say there is a solution. One way to provide sufficiently finer clock resolution is what, is mapping a hardware clock into the address space of applications.

I hope, you have already known this address space in your what may be in operating system or somewhere else. So, one way to provide sufficiently finer clock resolution is, through mapping a hardware clock into the address space of the applications. Then, an application can then read the hardware clock, directly through what, directly through a normal memory read operation without having to make a system call. So, this is a simple example, this is the simple way we have told, how to provide the high clock resolution.

First, you have to map a hardware clock into the address space application, then an application, it can read, the order clock directly through what, through a normal memory read operation without having to make a system clock. So, this is the way by which you can provide high clock resolution or finer clock resolution.

(Refer Slide Time: 20:39)



EXAMPLE

- Many modern processors such as Pentium provide a **hardware timestamp counter**:
 - These can be memory mapped to the address space of the real-time tasks.
 - This counter starts at 0 and increments after each hardware clock interrupt.
 - This means that during every nano second interval, the counter increments several times.

The slide features a video inset of a man speaking, and logos for IIT Bombay and NIPTEL at the bottom.

We will take a small example. Many modern processors, such as, Pentium etc, they provide a hardware timestamp counter. So, they provide a hardware timestamp counter, these can be memory-mapped. So, many modern processors such as Pentium they provide a hardware timestamp counter, these can be easily memory-mapped to the address space of the real-time task.

Now this counter, what does it do, this timestamp counter it starts at 0 and then it increments after each hardware clock interrupt. So, after each hardware clock interrupt occurs, it increments by 1. This counter starts at 0 and increments after each hardware clock interrupt is there. This means, that during every nanosecond interval, what will happen, the counter increments several times.

So, as I have already told you, these counter starts 0 and increments after each hardware clock interrupt. Now, this means, that what, during every nanosecond interval this counter it increments several times. So, in this way, we can provide what higher clock resolution, finer clock resolution to those types of real-time application, real-time applications.

(Refer Slide Time: 21:52)

The image shows two presentation slides. The top slide is titled "EXAMPLE cont ..." and lists a limitation: "Limitation: Has portability problems." It includes two bullet points: "e.g. when application running on a Pentium processor is ported to a different processor, the new processor may not have a high resolution counter, and" and "the memory address map and the resolution would differ." The bottom slide is titled "EXAMPLE" and lists that many modern processors provide a "hardware timestamp counter." It includes three bullet points: "These can be memory mapped to the address space of the real-time tasks.", "This counter starts at 0 and increments after each hardware clock interrupt.", and "This means that during every nano second interval, the counter increments several times." Both slides feature a video inset of a speaker and logos for IIT Bombay and NPTEL.

EXAMPLE cont ...

- **Limitation: Has portability problems.**
 - e.g. when application running on a Pentium processor is ported to a different processor, the new processor may not have a high resolution counter, and
 - the memory address map and the resolution would differ.

EXAMPLE

- Many modern processors such as Pentium provide a **hardware timestamp counter**:
 - These can be memory mapped to the address space of the real-time tasks.
 - This counter starts at 0 and increments after each hardware clock interrupt.
 - This means that during every nano second interval, the counter increments several times.

It has a limitation. What is the limitation? It has a portability problems. Because I have told you, you have to use a what hardware timestamp counter, it has portability problems. For example, when any application running on a Pentium processor, it is ported to some another some other or to a different processor. What will happen?

The new processor may not have a high-resolution counter, then what will happen, and automatically the memory address map and the resolution would differ, is it not it. So, because where we are running our application, it may be a Pentium processor, but when report it a or transferred to a different processor, the new processor may not have a high-resolution counter.

And this will lead to the fact that the memory address map and the resolution would differ. So, this is one of the drawback of this solution that we have provided that you can use some hardware timestamp counter. So, this is the drawback of this method.

(Refer Slide Time: 22:50)



TIMER SERVICES

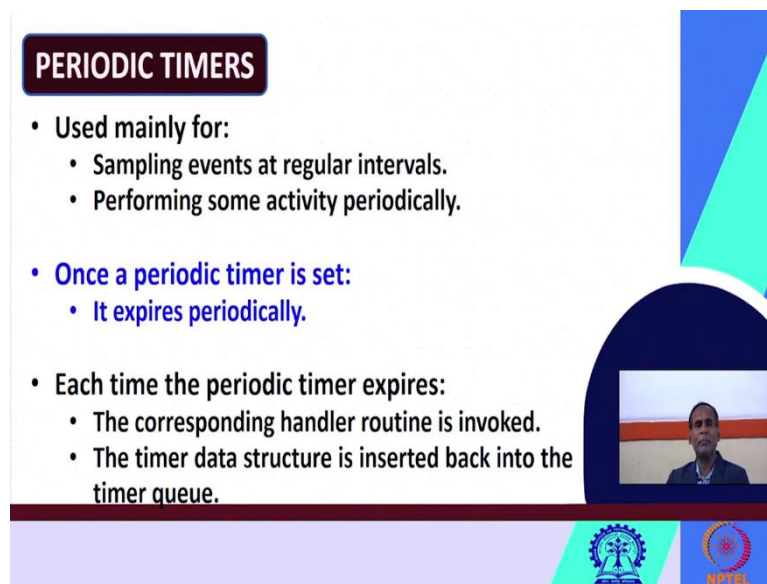
- **Timer service:**
 - A vital service provided to applications by real-time operating systems.
- **There are 2 types of timers:**
 - **Periodic.**
 - **One shot (or aperiodic).**

The slide features a dark blue header with the title 'TIMER SERVICES' in white. Below the title is a bulleted list. A video inset in the bottom right corner shows a man speaking. The slide is decorated with a blue and green geometric design on the right side and logos for IIT Bombay and NPTEL at the bottom.

Now, let us see the other important aspect. The time services. I have already told you time service is a very vital service provided to the applications by real-time operating systems. So, timer service is one of the important services provided to the different applications by real-time operating systems.

There are two types of timers actually available. So, this timer service, as I have already told you this an important service provided to the applications by real-time operating systems. There are two important types of timers. One, periodic timer, two, aperiodic timer or it is also known as one shot timer. Let us see, the details of periodic timer and aperiodic timer.

(Refer Slide Time: 23:37)



PERIODIC TIMERS

- Used mainly for:
 - Sampling events at regular intervals.
 - Performing some activity periodically.
- Once a periodic timer is set:
 - It expires periodically.
- Each time the periodic timer expires:
 - The corresponding handler routine is invoked.
 - The timer data structure is inserted back into the timer queue.

The slide features a dark blue header with the title 'PERIODIC TIMERS' in white. The main content is a list of three bullet points. The first bullet point is 'Used mainly for:' followed by two sub-bullets: 'Sampling events at regular intervals.' and 'Performing some activity periodically.' The second bullet point is 'Once a periodic timer is set:' followed by one sub-bullet: 'It expires periodically.' The third bullet point is 'Each time the periodic timer expires:' followed by two sub-bullets: 'The corresponding handler routine is invoked.' and 'The timer data structure is inserted back into the timer queue.' On the right side of the slide, there is a video inset showing a man speaking. At the bottom of the slide, there are two logos: the IIT Bombay logo on the left and the NPTEL logo on the right.

First, we will see periodic timers. So, this is used mainly for what? This is used mainly, for sampling events at regular intervals. If you want to collect some sample at regular intervals, you can use periodic timers. For example, you want to measure the temperature of a blast furnace at regular intervals then you can use this word periodic timer. So, it is normally used for sampling events at regular intervals, performing some activity periodically.

So, these timers, they are used for performing some kind of activities periodically. So, the basic concept behind using periodic timer, is that, once a periodic timer is set then it expires periodically. Once you set a periodic timer it expires periodically, maybe after every 10 milliseconds or after every say 10 microsecond like that. So, once a periodic timer is set it expires periodically.

Each time the periodic timer expires what happens, the corresponding handler routine is invoked. So, whenever the periodic timer it expires, what happens, the corresponding handler routine is invoked, the timer data structure is inserted back into the timer queue. We have already told you, that the timers they are arranged in a queue we call, as timer queue, so each time the periodic time multi experts the corresponding handler routine is invoked.

And the timer data structure is inserted back, where, it is inserted back into the timer queue. This is how the periodic timers they work. Now, let us say, about the aperiodic timers or these what we can say one shot timers.

(Refer Slide Time: 25:18)

ONE SHOT TIMERS

- Each time it is set:
 - Expires once only.
- **Example - Watchdog timers:**
 - An important type of one shot timers.

```
f(){  
  wd_start();  
  Process  
  wd_tickle()  
}
```

The slide features a dark blue header with the title 'ONE SHOT TIMERS' in white. Below the title is a bulleted list. A pink box contains a code snippet. On the right side, there is a video feed of a man speaking. The bottom of the slide has a purple bar with navigation icons and logos for IIT Bombay and NPTEL.

PERIODIC TIMERS

- Used mainly for:
 - Sampling events at regular intervals.
 - Performing some activity periodically.
- **Once a periodic timer is set:**
 - It expires periodically.
- Each time the periodic timer expires:
 - The corresponding handler routine is invoked.
 - The timer data structure is inserted back into the timer queue.

The slide features a dark blue header with the title 'PERIODIC TIMERS' in white. Below the title is a bulleted list. On the right side, there is a video feed of a man speaking. The bottom of the slide has a purple bar with navigation icons and logos for IIT Bombay and NPTEL.

So, this is something about the one shot timers or aperiodic timers. So, let us see the difference between the periodic timers and the one shot or the aperiodic timers. In periodic timers once a periodic timer is set, it expires periodically, maybe after every 10 milliseconds or so.

But in case of this one shot timers or aperiodic timers, each time it is set it expires only once. Each time the one shot timer is set, it expires, what, it expires once only. Let us see the example of a one shot timer. So, the very much popularly what common example is watchdog timer. So, watchdog timers are the very popular examples of one shot timers. This is an important type of one shot timers.

(Refer Slide Time: 26:08)

ONE SHOT TIMERS

- Typically used to detect if a task misses its deadline:
 - Then initiate exception handling procedures upon a deadline miss.

```
f(){  
  wd_start();  
  Process  
  wd_tickle()  
}
```

PERIODIC TIMERS

- Used mainly for:
 - Sampling events at regular intervals.
 - Performing some activity periodically.
- Once a periodic timer is set:
 - It expires periodically.
- Each time the periodic timer expires:
 - The corresponding handler routine is invoked.
 - The timer data structure is inserted back into the timer queue.

Typically, why it is used? I have already told you. Normally, these periodic timers they are used for sampling events at regular intervals, and they are also used for performing some activity periodically. But why, the answer timers are used? They are normally used to detect if a task misses its deadline. So, you know, I have already told you there are three important Ds in real-time operating system, deadline, duration and delay.

So, how to know, that a task has missed the deadline? So, by using a one shot timer you can see, you can determine if a task has missed its deadline. And if the timer gives the indication that the task has missed the deadline then what will happen, then it initiates the exception handling procedures based on a deadline miss.

So, when this one shot timer it indicates that a task has miss the deadline then it initiates exception handling procedures upon a deadline miss. So, whenever a deadline is missed, the one shot timers it has detected then it initiates the different exception handling procedures. So, here I have shown a simple example, let us see, how does it work.

(Refer Slide Time: 27:25)

ONE SHOT TIMERS

- In the following figure, a watchdog timer is set for critical function f() through a wd_start(t1) call.
- The wd_start(t1) sets the watchdog timer to expire by the specified deadline (t1 time units) from the starting of the task.
- If function f() does not complete even after t1 time units have elapsed, then the watchdog timer expires, indicating that the task deadline is missed & the exception handling procedure is initiated. If the task completes within deadline, then watchdog timer is reset using a wd_tickle() call.

```
f(){  
  wd_start();  
  Process  
  wd_tickle()  
}
```

The slide also features a video inset of a speaker, a navigation bar at the bottom, and logos for IIT Bombay and NPTEL.

So, this is a very simple example of what one shot timer or we call that watchdog timer. So, one example is, watchdog timer I have already told you. So, here, I have taken any critical, a critical function f, and now see, the start function with a watchdog WD stands for watchdog start then whatever the process you have to write you write, then, watch wd tickle. Watchdog tickle, now let us see how it works.

So, in the following figure a watchdog timer is set at the start. In the following figure a watchdog timer is set at the start or at the beginning for a critical function f, through what, through a method call or a function call named as wd start. So, in the following figure a watchdog timer is set for a critical function f at the initial or at the start through a call through a function called named as wd_start t1.

Now, actually this wd start has two parameters. One, is t1 another the exception handler. So now the wd start t1, it sets the watchdog timer. So, this function wd start it sets the watchdog timer to expire by the specified deadline.

What is the deadline it has set? I have already told you. It has two parameters t1 and this handler exception handler, so bar this handler exception handler so t1 is the specified time. Say, you

can say 20 milliseconds. So, this wd_start t1 this method it sets the watchdog timer to expire, how, by the specified deadline. Maybe, t1 units or t2 time units like that. From where? From the starting a bit task. I am repeating again, the method wd_start t1 it sets the watchdog timer to expire by the specified deadline. Here it is t1 time units from, where, from starting up the task.

Now there are two possibilities. The function f may be completed within the time into the time duration t1 or it may not be completed during the time duration t1. Now, let us see if it is not complete what will happen. If the critical function f does not complete even after t1 units have been expired t1 units of time have been elapsed, then what will happen, then the watchdog timer automatically it expires.

And what does it indicates? Indicating that, the task deadline is missed. The task t1 is missed but the function f1 is not completed. So, what will happen? What is the second parameter I have already told you? In wd_start t1, exception handler. So, since the function f has not been completed, but the time t1 is expired so it says that, the task deadline is missed and the corresponding exception handling procedure is initiated.

So, what is the exception handler that is initiated. Now, let us go to the other option. That during t1 time that function f is completed. So, if the task completes within deadline. So, if the task, if the critical task completes within the specified deadline, that is, within t1 time units, then what will happen, then the watchdog timer is reset using a wd_tickler call.

So, if the task is completed within the deadline, within the time unit t1, then the watchdog timer is reset. Because the critical tasks is already completed during this deadline, so the watchdog timer is reset using what, using the method or using the function wd_tickler() call. Using a function called named as the wd_tickler().

So, in this way, the watchdog timer it works. Very, very important in real-time systems. Please remember it.

(Refer Slide Time: 31:31)

CONCLUSION

- Investigated why RTOS has poor fine resolution system clocks.
- Examined various time services based on system clock.
- Examined different activities carried out by handler routine after clock interrupt.
- Learnt about hardware timestamp.
- Discussed about two different types of timers.

REFERENCES

1. Rajib Mall, Real-Time Systems: Theory and Practice, 1st Edition, 2007, Pearson Education
2. C. M. Krishna & K. G. Shin, Real-Time Systems, 2017, Tata McGraw Hill Education

So today I have discussed the reasons, why real-time systems, they have a poor resolution, not a finer resolution, a poor finer resolution system clocks. Maybe, the order of milliseconds or microseconds not of the order of nanosecond. We have examined the various time services based on the system clock. We will also have examined the different activities, which are carried out by the handler routine after the clock interrupt occurs.

We have also learned about the use of hardware timestamps in order to provide what higher clock resolution. It has also a limitation, I have told, that is the portability issues. We have also discussed about the different types of timers. We have seen two important timers here, one is the periodic timer and the other one is the aperiodic timer or the one shot timer.

The one of the what examples of one shot timer I have already told you, watchdog timer is one of the very good example of one shot timers. This is what we have discussed today. We have taken it from these two books. Thank you very much.