

**Real Time Systems**  
**Professor Durga Prasad Mohapatra**  
**Department of Computer Science and Engineering**  
**National Institute of Technology Rourkela**  
**Lecture 43**  
**Unix-Based Real-Time Operating Systems**

Good afternoon to all of you. In the last line, some of the previous lessons, we have discussed the Unix as a real-time system.

(Refer Slide Time: 00:48)

**CONCEPTS COVERED**

- Extensions to the Traditional Unix Kernel
- Host/Target Approach
- Preemption Point Approach
- Self-Host Systems
- Spin Locks

**KEYWORDS**

- Non-Preemptive Kernel
- Dynamic Priority Levels
- PSOS
- Micro-Kernel
- Spin Lock

Today we will discuss about some of the Unix-based real-time systems. That means some real-time operating systems, which are based on the concept of Unix. We will discuss about the extensions to the traditional Unix kernel, the host target approach, different what methods you will see are different, what approaches which are based on the traditional Unix to support real-

time operating systems such as host target approach, preemption point approach, self-host systems, spinlocks, etcetera. These are the keywords we will discuss.

(Refer Slide Time: 00:58)

**EXTENSIONS TO THE TRADITIONAL UNIX KERNEL**

- Several effort in this direction have been reported, including:
  - Implementing real-time task schedulers
  - Implementing finer granularity timers
  - Real-time file systems, etc.
- **Problems still Unsolved:**
  - **Non-Preemptive Kernel.**
  - **Dynamic priority level.**

The slide features a dark red header with the title in white. The content is a bulleted list with sub-bullets. A small video inset on the right shows a man speaking. At the bottom, there are navigation icons and logos for IIT Bombay and NPTEL.

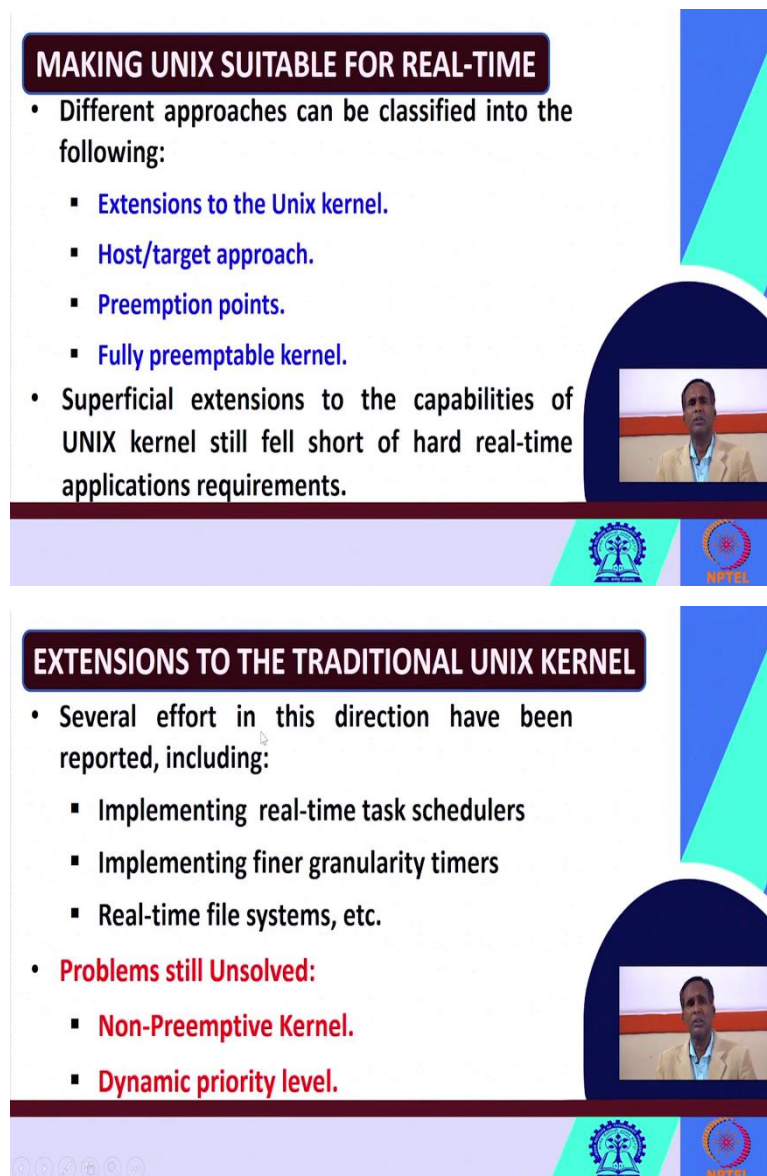
So now let us see how the traditional Unix kernel can be extended to support real-time systems. So, several efforts in this direction I have already been reported, such as implementing the real-time task schedulers, how the real-time task schedulers can be implemented.

Similarly, how to achieve a finer granularity timer or how to implement a finer granularity timer maybe of the resolution of the order of nanosecond also, and how to implement real-time file systems etcetera.

So, the several report in this direction I have already been reported, but still, but problems still unsolved; two important, two such important problems we have seen earlier such as the non-preemptive kernel and other is the dynamic priority kernel, we have already known that these two are the bottlenecks of the traditional Unix system,

It has a non-preemptive kernel and it uses dynamic priority level. So, these two drawbacks we have seen earlier for the traditional Unix operating systems. Now so still, these problems remain unsolved. So that is why the traditional Unix system it means it is very much difficult to use it as a real-time operating system.

(Refer Slide Time: 2:10)



The image shows two presentation slides. The top slide is titled "MAKING UNIX SUITABLE FOR REAL-TIME" and lists several approaches. The bottom slide is titled "EXTENSIONS TO THE TRADITIONAL UNIX KERNEL" and lists further extensions and unsolved problems. Both slides feature a video inset of a speaker and navigation icons at the bottom.

### MAKING UNIX SUITABLE FOR REAL-TIME

- Different approaches can be classified into the following:
  - Extensions to the Unix kernel.
  - Host/target approach.
  - Preemption points.
  - Fully preemptable kernel.
- Superficial extensions to the capabilities of UNIX kernel still fell short of hard real-time applications requirements.

### EXTENSIONS TO THE TRADITIONAL UNIX KERNEL

- Several effort in this direction have been reported, including:
  - Implementing real-time task schedulers
  - Implementing finer granularity timers
  - Real-time file systems, etc.
- **Problems still Unsolved:**
  - **Non-Preemptive Kernel.**
  - **Dynamic priority level.**

So, now let us see how Unix can be made suitable for real-time making Unix suitable for real-time systems. So, different approaches can be classified into the following. So, these are the possible some of the different approaches and they are based on these traditional Unix like extensions to the Unix kernel.

One approach called the host target approach, another approach we can see preemption points approach, another is fully preemptable kernel. So these approaches are based on this Unix concept, so we are trying to extend the Unix to or these are the Unix based approaches for the real-time systems. We will see the first one the extensions to the Unix kernel I have already told you.

So several reports in this direction I have already been mailed such as implementing real-time task scheduler, implementing finer granularity timers, implementing real-time file systems etcetera, but still the two important problems they lie, that is using because it uses non preemptive kernel and it uses dynamic priority level.

The superficial extensions to the capabilities of the Unix kernel still it fell short of hard real-time application requirements, they do not meet, these superficial extensions, they do not meet how to handle the hard real-time applications they do not meet the requirements of hard real-time application. So just by superficially extensive extending this traditional Unix kernel, it may not be sufficient, still those problems will lie.

(Refer Slide Time: 03:42)

**HOST/TARGET APPROACH**

- Develop application on Unix host:
  - Download to a dedicated OS running on the target.
  - The Unix host is connected to the target via a TCP/IP interface.
  - Examples of this category:
    - VRTX (Mentor graphics), VxWorks, and PSoS (Wind river systems), etc.

The slide features a video inset of a man in a light-colored shirt speaking. At the bottom, there are logos for IIT Bombay and NPTEL, along with navigation icons.

So we will see the next approach, that is the host target approach. So, this approach is used to develop applications on Unix host. So, this host target approach it is used for what? It is used to develop applications on Unix hosts. So there will be host where we can use Unix.

So host target approach can be used to develop applications on Unix host. So here what is the first step, download you download the application to a dedicated OS running on the target. So, you download the, you first develop the application on the Unix host.

Then you download it to a dedicated operating system running on the target, then this Unix host, it is connected to the target. So you see two things the Unix host on the Unix host and the target this Unix host is connected, it is attached to the target via interface, via an interface called as TCP IP interface.

So we can see different what operating systems based on this host targeted approach such as VRTX from mentor graphics, VxWorks and PSoS. They are both from Wind river systems and so on. These are the examples of the operating systems real-time operating systems which are based on this host target approach.

(Refer Slide Time: 04:57)

**HOST/TARGET APPROACH cont...**

Host

Target

**HOST/TARGET APPROACH**

- Develop application on Unix host:
  - Download to a dedicated OS running on the target.
  - The Unix host is connected to the target via a TCP/IP interface.
  - Examples of this category:
    - VRTX (Mentor graphics), VxWorks, and PSoS (Wind river systems), etc.

Let us see in the deeper so this is the host machine, where this Unix is there and as I have already told you first develop the application on the Unix host in this machine, then you try to download where you try to download to a dedicated OS running on the target. So this is the target. So here we download this application to a dedicated OS, which is this dedicated OS is running on the target and this host and target and these are communicated, they are connected through TCP IP.

(Refer Slide Time: 05:29)

**HOST/TARGET APPROACH cont...**

- The target operating system is intended to be compact and fast:
  - Suitable for embedded applications.
  - Need not support: Compilation, debugging, libraries, virtual memory, etc.
  - Task preemption time is of the order of 10 micro seconds:
    - Compared to 100 milliSec for fast Unix implementations,
    - 1 Sec for the traditional Unix system V

**HOST/TARGET APPROACH cont...**

Host

Target

The diagram shows a host computer (monitor and tower) connected to a target circuit board. A dotted line connects the host to the target. The host is labeled 'Host' and the target is labeled 'Target'.

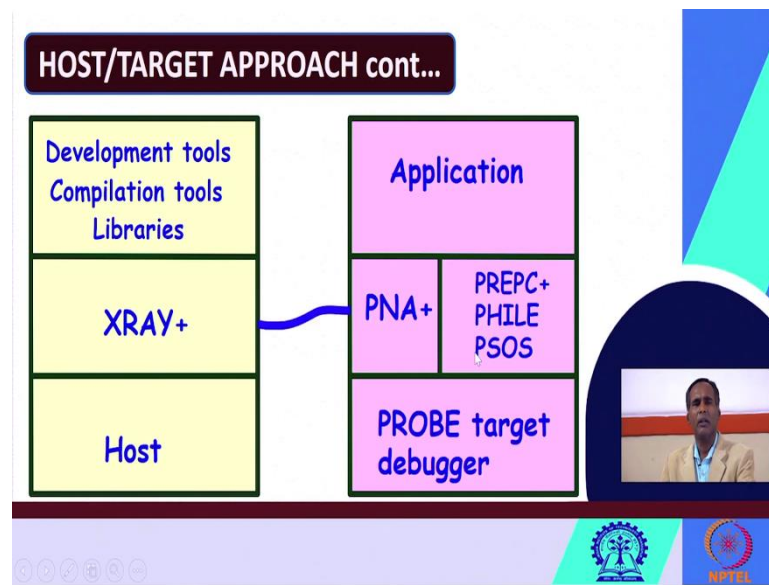
Now let us see more detail about the host target approach. Here the target operating system is intended to be compact and fast. The target operating system which one? This is the target operating system, this is the host operating system, the target operating system is intended to be compact and fast. So, this target operating system it should be very much compacted and it should be very much first.

So that is what I am saying the target operating system is intended to be compact and fast so that it can be made suitable for the embedded applications. So this target operating system it did not support the compilation, debugging, libraries, virtual memory, etcetera. They did not support this. So, they may be provided in the what host system, then this task preemption, the task preemption time in the order of the 10 microseconds.



So in this case, the task preemption time is of the order of 100 microseconds which is very much fast in compared to the what is very much fast in compared to the 100 millisecond for the fast Unix implementations or of the it is also faster in comparison to 1 second for the traditional Unix system V. So basically speaking that the task preemption time in this what target operating system is of the order of 10 microseconds.

(Refer Slide Time: 06:50)



This host target approach looks like this, this left-hand side is this unique host operating system, right-hand side is the target operating system. So in this host operating system, you can use with the different development tools, different compilation tools, different libraries, and XRAY plus, so this is the host system and on the target, we will use this application, and here some important components we see like PNA plus and this PREPC plus PHILE and PSOS.

So, what is PNA Plus? I will just say and what is PSOS I will just say, so this host and so in the target we are using a debugger called this a PROBE target debugger. On the target or on the target we are using a debugger called as a probe-target debugger, the host and the target they are connected through TCP IP. Now let us see this particular the terms like XRAY plus PNA plus and the PSOS. What do they represent, let us quickly see.

(Refer Slide Time: 07:54)

**HOST/TARGET APPROACH cont...**

- The host provides development tools:
  - Cross-compiler.
  - Libraries.
  - Editor and other development tools.
- XRAY+ is the source level cross debugger/ analyser.
- PNA+ is the network manager.

**HOST/TARGET APPROACH cont...**

Development tools Compilation tools Libraries	Application
XRAY+	PNA+    PREPC+ PHILE PSOS
Host	PROBE target debugger

So, I have already told you the host provides the development tools. This is the host operating system This provides the development tools, compilation tools and libraries, this I have already told you. The host provides the development tools such as cross compiler, different libraries, editor and other development tools.

So these are provided in the host operating system, you can see that so another element in here called as XRAY plus, what is the XRAY plus? So XRAY plus is the source level cross debugger analyzer. So, XRAY plus is the source level what cross debuggers analyzer and because you see here you are using a target debugger in the on the target you are using a debugger called as a probe-target debugger.



So on the host operating system or in the host operating system, we are using XRAY plus as which is acting at the source level cross debugger and analyzer. What is the PNA plus which is present on this target? So PNA plus is the network manager because they are connected through TCP IP, the host and the target, they are connected to TCP IP. So somebody should manage this networking activity. So PNA plus is working is acting as the Network Manager.

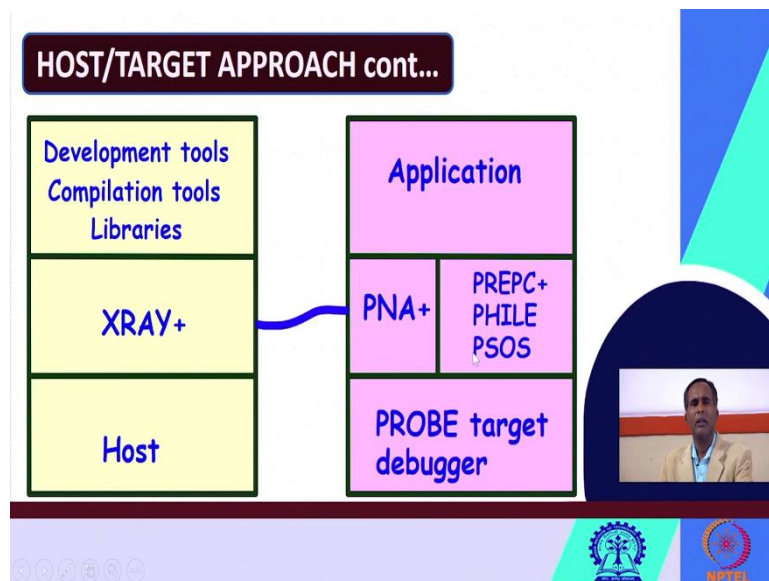
(Refer Slide Time: 09:06)

**HOST/TARGET APPROACH cont...**

- The application is developed:
  - Using editor, compiler, debugger, and other tools available on the host.
- Once the application has been fully tested and debugged:
  - **It is fused in ROM.**

**HOST/TARGET APPROACH**

- Develop application on Unix host:
  - **Download to a dedicated OS running on the target.**
  - The Unix host is connected to the target via a TCP/IP interface.
  - Examples of this category:
    - VRTX (Mentor graphics), VxWorks, and PSoS (Wind river systems), etc.

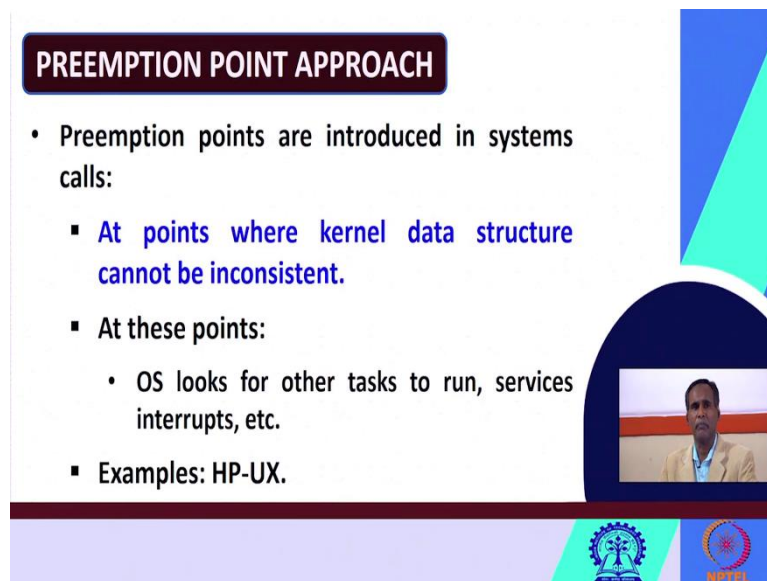


Now let us come to the target side. So once the so the application is developed first where, I have already told you the application is developed on the Unix host, and then that will be downloaded to the dedicated OS running on the target. So that is what I am saying here the application is developed using what using the editor compiler debugger and other tools available where? Available on the host I have already told you.

On the host these development tools, compilation tools and libraries are there using this you can develop the application. The application is developed using the editor, compiler, debugger, and the other tools available on the host. So once the application has been fully tested and debugged then what it is done? It is fused ROM.

So, once the application if they have been fully tested and then debugged, then it is fused in ROM, so where ROM is there, ROM is in the target machine in the target operating system. So then so once this application has been fully tested and debugged then it is fused in ROM.

(Refer Slide Time: 10:03)



**PREEMPTION POINT APPROACH**

- Preemption points are introduced in systems calls:
  - At points where kernel data structure cannot be inconsistent.
  - At these points:
    - OS looks for other tasks to run, services interrupts, etc.
  - Examples: HP-UX.

The slide features a dark blue header with the title in white. The main content is on a white background with a dark blue border. A video inset shows a man in a light blue shirt. Logos for IIT Bombay and NPTEL are at the bottom right.

Next, we will go to the next approach that is the preemption point approach. So preemption points are introduced in system calls. So, where the preemption points are introduced, the preemption points are introduced in the system calls or the points where kernel data structure cannot be inconsistent. So, we have to look at the points where the kernel data structure cannot be inconsistent.

So, there we will introduce what? There we will introduce the preemption points. So, preemption points, at preemption points what will happen we can make preemption or we can preempt the running task and we can allow the waiting higher priority tasks to be run. So, that is why this approach its name is known as preemption point approach.

Here preemption points are introduced in the system calls, where? At the points, where the kernel data structure cannot be inconsistent. So, at these points what will happen? At these preemption points the operating system it will search for the other tasks to run and search for the services interrupts etcetera.

So, normally the tasks which is having high priority, now it will come and it will preempt the current running tasks which may have lower priority and then this high priority task may run. So, examples of operating systems based on the preemption point approach is HP-UX.

(Refer Slide Time: 11:27)

**PREEMPTION POINT APPROACH cont...**

- Vendors introduced preemption points in system routines.
- Preemption points in the execution of a system routine are instants at which data structure is consistent.
- At this point, kernel can safely be preempted:
  - Waiting higher priority real-time tasks to run without corrupting any kernel data structures.

**PREEMPTION POINT APPROACH**

- Preemption points are introduced in systems calls:
  - At points where kernel data structure cannot be inconsistent.
  - At these points:
    - OS looks for other tasks to run, services interrupts, etc.
  - Examples: HP-UX.

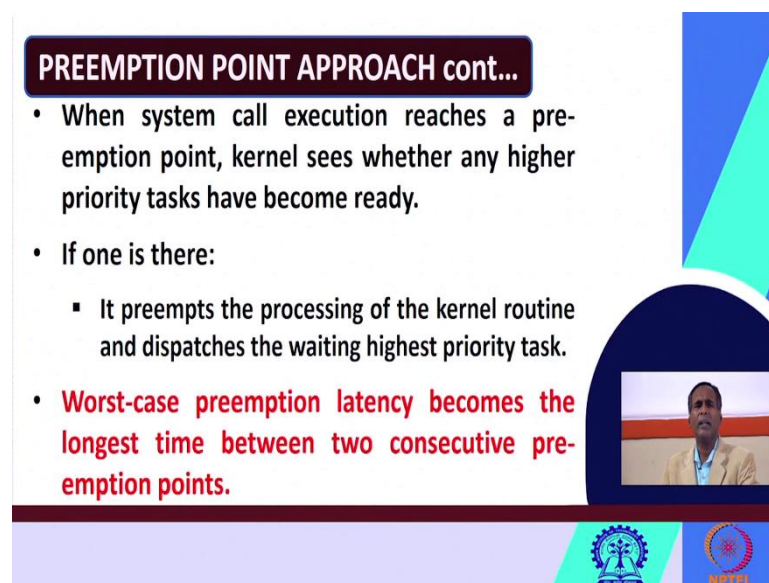
The vendors, they introduced preemption points in system routine. So, who introduced these preemption points? So, I have already told you that preemption points are introduced in the system calls, so preemption points are introduced in system calls are the points where the kernel data structure cannot be inconsistent.

Now, who introduces the preemption points, the vendors they introduce the preemption points in the system routines, the preemption points in the execution of a system routine are instance at which data structure is consistent. So, what do you mean by these preemption points? So, the preemption points in the execution of a system routine these are all what? These are instance of time or instants of points; these are instance at which the data structure is consistent.

At this point, the kernel can safely be preempted, nothing will happen, no disturbance or no effect will be there. So at these preemption points the kernel can safely be preempted. So that what will happen? The waiting higher priority real-time tasks that can run without corrupting any kernel data structure, because these are the preemption points where the kernel can be safely preempted, the data structure is consistent, the data structure will not be corrupted.

So here at this point kernel can safely preempted, so that the waiting higher priority real-time tasks, they can run without corrupting any kernel data structure. This is something about the preemption point approach example we have seen HP-UX.

(Refer Slide Time: 12:56)



**PREEMPTION POINT APPROACH cont...**

- When system call execution reaches a preemption point, kernel sees whether any higher priority tasks have become ready.
- If one is there:
  - It preempts the processing of the kernel routine and dispatches the waiting highest priority task.
- **Worst-case preemption latency becomes the longest time between two consecutive preemption points.**

The slide features a video inset of a man in a tan jacket speaking. At the bottom, there are logos for IIT Bombay and NPTEL.

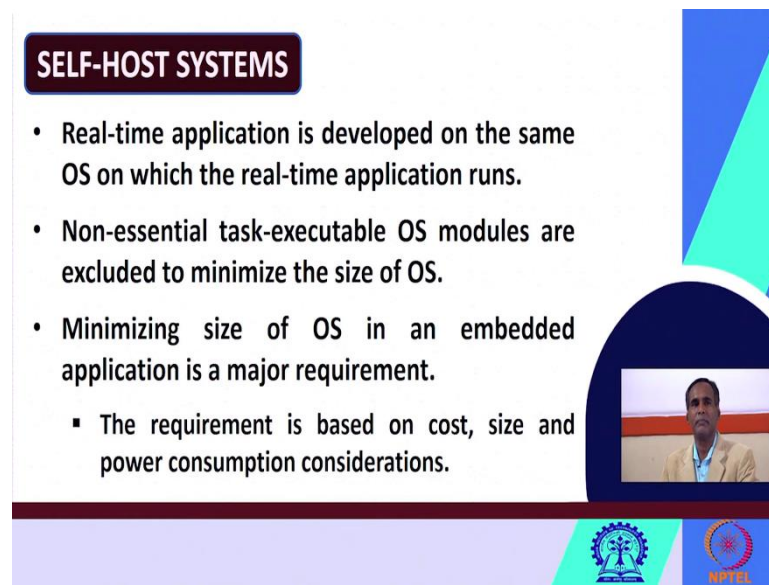
When the system call execution reaches a preemption point, then what will happen? The kernel will check, it checks, it receives whether any higher priority task have become ready very simple. So when the system call execution it reaches at a preemption point, then what the kernel will try to do?

The kernel will try to see; the kernel will try to check whether any higher priority tasks it has become ready. So then if it finds any higher priority tasks, in comparison to which is running, if any such high priority task is ready, then what it will do, if when the kernel will find yes, such a task is ready.

If one is there, then the kernel preempts the processing of the kernel routine, and then it will dispatch the waiting highest priority tasks, it will preempt the lower priority tasks, and it will dispatch the waiting highest priority tasks to run.

In case of the preemption point approach. The worst-case preemption latency becomes the longest time between two consecutive preemption points. So the worst-case preemption latency, what it will happen or what will be the longest time here you can see? The worst-case preemption latency, it becomes the longest time between the two consecutive preemption points.

(Refer Slide Time: 14:14)



**SELF-HOST SYSTEMS**

- Real-time application is developed on the same OS on which the real-time application runs.
- Non-essential task-executable OS modules are excluded to minimize the size of OS.
- Minimizing size of OS in an embedded application is a major requirement.
  - The requirement is based on cost, size and power consumption considerations.

The slide features a dark blue header with the title 'SELF-HOST SYSTEMS' in white. The main content is a list of three bullet points. The first two are in black text, and the third is in bold black text with a sub-bullet point. A small video inset shows a man speaking. The slide has a decorative background with blue and green geometric shapes and logos for IIT Bombay and NPTEL at the bottom.

Now we will see the next approach that is the self-host system approach. So in this approach, the real-time application is developed where? On the same operating system that is why the name is self-host systems. So in this approach, the real-time application is developed on the same operating system on which the real-time application runs. The nonessential tasks executable OS modules are excluded to minimize the size of the OS.

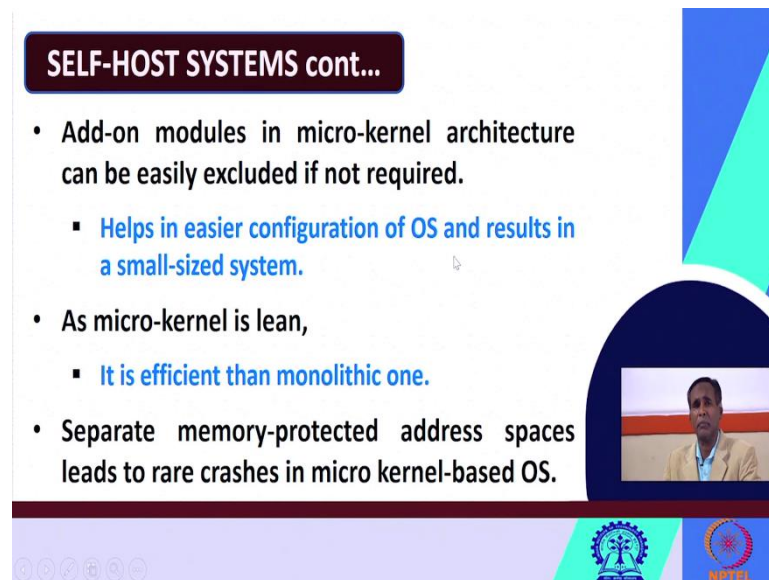
So in order to minimize the size of the OS, the modules, which are not so important, which are not so essential, these nonessential tasks executable operating system modules, they are excluded, they are what, they are not taken into account they are excluded to minimize the size of OS. They are not included in order to minimize the size of OS, minimizing the size of the OS, how does it will affect?

So, minimizing the size of the OS in an embedded application is a major requirement in case of what, these major requirements taken into consideration the size. So, minimize So, minimizing size of an operating system in an embedded application is a major requirement. And this requirement is based on the cost size and power consumption considerations.



So, while we will try to make the size of this operating system very less you will try to make it minimized, here the requirement is based on what the cost size and power consumption considerations take into account the cost size and power consumption considerations you can minimize the size of the operating system in an embedded application.

(Refer Slide Time: 15:50)



**SELF-HOST SYSTEMS cont...**

- Add-on modules in micro-kernel architecture can be easily excluded if not required.
  - Helps in easier configuration of OS and results in a small-sized system.
- As micro-kernel is lean,
  - It is efficient than monolithic one.
- Separate memory-protected address spaces leads to rare crashes in micro kernel-based OS.

The slide features a dark blue header with the title 'SELF-HOST SYSTEMS cont...' in white. The main content is a list of bullet points. A small video inset shows a man in a tan jacket. The bottom of the slide has a purple bar with navigation icons and logos for IIT Bombay and NPTEL.

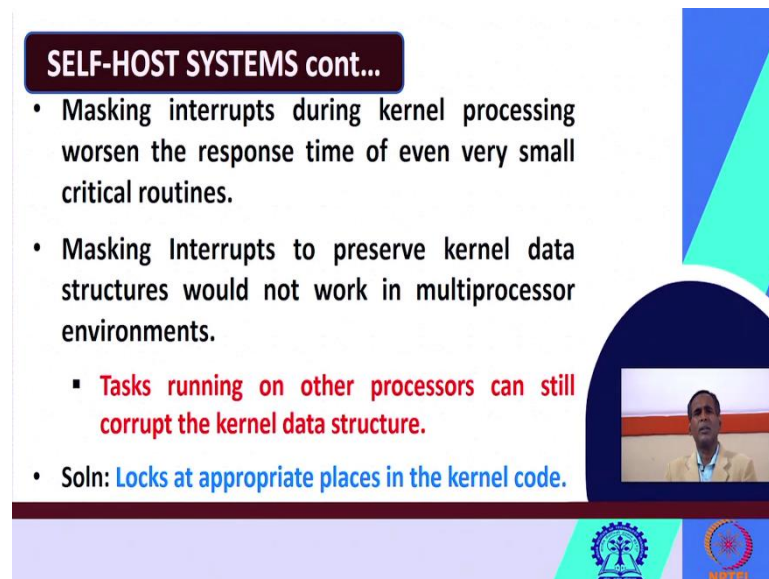
So, here are the Add-on modules in microkernel architecture can be easily excluded if not required. So, if I have already told you microkernel architecture in some of the previous classes, so, in the microkernel architecture, there are several add-on modules. So, if you find some of the add-on modules, they are not so much required they are not important they are not relevant, if we can simply exclude them, please do not include them.

So, the add-on modules in microkernel architecture can be easily excluded if not required, it helps in easier configuration of operating system and results in a small size system. So, why we will not add all the what add-on modules. So, if they are not required simply do not include them you please exclude them, then what will happen? How does it will help?

So, excluding those add-on modules will help in your configuration of the operating system and it will result in what a small-sized system so, the operating system that will get it will up or yes the system that will get it will be of reasonable size or small size. As you know that are the microkernel is lean, it is efficient than monolithic one. You have already known this microkernel is efficient than monolithic one. So, as a microkernel is lean, it is efficient than monolithic one.

Separate memory-protected address, it separates the memory-protected address spaces or separate memory-protected address spaces leads to rare crashes in a microkernel-based operating system. So since in this microkernel-based operating system, these memory-protected address spaces are separate This separate memory-protected address spaces, it leads to what it leads to what very rare or very less number of crashes in microkernel based operating systems.

(Refer Slide Time: 17:33)



**SELF-HOST SYSTEMS cont...**

- Masking interrupts during kernel processing worsen the response time of even very small critical routines.
- Masking Interrupts to preserve kernel data structures would not work in multiprocessor environments.
  - Tasks running on other processors can still corrupt the kernel data structure.
- Soln: Locks at appropriate places in the kernel code.

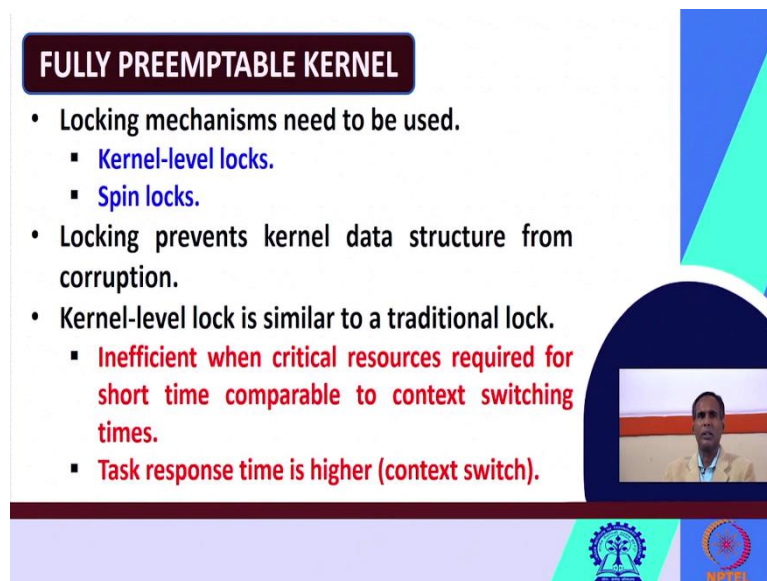
The slide features a dark blue header with the title 'SELF-HOST SYSTEMS cont...' in white. The main content is on a light blue background with a dark blue sidebar on the right containing a video thumbnail of a man speaking. At the bottom, there are logos for IIT Bombay and NPTEL.

So now, I hope we have already known interrupts and masking the interrupts so now let us say this masking interrupts how it will affect. So masking the interrupts during the kernel processing, it worsen the response time of even very small critical routines. By masking the interrupts during the kernel processing, it worsen the response time for even very small critical routines.

So masking interrupts to preserve kernel data structures would not work in multiprocessor environments. So this masking of the interrupts to preserve the kernel data structure may not work in multiprocessor environments. So tasks running on other processors can still corrupt the kernel data structure. Why it will not work in the multiprocessor environments?

Because it means the masking why this masking interrupts or may not to work in multiprocessor environments because here the task running on other processors, they can still corrupt the kernel data structure that is why masking interrupts may not work in the multiprocessor environments. So what will be the solution, the solution is that you can use lock, locks at appropriate places in the kernel code. You may use locks different locks at the appropriate places in the kernel code.

(Refer Slide Time: 18:47)



**FULLY PREEMPTABLE KERNEL**

- Locking mechanisms need to be used.
  - Kernel-level locks.
  - Spin locks.
- Locking prevents kernel data structure from corruption.
- Kernel-level lock is similar to a traditional lock.
  - Inefficient when critical resources required for short time comparable to context switching times.
  - Task response time is higher (context switch).

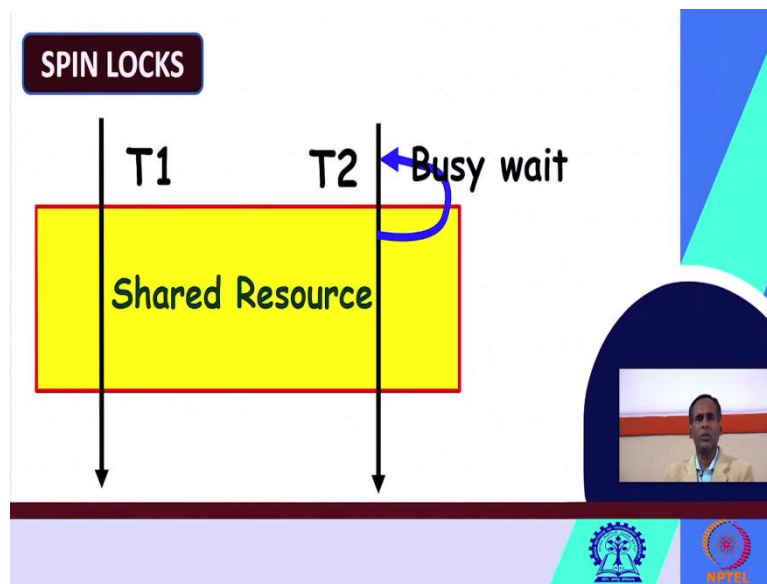
The slide features a dark blue header with the title 'FULLY PREEMPTABLE KERNEL' in white. Below the title is a list of bullet points. The first bullet point has two sub-bullets. The second and third bullet points are single lines. The fourth bullet point has two sub-bullets, one in red text. To the right of the text is a video inset showing a man in a light-colored shirt. At the bottom of the slide are two logos: the Indian Institute of Technology (IIT) logo on the left and the NPTEL logo on the right.

So let us say I will use this use of the kernel locks in the next model, the next model is or the next approach is fully preemptable kernel. So here we have to use different locking mechanisms such as kernel level locks and the spinlocks, etcetera. So, why you will use this locking because the locking mechanism it prevents kernel data structure from corruption. You can use the locking mechanism, then it may what, prevent the kernel data structure from getting corrupted. The kernel-level lock is similar to traditional lock.

Now let us see why we have introduced to two new terms called as kernel-level lock and spinlock. So how does this kernel-level lock differ from the traditional lock? So kernel-level lock is almost similar to a traditional lock but few differences. So it is inefficient when critical resources required for short time comparable to context switching times.

So this kernel lock is very much similar to the traditional lock but it is inefficient when the critical resources are required for a very short time comparable to the context switching and time. So the task response time is higher. So in case of this, what I am discussing about the difference between kernel lock and the traditional lock, as I have already told you that kernel lock is similar to traditional lock, but here the task response time is higher.

(Refer Slide Time: 20:10)



### SPIN LOCKS cont...

- A shared resource is required by both tasks  $T_1$  and  $T_2$  for very short times.
  - This resource is protected by a spin lock.
- Suppose task  $T_1$  has acquired spin lock guarding the resource.
- Task  $T_2$  requests the resource.
- Since task  $T_1$  has locked it,  $T_2$  cannot get access to the resource.
  - It just busy waits and does not block and suffer context switch.

### SELF-HOST SYSTEMS cont...

- Masking interrupts during kernel processing worsen the response time of even very small critical routines.
- Masking Interrupts to preserve kernel data structures would not work in multiprocessor environments.
  - **Tasks running on other processors can still corrupt the kernel data structure.**
- Soln: **Locks at appropriate places in the kernel code.**

So, now let us see about the second lock that is the spinlock. So, here I have shown the example of a spinlock. Let me explain it. So suppose there is a shared resource and this is needed by two tasks, T1 and T2. And for very short times, very short duration of times. So, T1 and T2 they require the shared resource for very short periods of times. This resource is protected by a spinlock.

So, what you can do, one the solution I have already told you, you can use different locks. So, one such lock is spinlock. So here So, since T1 and T2 both require the same shared resource, this resource is protected by a spinlock. Now, let us see how does it works suppose now task T1 has acquired spinlock guarding the resource.

So, now, T1 first initially it has a get access to this spinlock. Suppose task T1 has acquired this spinlock guarding the resource. Now, then what will happen? So T2 has to wait now task T2 after some time it requires the resource. So task T2 it requires the resource.

Now since task T1 has locked the resource then obviously T2 cannot get access to the resource. Then what it will do so it is just busy waits and does not block and suffer context switch. So T1 is having is using this what spinlock so instead of what getting blocked T2 what T2 does, since task T1 has already locked it. So T2 of course cannot get access to the resource.

What does it do? It just busy waits. So this T2 it is just busy waits and it does not block, it is not blocked, it does not block and does not suffer a context switch. So, T2 it does not what suffer, suffer by any context switch and the T1 is holding a resource and it acquires the spinlock. So now T2 it will just wait for some time because the time duration here you have known is very small period of time they require the resources.

So T2 will not be blocked. So T2 will be in busy wait. So T2 cannot get access to the resource it just a busy waits and it does not block and suffer the context and suffer from context switch.

(Refer Slide Time: 22:24)

**SPIN LOCKS cont...**

- $T_2$  gets the resource as soon as  $T_1$  relinquishes the resource.
- Spin locks are normally implemented:
  - Using the cache coherency protocol that each processor loops on a local copy of the lock variable.
- Spin locks on a uniprocessor get compiled in as calls to “cli” and “sti” instructions.

**SPIN LOCKS**

T1 T2 Busy wait

Shared Resource

Then what will happen when  $T_2$  will get the resource? So as soon as  $T_1$  relinquishes or releases the resource then  $T_2$  gets the resource. So, normally here with this solution we have got this by using a spinlock. So first  $T_1$  is granted access or it acquires the spinlock instead of blocking  $T_2$  it busy waits and when  $T_1$  releases or relinquishes the spinlock then  $T_2$  gets access to this spinlock and then it gets access to the shared resource.

Now, how the spinlocks can be implemented. So spinlocks are normally implemented using cache coherency protocol. I hope in some of the earlier semesters you might have heard about the cache coherency protocol. So the spinlocks they can be implemented by using the cache coherency protocol that each processor loops on a local copy of the lock available.



So each processor can get a local copy of the lock variable. So this is called as what this method where this cache coherency protocol may work. So the spinlocks they are normally implemented using the cache coherency protocol, such that each processor, it loops on a local copy of the lock variable. So it will take the uniprocessor systems, the spinlocks on a uniprocessor, they get compiled in as what? They get compiled in as calls to instructions such as CLI and STI.

So, the spinlocks on the uniprocessor get compiled in as the calls to the CLI instruction and STI instructions. So, this is how spinlock can be implemented more details about how to implement spinlocks are there in your book please have a look into that.

(Refer Slide Time: 24:05)

**RT PRIORITIES IN SELF HOST SYSTEMS**

- Three available priority levels:
  - **Idle or Non-Migrating (255)** – It is the lowest priority level. The task runs when there are no other tasks to run. These priorities are static and are not recomputed periodically.
  - **Dynamic Priority (128-254)**: Recomputed periodically. The tasks at this level operate at priorities higher than idle priority, but lower than RT priorities.

**RT PRIORITIES IN SELF HOST SYSTEMS**

- **Real-Time Priority (0-127)**: These priorities are static and are not recomputed during run time. Hard RT tasks operate at these levels. Tasks having RT priorities get precedence over tasks with dynamic priority level.

Now let us quickly look at the real-time priorities in self-host systems. We are discussing about the self-host systems. So what are the priorities in self-host systems? In self-host systems, there are three priority levels are available; one is the idle or nonpriority, non-migrating priority level, then the dynamic priority level, then the real-time priority level.

First, let us see about this, so in this self-host system, there are 256 priority levels available three available priority levels, and how many priorities 256 priorities you can see. So they are divided into three levels. So first one is the idle or non-migrating, it is the lowest priority level and the task, the task runs when there are no other tasks to run.

So a task at this level may run when there are no other tasks to run at this level, this please remember these priorities are static and are not recomputed periodically. So, these idle or non-migrating what priority is static and they are not re-computed periodically. So, the next level is called as dynamic priority level, you know the dynamic, why they are dynamic priority level? Because they are recomputed periodically, the tasks at this priority level operate at priorities higher than the higher priority.

So, this dynamic priority if the any task running in this priority level they get more priority they get, they operate at priority higher than this idle priority, but they are lower than the RT priorities. That is the real-time priorities. So, the tasks at this level they operate at priorities higher than this idle or non-migrating priorities, but lower than the real-time priority.

Now let us see what is this real-time priority. So, real-time priority means these priorities which are static, because we know this is a requirement for real-time operating systems. So, these priorities are static in nature and are not recomputed during runtime. So, these priorities they are not recomputed during the runtime.

All the hard real-time tasks, they operate are these priority levels. So, all the hard real-time tasks they operate are these real priority levels, why? Because the hard real-time tasks, for the hard real-time tasks the deadline is very much important, we should not miss the deadline. So, there must be given topmost priority. So hard real-time tasks, they operate at these levels.

So tasks, which are having real-time priorities, they get precedence over the tasks with the dynamic priority level. So, the tasks which are so, the tasks are having the real-time priority, they get a precedence over what they get precedence over the what over the dynamic priorities, dynamic priorities, and the idle priorities.

So, the tasks are having real-time priorities get precedence over the tasks with the dynamic priority level. So, I have already told you that there are 256 priorities. So, the topmost priority is given to real-time priority level. So, it is number is 128 priorities that means 0 to 127 these priorities are given to real-time priority.

They are the highest priority topmost priority next priority is given to this dynamic priority level, here the priorities are what assigned from the levels 128 to 254. So, another 127 priorities are assigned to dynamic priority level.

And the last priorities what only one that is 255. So, this priority is given to idle or non-migrating priority levels, this is how the real-time priorities are assigned in self-host systems. So, in this way, we have discussed the different approaches these different Unix-based approaches for real-time operating systems.

(Refer Slide Time: 28:04)

**CONCLUSION**

- Discussed about the extensions to the traditional UNIX kernel.
- Learnt about the host-target approach.
- Discussed about preemption point approach.
- Highlighted about the benefits of self-host systems.
- Learnt about handling shortcomings of non-preemptive kernel using spin-locks.

So, today we have discussed about the extensions to the traditional Unix kernel and we have still seen that the problems such as non-preemptable kernel, dynamic priority levels are there. We have also learned about the different approaches, different Unix-based approaches for the real-time systems such as host target approach, preemption point approach, and self-host systems approach.

We have also discussed about the benefits of self-host systems; we have also learned about the handling, the shortcomings of the non-preemptive kernel using spinlocks. I already told you how by using spinlocks you can overcome some of the shortcomings of the non-primitive kernel.

(Refer Slide Time: 28:51)



**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

The slide features a dark red header with the word 'REFERENCES' in white. Below the header, two references are listed in black text. On the right side, there is a video inset showing a man in a light-colored shirt speaking. The slide is decorated with a blue and green geometric pattern on the right and logos for IIT Bombay and NPTEL at the bottom.

So we have again taken these things from these two books. That is something about the different Unix best approaches per the real-time systems. Thank you very much.