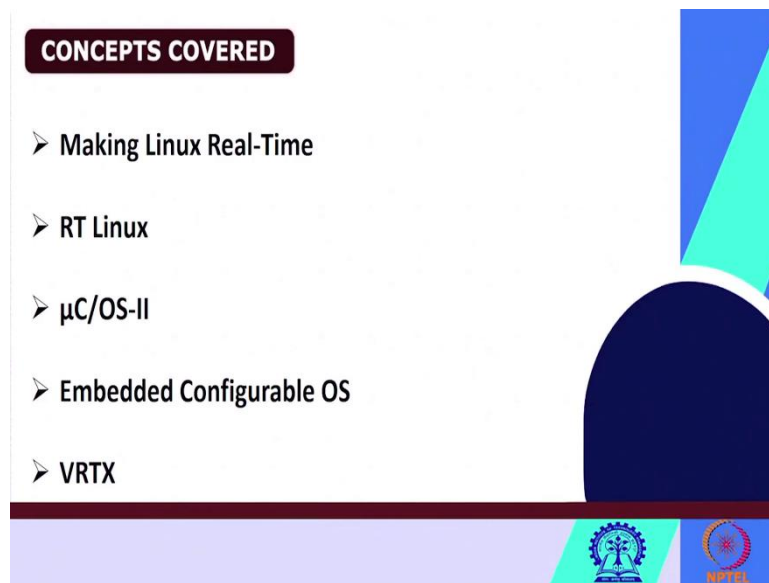


Real Time Systems
Professor Durga Prasad Mohapatra
Department of Computer Science and Engineering
National Institute of Technology Rourkela
Lecture 44

A survey of some contemporary Real-Time Operating Systems

Good afternoon to all of you. Today we will discuss a survey of some available commercial contemporary real-time operating system. There are so many commercial real-time systems operating systems are available. Today we will discuss few of them.

(Refer Slide Time: 00:37)



We will discuss first the we will see how this Linux operating system can be made in real-time. Then we will see about these details of RT-Linux, another real-time operating system you will see Micro C operating system 2, then embedded configurable operating system and finally, we will see about a type of real-time operating system called VRTX.

(Refer Slide Time: 00:59)

KEYWORDS

- Preemption Improvement
- Interrupt Abstraction
- Hardware Abstraction Layer
- Programmable Interval Timer
- Embedded Configurable OS

So we will see some of these keywords like Preemption improvement, Interrupt abstraction, HAL or hardware abstraction layer, Programmable interval timer, embedded configurable operating system, VRTX, etcetera.

(Refer Slide Time: 01:15)

MAKING LINUX REAL-TIME

- Two main approaches:
 - **Preemption improvement:** Recent patches from Ingo include a (large) number of technologies for improving preemption and debugging preemption issues with the Linux kernel.
 - **Interrupt abstraction:** Here, the real-time kernel takes over interrupt handling from Linux.
 - The Linux kernel “thinks” it is disabling interrupts, but it really is not.

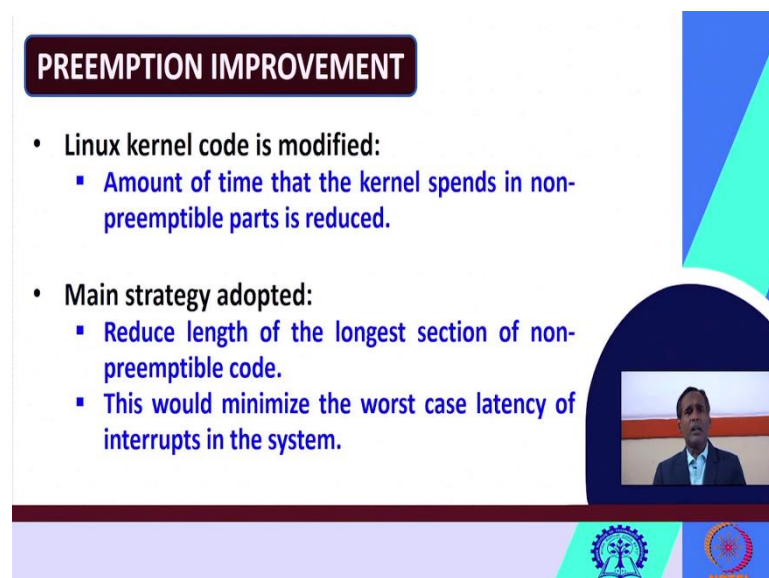
Let us start with the Linux. In the previous lectures we have already seen Unix and windows and about their suitability for real-time applications. This we have already discussed in the previous classes, today we will talk about these fundamentals of Linux and how it can be made real-time. A special person will see that RT-Linux which is what can be used for real-time applications.

So, let us see how we can make Linux real-time or how making Linux real time will be feasible. There are two main approaches available for making Linux real-time one is this Preemption improvement, another is the Interrupt abstraction. You have already known that Linux is a free software and but it has several advantages, you know, it is normally a general purpose operating system, but how we can make it real-time oriented let us see.

For these two available approaches are their number one is Preemption improvement. In this approach, the recent patches Ingo, Ingo you can consider as a company. So, recent patches from Ingo it includes a large number of technologies for improving the preemption and debugging of preemption issues with the Linux kernel. So, in this preemption improvement several recent patches, they are considered.

These recent patches are taken from Ingo they include a large number of technologies for improving the preemption and debugging issues with the Linux kernel. So, in the second approach that is the interruption abstraction; here the real-time kernel it takes over the interrupt handling from the Linux. So, or the Linux kernel you can say so, in this approach interrupt abstraction, the real-time kernel takes over interrupt handling from the Linux kernel. So here, the Linux kernel thinks that it is disabling the interrupts but actually it is not really like that it is not, it is really it is not.

(Refer Slide Time: 03:23)



PREEMPTION IMPROVEMENT

- **Linux kernel code is modified:**
 - Amount of time that the kernel spends in non-preemptible parts is reduced.
- **Main strategy adopted:**
 - Reduce length of the longest section of non-preemptible code.
 - This would minimize the worst case latency of interrupts in the system.

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

So now, let us see the first approach in detail that the preemption improvement. So here the Linux code, it is modified the available the existing Linux kernel code it is modified the amount of time that the kernel spends in the non-preemptible parts it is substantially reduced. Here, what the main strategy adopted?

So, in this approach, we reduce the length of the longest section of the non-preemptible code. And why we do this, this would minimize the worst-case latency of the interrupts in the system. That is why we have to reduce the length of the longest section of the non-preemptible code. This is the main strategy adopted in case of the preemption improvement.

(Refer Slide Time: 04:05)

INTERRUPT ABSTRACTION

- In stead of instrumenting Linux kernel to increase preemptibility:
 - The entire kernel is made preemptible.
 - A separate layer intercepts and manages the hardware interrupts.
 - Hardware abstraction layer has complete control over the hardware interrupts.
 - Under the real-time scheduler:
 - Linux kernel runs as a low priority task.

PREEMPTION IMPROVEMENT

- Linux kernel code is modified:
 - Amount of time that the kernel spends in non-preemptible parts is reduced.
- Main strategy adopted:
 - Reduce length of the longest section of non-preemptible code.
 - This would minimize the worst case latency of interrupts in the system.

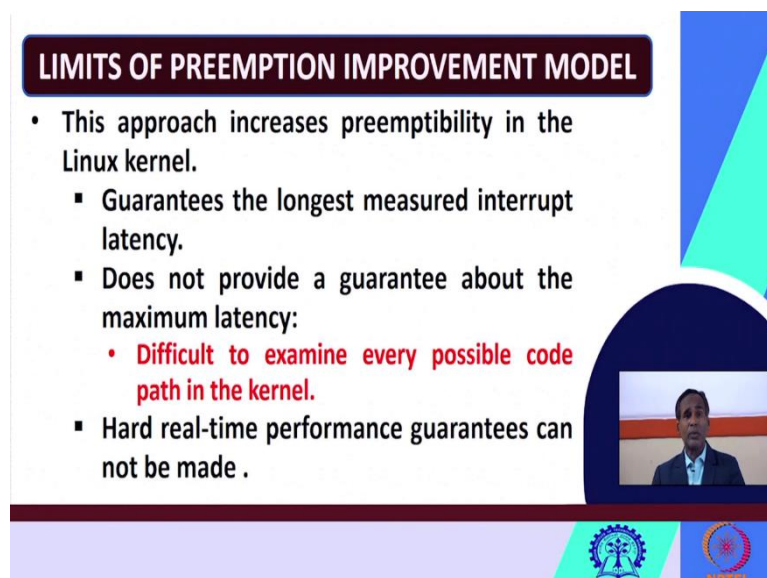
Then let us see what is happening in interrupt abstraction. So, in case of this interrupt abstraction, instead of instrumenting the Linux kernel, what we are instrumenting? In Preemption improvement, I have already told you in preemption improvement. the Linux code is modified; the Linux code is instrumented but here we do not instrument the Linux kernel, we do not what modify the Linux kernel.

So instead of instrumenting, the Linux kernel to increase the preemptibility, what we do here? Here the entire kernel is made preemptible we are in case of interrupt abstraction the whole kernel is made preemptible. So here is separate layer called as the hardware abstraction layer or HAL. So, in this interrupt abstraction, a separate layer called the hardware abstraction layer is used this separate layer, maybe call it the hardware abstraction layer it intercepts and manages all the hardware interrupts.

So please mark the difference between the interrupt abstraction and the preemption improvement. In case of preemption improvement, the Linux kernel code is modified or it is what instrumented but in case of interrupt abstraction, the entire kernel is made preemptible and a separate layer is inserted which we call as abstraction, hardware abstraction layer this layer it intercepts and manages the hardware interrupts.

So, this hardware abstraction layer it has the complete control over the hardware interrupts So, who does this? So, this hardware abstraction layer it has the complete control over the hardware interrupts. So, under the real-time scheduler, so, in case of this interrupt abstraction under the real-time scheduler, the Linux kernel runs as a low priority task. So, there are two kernels you are observing. One is this Linux kernel, maybe under the real-time kernel. So, here under the real-time scheduler, the Linux kernel runs as a low priority task.

(Refer Slide Time: 05:59)



LIMITS OF PREEMPTION IMPROVEMENT MODEL

- This approach increases preemptibility in the Linux kernel.
 - Guarantees the longest measured interrupt latency.
 - Does not provide a guarantee about the maximum latency:
 - **Difficult to examine every possible code path in the kernel.**
 - Hard real-time performance guarantees can not be made .

The slide features a dark blue header with the title in white. The main content is a bulleted list with sub-bullets. A video inset in the bottom right shows a man in a suit 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.

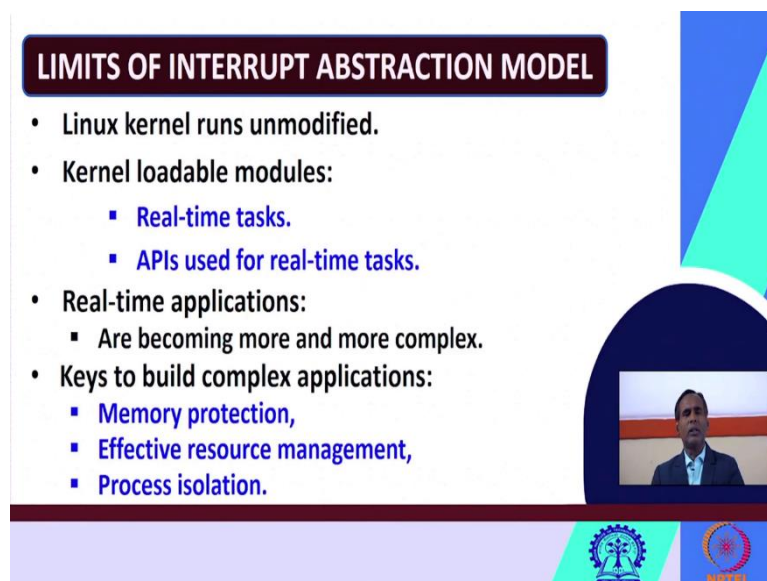
Let us quickly look at the limitations of the preemption implement model and the limitations of this interruption abstraction model or the interrupt abstraction model. First, we will see the limitations of the preemption improvement model. So, this approach it increases preemptibility in the Linux kernel. So, we can say that this preemption improvement model, this approach it

increases the preemptability in the Linux kernel, it is guaranteed the longest measured interrupt latency.

The preemption improvement model it gives guarantee that the longest time measured the interrupt latency is there. It guarantees the longest measured interrupt latency, but the limitation is that it does not provide a guarantee about the maximum latency. It does not give guarantee, it does not provide guarantee about the maximum latency why, because it is difficult to examine each and every possible code paths in the kernel.

It is very much difficult to examine each and every possible code path in the kernel that is why it does not provide a guarantee about the maximum latency. So, the hard real-time performance guarantees cannot be made, also in this preemption improvement model another drawback is that the hard real-time performance guarantees it cannot be made, it cannot be insured in case of the preemption implement model.

(Refer Slide Time: 07:12)



LIMITS OF INTERRUPT ABSTRACTION MODEL

- Linux kernel runs unmodified.
- Kernel loadable modules:
 - Real-time tasks.
 - APIs used for real-time tasks.
- Real-time applications:
 - Are becoming more and more complex.
- Keys to build complex applications:
 - Memory protection,
 - Effective resource management,
 - Process isolation.

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

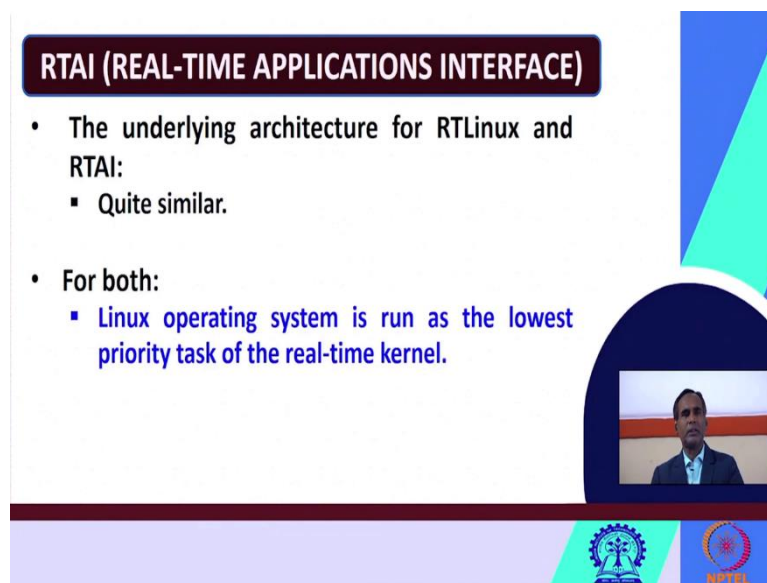
So let us now see the limitations of the interrupt abstraction model. So, here I have already told you in this interrupt abstraction model the Linux kernel runs on modified, the Linux kernel runs on a modified. The kernel loadable modules they actually include the deal with the real-time tasks, the kernel loadable modules are the real-time tasks they are written maybe in form of the real-time tasks they are written in the form of the kernel loadable modules.

The APIs are used for real-time task, so for real-time task, the real-time task so many API's are used. So, the real-time you know that nowadays the real-time applications they are becoming more and more complex. As they are becoming popular they are also becoming more

and more complex. So, in this interrupt abstraction model, what are the keys to build, to develop the complex real-time applications. So some of these what keys are like, you have to provide a memory protection, you have to provide efficient resource management.

We have to provide process isolation these are the key issues these are the major challenges to build the complex applications using the interrupt abstraction model. So, they are the important issue they are the key challenges or the major issues for building real complex real-time applications.

(Refer Slide Time: 08:39)



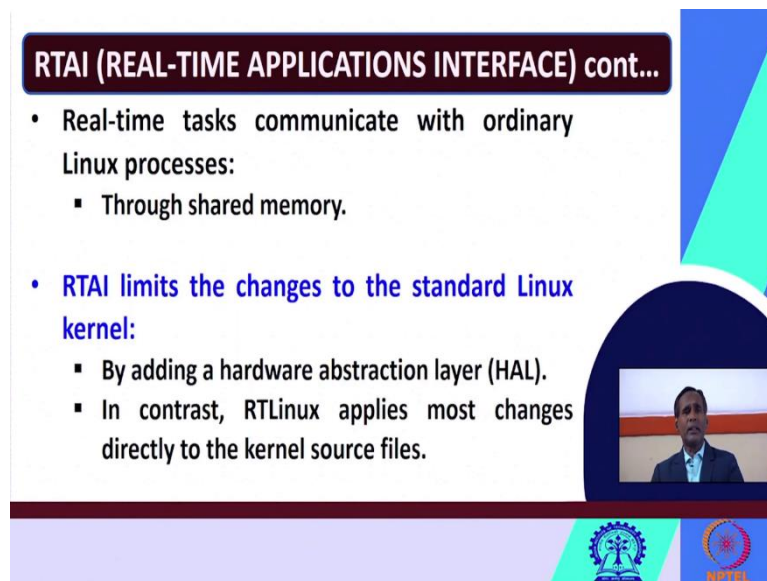
RTAI (REAL-TIME APPLICATIONS INTERFACE)

- The underlying architecture for RTLinux and RTAI:
 - Quite similar.
- For both:
 - Linux operating system is run as the lowest priority task of the real-time kernel.

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

Now let us see another variation that is called as RTAI. It stands for real-time applications interface. The underlying architecture for RT-Linux and for RTAI. RT-Linux means this real-time version of Linux. So, the underlying architecture for RT-Linux and RTAI are quite similar. For both the systems that is RTAI and RT-Linux, the Linux operating system is run as the lowest priority tasks of the real-time kernel. For both RTAI and RT-Linux, the Linux operating system is run as what not the highest priority, it runs as the Linux operating system is run as the lowest priority task of the real-time kernel.

(Refer Slide Time: 09:20)



RTAI (REAL-TIME APPLICATIONS INTERFACE) cont...

- Real-time tasks communicate with ordinary Linux processes:
 - Through shared memory.
- RTAI limits the changes to the standard Linux kernel:
 - By adding a hardware abstraction layer (HAL).
 - In contrast, RTLinux applies most changes directly to the kernel source files.

The slide features a video inset of a man in a suit on the right side. At the bottom, there are logos for IIT Bombay and NPTEL.

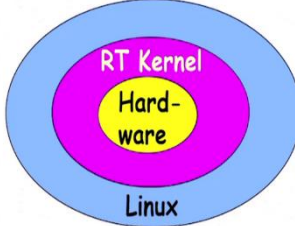
So now, let us see how this what task communicate in RTAI, the real-time task in RTAI they communicate with ordinary Linux processes through a shared memory and this RTAI limits the changes to the standard Linux kernel. RTAI limits the changes that are to be made to the standard Linux kernel, how it does this. So, just like I have told you HAL in case of this what interruption interrupt abstraction approach. So, this RTAI limits the changes to the standard Linux kernel by adding a separate layer called as this hardware abstraction layer. In contrast to what RT-Linux does?

So, in RTAI or RTAI limits the changes to the standard Linux kernel how? By adding a separate layer called as this hardware abstraction layer but in RT-Linux what happens it applies or RT-Linux applies most changes directly to the kernel source files. So, in case of RT-Linux, what it does it applies the almost all of the changes directly, to what, to the kernel source files, whereas RTAI limits the changes to the standard Linux kernel. This is how these two operating systems they are different. Now let us see details about this RT-Linux because this is a very a commonly used operating system in the industry or even in academics.

(Refer Slide Time: 10:43)

RT LINUX

- **RT kernel intercepts all interrupts:**
 - If an interrupt is to cause a RT task to run, the RT kernel preempts Linux (if Linux is running that time) and lets the RT task run.



The diagram illustrates the architecture of RT-Linux. It consists of three concentric layers: a central yellow circle labeled 'Hard-ware', a middle pink ring labeled 'RT Kernel', and an outer light blue ring labeled 'Linux'. To the right of this diagram is a small video inset showing a man speaking. At the bottom right of the slide, there are logos for IIT Bombay and NPTEL.

So let us see details of RT-Linux. So, this RT kernel intercepts all the interrupts. I have already told you that Linux is a general-purpose operating system it can be made what real-time oriented the real-time kernel it intercepts all interrupts that means if an interrupt is to cause a real-time task to run, the real-time kernel it preempts Linux kernel, if Linux is running at that time if it is not running then it is fine.

But if Linux is running at that time and an interrupt is to cause the real-time task to run then the real-time kernel it preempts the Linux and what does it do it lets the real-time tasks to run so then it will allow the real-time tasks to run. So, how does it look like you can see that these at the bottom layer this hardware layer is there on the other side Linux is there. So, this RT kernel it sits in between this hardware and the Linux. You can mark that this RT kernel it sits in between the hardware and the Linux. This is how this RT-Linux the architecture of the RT-Linux looks like.

(Refer Slide Time: 11:52)

RT LINUX cont...

- Different modules of the kernel can be selectively loaded:
 - To fit into the available memory.
- Both fixed and dynamic priority levels are supported.
- Context switch time is 15 μ Sec

The slide features a video inset of a man in a suit speaking. At the bottom, there are logos for IITM (Indian Institute of Technology Madras) and NPTEL (National Programme on Technology Enhanced Learning).

So, in case of the RT-Linux different modules of the kernel can be selectively loaded to fit into the available memory, you may not require you need not what a load almost all of the modules, what you can do, only those modules are required for you they can be fit into, they can be loaded.

So different modules of the kernel can be selectively loaded depending upon your requirement which are very much important relevant these modules can be loaded in order to fit into the available memory. Both fixed and dynamic priority levels are supported.

We know that in order to make it real-time we require also static priority levels, along with dynamic priority levels. So, in case of this RT-Linux both fixed that means static as well as dynamic priority levels are supported. The context switch time in the order of 15 microsecond in case of RT-Linux.

(Refer Slide Time: 12:43)

RT LINUX cont...

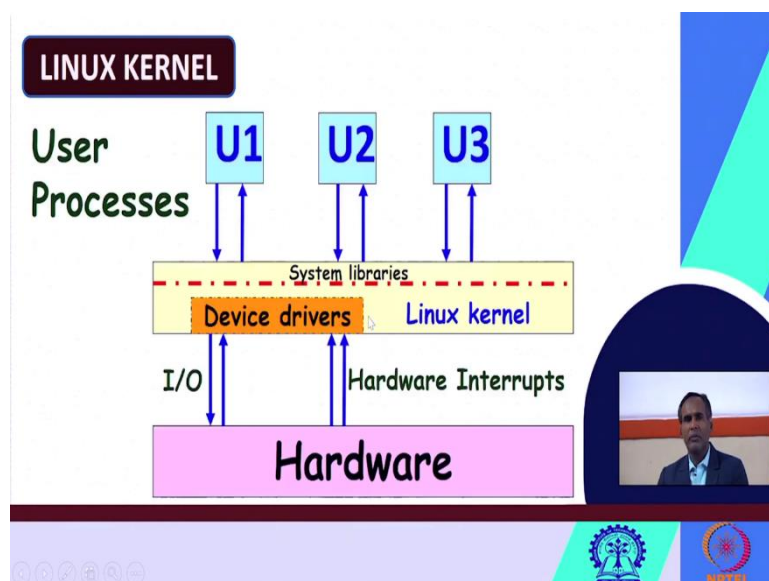
- The biggest advantage:
 - Source code is available.
- If a specific driver or a specific part of the kernel is known to be causing undesirable delay:
 - The code can be fine tuned.

The slide features a video inset of a man in a suit in the bottom right corner. At the bottom of the slide, there are navigation icons and logos for IIT Bombay and NPTEL.

Let us see what is the biggest advantage. The biggest advantage I have already told you this Linux is a general-purpose software and is a free software. The biggest advantage of RT-Linux is that the source code is available, the source code is available online and if a specific driver or a specific part of the kernel is known to be causing undesirable delay then what he can do simply that that code can be fine-tuned.

This is the advantage of RT-Linux since the source code is available, if a specific driver or a specific part of the kernel is known to be causing undesirable delay, it is giving you some delays then the code can be easily fine-tuned.

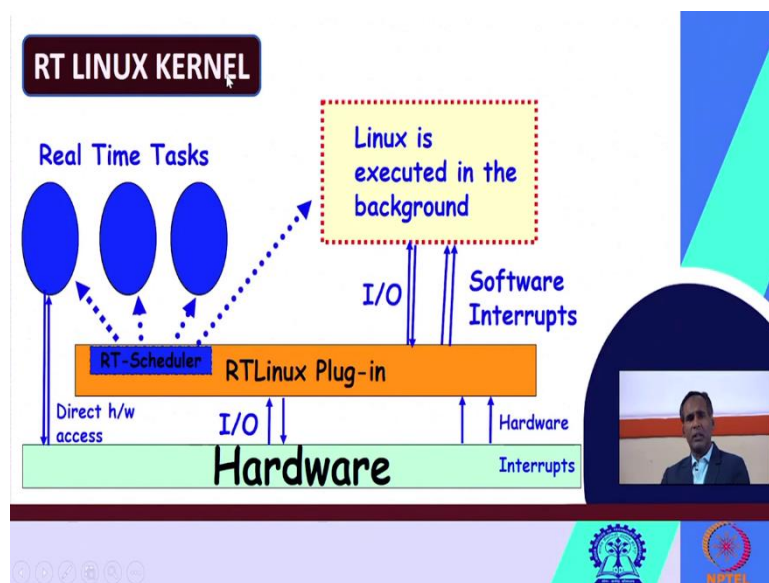
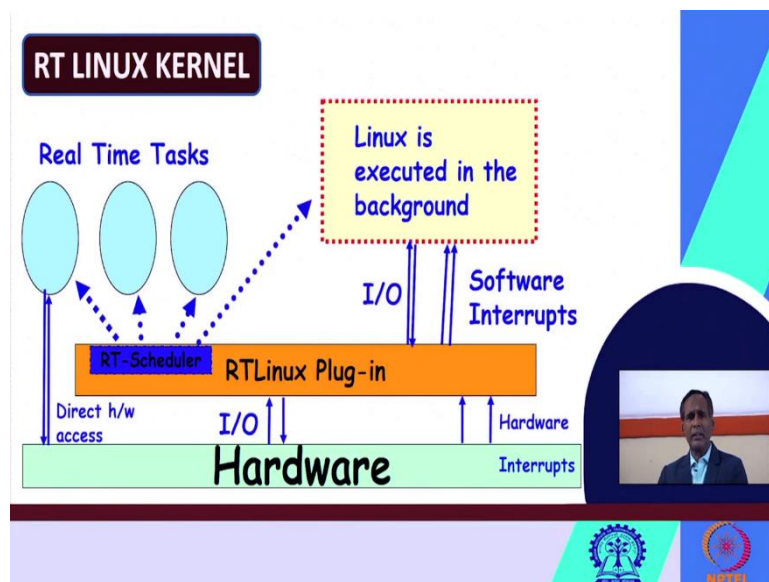
(Refer Slide Time: 13:23)

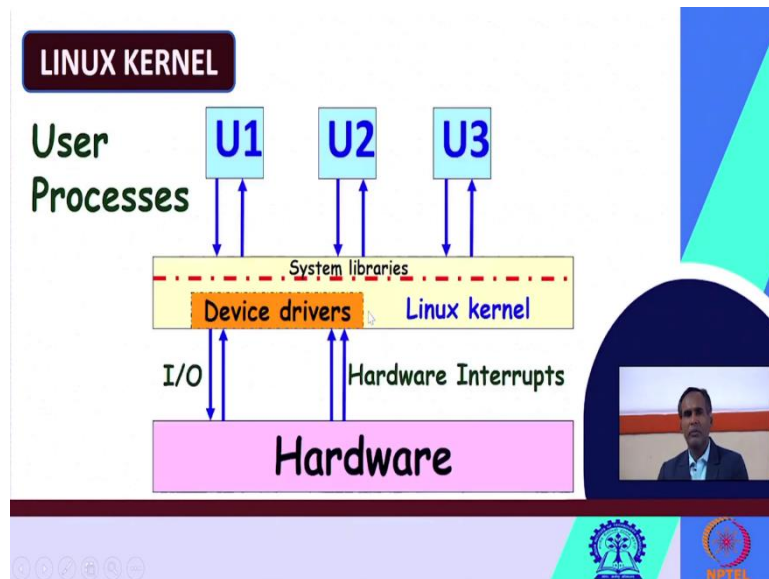


Let us see this Linux kernel architecture then you will see this RT Linux kernel architecture then you will compare. In this Linux kernel architecture what is happening you see at the bottom layer the hardware is there and the top layer the user processes U1, U2, U3 are the user processes and in between that the Linux kernel is there and in the Linux kernel you can see there are several device drivers available and on top of this the system libraries are there.

The various user processes they can communicate they interact to the Linux kernel through these system libraries. Then this Linux kernel can interact with the hardware or the device drivers present in the Linux kernel they can interact with the hardware through what you can say IO operations and hardware interrupts. This is how the Linux kernel looks like.

(Refer Slide Time: 14:20)



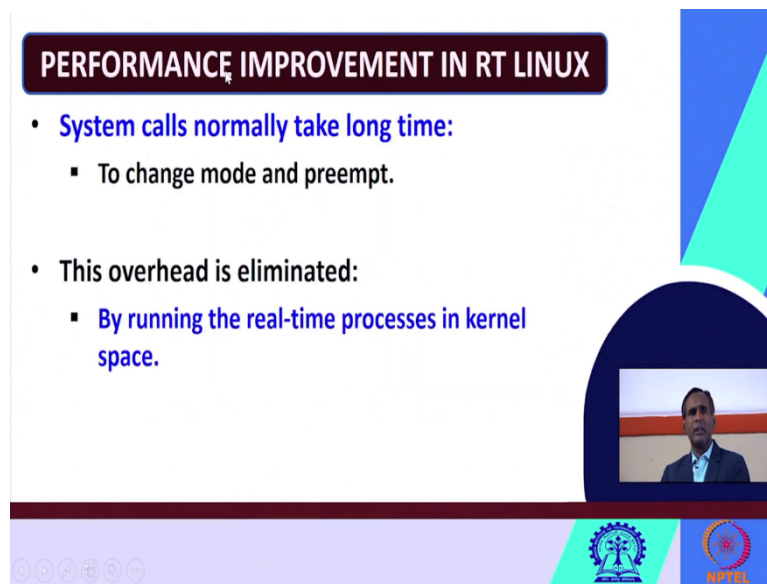


Then let us see how does this RT-Linux kernel look like please mark the difference. So, in case of this RT-Linux at the bottom layer hardware is there and on top of the hardware instead of having the Linux we are having RT-Linux plug in. Just see compared here it was what it was there was this Linux kernel in case of Linux kernel above the hardware layer the Linux kernel it there and so many device drivers out there but in case of RT-Linux you can see, so on top of this, the RT-Linux plug in is there.

And there are several real-time schedulers in this RT Linux and whatever they are earlier in the case of the device, what simple Linux kernel, so here Linux kernel, in case of the RT-Linux the Linux kernel is executed in the background. So, in between or above the hardware the RT-Linux is there and the Linux is executed in the background, it communicates with the RT-Linux through various IO operations on software interrupts.

And this RT-Linux plugin it communicates it interacts with the hardware through various IO operations on the hardware interrupts. This the RT-scheduler the real-time scheduler presents in this, the real-time scheduler present in this RT-Linux plug in. So, there may be several, real-time schedulers or at least there is one the real-time scheduler. The real-time scheduler present in the RT-Linux plugin, so, it schedules all these real-time tasks. The real-time tasks they can also directly assess the hardware. So, this is how the RT-Linux kernel it looks like. Please mark the difference between these what Linux kernel and the RT-Linux kernel.

(Refer Slide Time: 16:00)



PERFORMANCE IMPROVEMENT IN RT LINUX

- **System calls normally take long time:**
 - To change mode and preempt.
- **This overhead is eliminated:**
 - **By running the real-time processes in kernel space.**

The slide features a dark blue header with the title in white. The main content is on a white background with blue text. A video inset in the bottom right shows a man in a suit. The footer contains navigation icons and logos for IIT Bombay and NPTEL.

Then we will see the performance improvement in RT-Linux. In case of the RT-Linux the system calls normally take a very long time to change the mode and preempt. So, how then how this overhead can be eliminated, how that can be brought improvement in the performance in RT-Linux. So, this overhead as I have already told you overhead means what?

The system call or the system calls, they are normally taking a very long period of time to change the mode and the preempt. So, this overhead can be reduced can be eliminated this overhead can be eliminated by running the real-time process in kernel space. So, if you can run if you can execute the real-time processes in the kernel space, then perhaps this overhead can be eliminated and the system calls will not take very long time.

So, this overhead can be eliminated by running the real-time processes in kernel space. So, I have already told you this overhead can be eliminated if the real-time processes can be executed in the kernel space.

(Refer Slide Time: 16:51)

BENEFITS OF EXECUTING RT TASKS IN KERNEL SPACE

- Prevents threads from being swapped-out.
- Threads are executed in processor supervisor mode:
 - Have full access to the underlying hardware.
- RTOS and the application share a single address space:
 - System call becomes a simple function call (software interrupt has higher overhead).

PERFORMANCE IMPROVEMENT IN RT LINUX

- System calls normally take long time:
 - To change mode and preempt.
- This overhead is eliminated:
 - By running the real-time processes in kernel space.

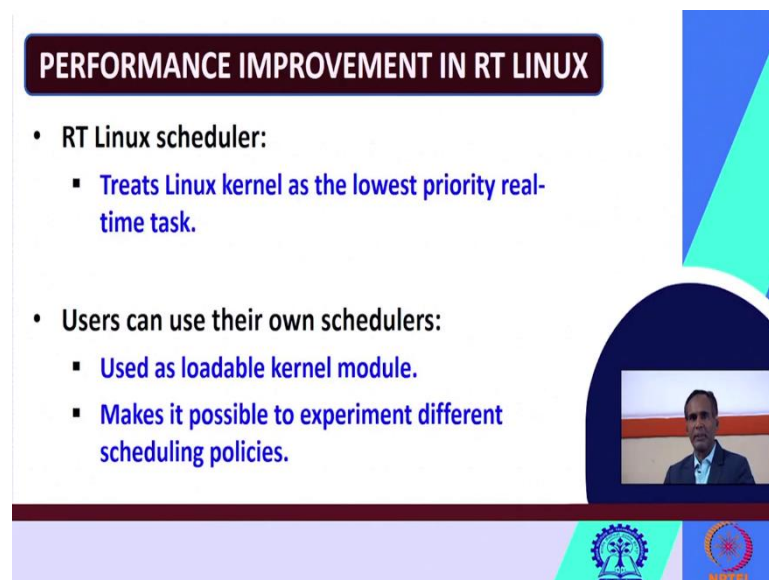
Let us quickly see what are the benefits you will get if we can execute the real-time tasks in the kernel space. First one first advantage is that it prevents the threads from being swapped out. Then the threads are executed in a processor supervisor mode.

And the threads have full access to the underlying software. We are discussing the advantages of executing the real-time tasks in kernel space. So, the threads are executed in the processor supervisor mode and they have full access to the underlying hardware. So, the real-time operating system and the application share are both the real-time operating system and the application they share a single address space. Due to using or due to executing the real-time tasks in kernel space.

The real-time operating system and the application both they share a single address space. So, now the system call becomes a simple function call because I have already told you earlier that the system calls normally, they take very long time, but this overhead can be eliminated or reduced by learning the real-time process in the kernel space.

So, by executing the real-time task in the kernel space, so here the real-time operating system on the application they are sharing the single address space and hence as a result the system call just to becomes a very simple function call. You may remember that the software interrupt has a higher overhead. So, this is these are the advantages that we can get by executing the real-time tasks in the kernel space.

(Refer Slide Time: 18:24)



PERFORMANCE IMPROVEMENT IN RT LINUX

- RT Linux scheduler:
 - Treats Linux kernel as the lowest priority real-time task.
- Users can use their own schedulers:
 - Used as loadable kernel module.
 - Makes it possible to experiment different scheduling policies.

The slide features a dark blue header with the title in white. The main content is on a white background with blue and red accents. A small video inset in the bottom right shows a man speaking. Logos for IIT Bombay and NITEL are at the bottom right.

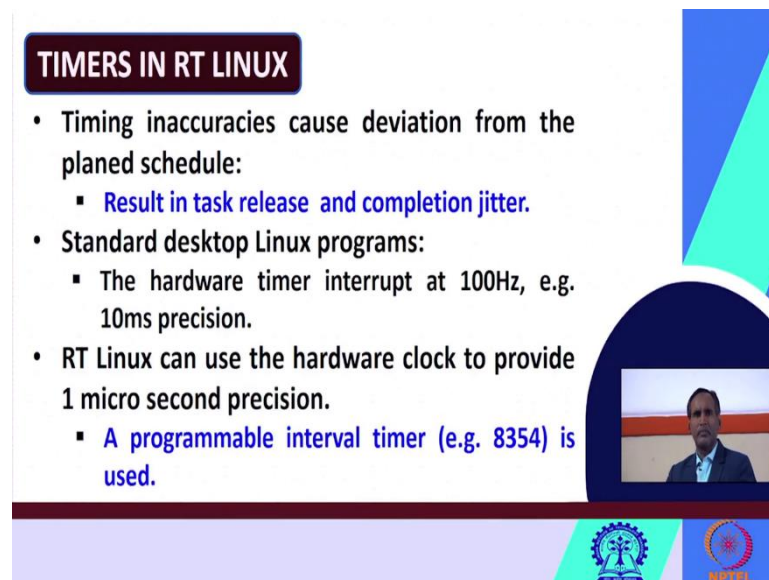
Now again, I go back to our point where I have left the performance improvement in RT-Linux. So, RT-Linux scheduler it treats the Linux kernel as the lowest priority level, real-time task. This I have already told you. Please remember that this RT-Linux scheduler, it always treats the Linux kernel as the lowest priority real-time tasks. Here, the users can use their own schedulers.

In RT-Linux, the users, they can use their own scheduler and these schedulers that can be used as can be written as a loadable kernel module, So, the users can use their own schedulers, these schedulers are used as loadable kernel modules. This thing makes it possible to experiment different scheduling policies.

So, you know, there are different scheduling policies, you want to experiment with all those different scheduling policies. So, if you can use your own scheduler, if it can be used as a

loadable kernel module, then as a programmer, it will be possible to experiment with the different scheduling policies. So, these are some of the advantages you can say, of this RT-Linux.

(Refer Slide Time: 19:31)



TIMERS IN RT LINUX

- Timing inaccuracies cause deviation from the planed schedule:
 - Result in task release and completion jitter.
- Standard desktop Linux programs:
 - The hardware timer interrupt at 100Hz, e.g. 10ms precision.
- RT Linux can use the hardware clock to provide 1 micro second precision.
 - A programmable interval timer (e.g. 8354) is used.

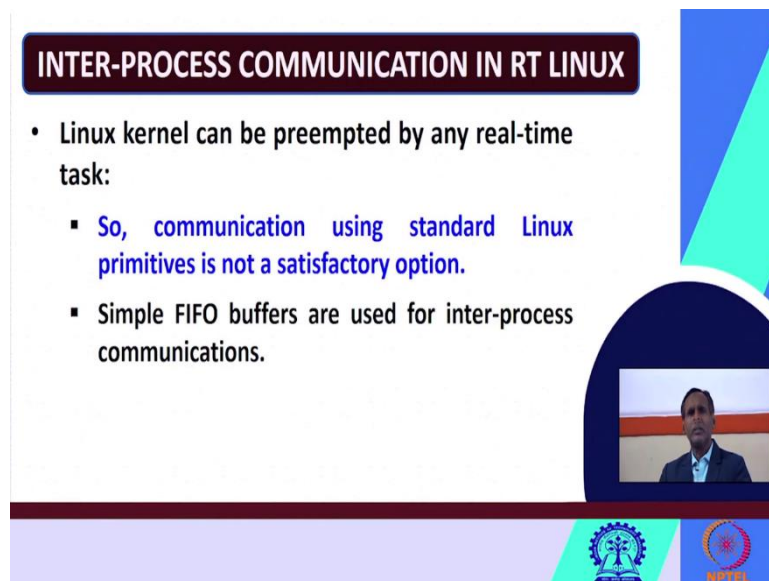
The slide features a dark blue header with the title 'TIMERS IN RT LINUX' in white. The main content is a list of bullet points. To the right of the text is a vertical decorative bar with blue and green segments. Below the text is a video inset showing a man speaking. At the bottom right, there are logos for IIT Bombay and NPTEL.

Now let us see how does the timers are provided, how does the timers work in RT-Linux? You know that timing inaccuracy is they cause deviation from the planed schedule. If the time service it is provided or the timing it is provided is inaccurate then obviously it will cause a what deviation from the planed schedule. The timing inaccuracies they cause the what deviation from the planed schedule.

This may result in task release and completion jitter. If the timings are inaccurate then this will result in task release and completion jitter. So, that way I have already told you somewhat you can say roughly delay. So, the standard desktop Linux programs in standard desktop Linux programs the hardware timer interrupts occurs at normally 100 hertz for example 10 milliseconds precision but RT-Linux can be used or the RT-Linux can use the hardware clock to provide a 1 microsecond precision.

So, in case of the standard desktop Linux programs, the hardware timer interrupts occur at 100 megahertz for example 10 millisecond precision. But the real-time Linux can use the hardware clock to provide a finer precision of the order of say 1 microsecond precision here a programmable interval timer for example 8354 it is used. So, this is how the timers in RT-Linux they work.

(Refer Slide Time: 20:57)



INTER-PROCESS COMMUNICATION IN RT LINUX

- Linux kernel can be preempted by any real-time task:
 - So, communication using standard Linux primitives is not a satisfactory option.
 - Simple FIFO buffers are used for inter-process communications.

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 in the bottom right shows a man in a suit. Logos for IIT Bombay and NPTEL are at the bottom right.

Then let us see how IPC works in RT-Linux. How this inter process communication was in RT-Linux, the Linux kernel can be preempted by any real-time tasks that I have already told you. So, you see RT-Linux there are two kernels Linux kernel and what real-time kernel that is why this RT-Linux is known as a dual kernel, here there are two kernels Linux kernel and RT real-time kernel.

Now let us discuss how this inter process communication occurs in RT-Linux. The Linux kernel can be preempted by any real-time task, so the communication using the standard Linux preemptive is not a satisfactory option. So, since this Linux kernel it can be preempted by any real-time tasks, so making communication using the standard Linux preemptives it is not of course a satisfactory opening option. So, what you can do? now let us see the simple FIFO buffers are used for inter process communications. So, in this RT-Linux simple FIFO buffers they are used for the inter process communications.

(Refer Slide Time: 21:59)

SHORTCOMINGS OF RT-LINUX

- **Duplicated coding effort:**
 - Tasks cannot make full use of existing Linux system services: file systems, networking, etc.
- **Fragile execution environment:**
 - Processes run in unprotected kernel space.
- **Programs need to be written using a small subset of POSIX API:**
 - **Moving existing Linux code to RT environment becomes difficult.**

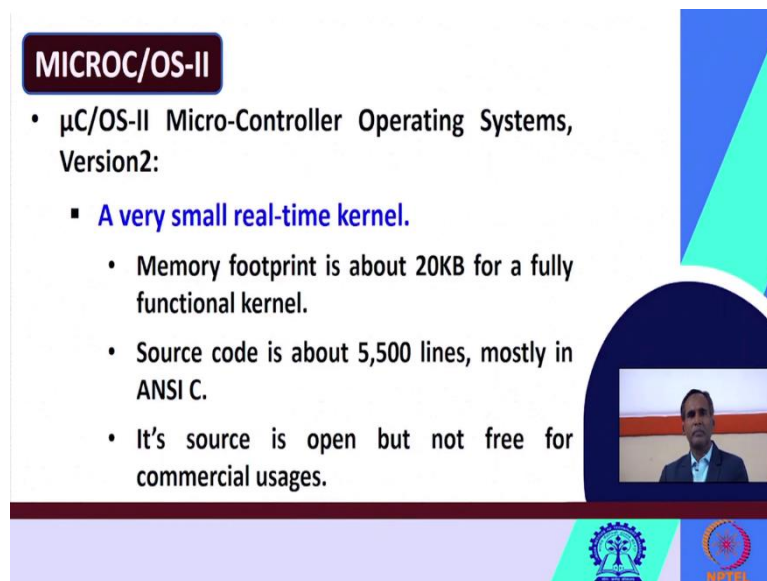
Now at the end let us quickly look at these after shortcomings of RT-Linux, what are the possible disadvantages? First one is the duplicated coding effort, what do you mean by duplicated coding effort? So, something you have to write twice the coding effort is just duplicated. Tasks cannot make full use of existing Linux system services. So, what are the existing Linux system services or file systems networking etcetera.

So, the tasks in RT-Linux they cannot make full use of the existing Linux system services. So, you have to again write down the code for same. So, you are putting duplicated code, you are making duplicated the coding effort, the coding effort is duplicated. Then fragile execution environment. The processes run in unprotected kernel space. That is another major disadvantage of RT-Linux.

The processes in RT-Linux, they run in unprotected kernel space, which is not desirable, then programs need to be written using a small subset of POSIX API, this RT-Linux does not use all the functionalities of this or all the APIs of POSIX, it just uses a small subset of POSIX API. So, then what will happen for complex real-time applications you cannot write because it is using only a small set of POSIX-API. So, you cannot write what complex real-time applications using this RT-Linux.

So, programs need to be written in RT-Linux programs need to be written using a very small subset of POSIX API and hence moving the existing Linux code to RT environment becomes difficult. You have to again write. It is very much difficult to write the complex real-time applications from the moving the existing Linux code to RT environment it becomes very much difficult. So, these are some of the possible shortcomings of RT-Linux.

(Refer Slide Time: 23:43)



MICROC/OS-II

- μ C/OS-II Micro-Controller Operating Systems, Version2:
 - **A very small real-time kernel.**
 - Memory footprint is about 20KB for a fully functional kernel.
 - Source code is about 5,500 lines, mostly in ANSI C.
 - It's source is open but not free for commercial usages.

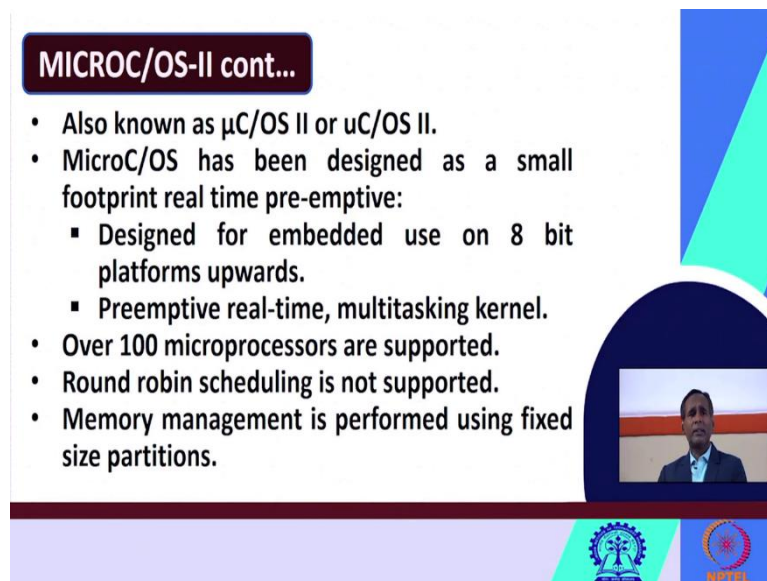
The slide features a dark blue header with the title 'MICROC/OS-II' in white. Below the title, there are three bullet points. The first is a main bullet point, and the second is a sub-bullet point. The third is a list of three sub-bullet points. To the right of the text, there is a video inset showing a man in a suit. At the bottom right, there are two logos: one for IIT Bombay and one for NPTEL.

So, this is something about this RT-Linux, then we will go to another real-time operating system called MICROC operating system 2, even if this μ C, you can use for MICROC. So, MICROC operating system or μ C operate OS2 it represents a MICROC operating system 2. So, microcontrollers and this MC stands for microcontroller operating system version two, so MICROC operating system 2 stands for microcontroller operating systems version 2 in place of micro you can use this symbol μ , this a very small real-time kernel.

And here the memory footprint is about just 20 kilobytes for a fully function kernel because it is a very small real time kernel. So, this memory size is about to just 20 kilobytes for a fully functional kernel. The source code about the source code is about just 5500 lines and it is written mostly in ANSI C.

It is a source is open but not free for commercial usage. You can get the source from the net but not for commercial usages only for academic research usages you can use it. So, its source code is open but not free for commercial usages. And it is only available for academic and research applications.

(Refer Slide Time: 24:59)



MICROC/OS-II cont...

- Also known as μ C/OS II or uC/OS II.
- MicroC/OS has been designed as a small footprint real time pre-emptive:
 - Designed for embedded use on 8 bit platforms upwards.
 - Preemptive real-time, multitasking kernel.
- Over 100 microprocessors are supported.
- Round robin scheduling is not supported.
- Memory management is performed using fixed size partitions.

The slide features a dark blue header with the title in white. The main content is on a light blue background. A video inset on the right shows a man in a suit speaking. At the bottom, there are logos for IIT Bombay and NPTEL.

Let us see the details about MICROC operating system 2. So, it is also known as mu C OS 2 or UCOS2 any notation you can use. This MICROC operating system has been designed initially as a small footprint in real-time pre-emptive. It was designed for embedded use on the 8-bit platforms on upwards. So, initially this MICROC, it was designed for embedded applications, for embedded use on 8-bit platform upwards.

This uses a pre-emptive real-time and a multitasking kernel. Over 100 microprocessors are supported by MICROC OS 2. In MICROC OS 2, round robin scheduling is not supported please see this MICROC OS 2 in this operating system round robin scheduling it is not supported. The memory management is performed using fixed size partition. So, in case of MICROC OS 2, the memory management how it is performed? It performed using fixed size partitions.

(Refer Slide Time: 26:02)

ECOS (EMBEDDED CONFIGURABLE OS)

- An open source, royalty-free RTOS:
 - Intended for embedded applications.
- Highly configurable:
 - Allows the OS to be customised to application requirements.

<http://ecos.sourceware.org>

- eCos is targeted at high-volume applications:
 - Consumer electronics, telecommunications, automotive, etc.

MICROC/OS-II cont...

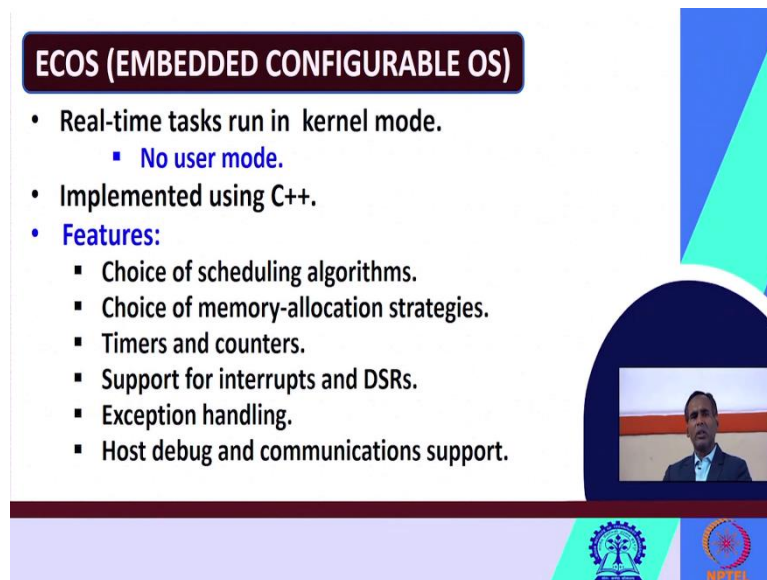
- Also known as μ C/OS II or uC/OS II.
- MicroC/OS has been designed as a small footprint real time pre-emptive:
 - Designed for embedded use on 8 bit platforms upwards.
 - Preemptive real-time, multitasking kernel.
- Over 100 microprocessors are supported.
- Round robin scheduling is not supported.
- Memory management is performed using fixed size partitions.

We will see another real-time operating system that is ECOS. So, here you have seen a MICROC OS2, here you will see ECOS. ECOS stands for embedded configurable operating system. This is also an open source. ECOS this is an open source, royalty-free RTOS. It is intended for embedded applications also, this operating system is highly configurable, it is highly configurable, it allows the operating system to be customized to application requirements you can easily customize this, what operating system according to your own needs according to your own requirements.

So, this is highly configurable, it allows the operating system to be customized to the application needs or the application requirements you can get this ECOS from this website because this is freely available. The ECOS is targeted at high volume applications. So, normally

when your applications or the applications you are using they are of very high volume then ECOS is targeted those applications. So, ECOS is targeted at high volume applications such as for customer electronics, for telecommunications, for automotive, automotive, etcetera. For those types of high-volume applications ECOS can be used.

(Refer Slide Time: 27:14)



ECOS (EMBEDDED CONFIGURABLE OS)

- Real-time tasks run in kernel mode.
 - No user mode.
- Implemented using C++.
- Features:
 - Choice of scheduling algorithms.
 - Choice of memory-allocation strategies.
 - Timers and counters.
 - Support for interrupts and DSRs.
 - Exception handling.
 - Host debug and communications support.

So, the ECOS here or the real-time tasks run in kernel mode. Please remember in ECOS the real-time tasks run in kernel mode, that is no user mode. ECOS is normally implemented using C plus plus. The features those are supported by ECOS are as follows. So, choice of scheduling algorithm. So, in ECOS, ECOS supports choice of scheduling algorithm, you can choose your scheduling algorithm you can make choice of the scheduling algorithm that you like that you want to use.

Choice of memory applications-allocation strategies, similarly you can use your preferred memory allocation strategies. Also, it supports the timers and counters, it also supports for the interrupts and the DSRs. And also, exception handling is provided is supported in ECOS supports exception handling, and host debug and communication support. So, ECOS also support the host debug and communication support. So, these are some of the specific features of ECOS, we will not go into details of ECOS.

(Refer Slide Time: 28:18)

VRTX

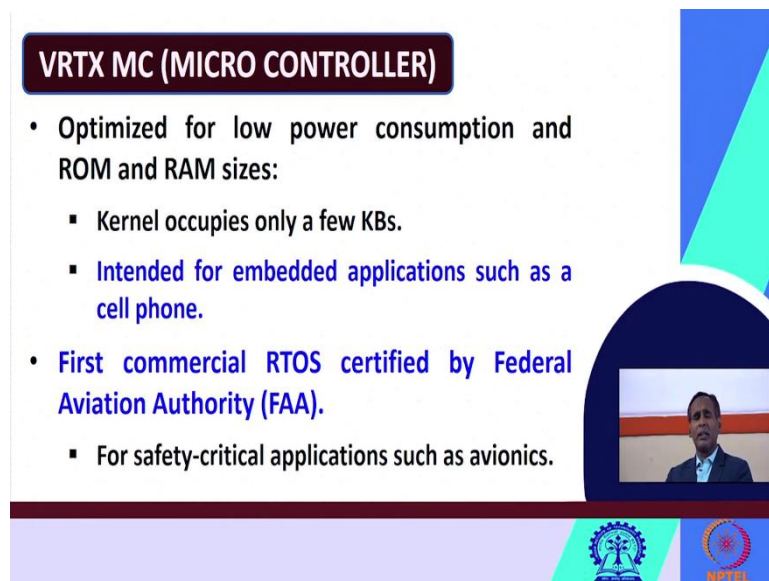
- **Versatile Real-Time Executive (VRTX):**
 - Developed by Ready systems in 1980s.
 - Acquired by Mentor Graphics in 1995.
- VRTX is suitable for:
 - SoC architectures.
 - Traditional board-based embedded systems.
- Originally 4KB in size:
 - Did not support many required functionalities.
 - Could not be used as a full RTOS.
- Now comes in 2 versions like **VRTX SA** and **VRTX MC**.

The slide features a dark blue header with the title 'VRTX' in white. The main content is on a white background with blue and red accents. A small video inset shows a man speaking. At the bottom, there are logos for IIT Bombay and NPTEL.

Now, let us see another real-time operating system which is called VRTX, very popular operating system. VRTX stands for versatile real-time executive. It was developed by Ready Systems in year in 1980s, then it was acquired by Mentor Graphics in 1995. Then, let us say VRTX is suitable for which operating system, for which kinds of systems? So, normally VRTX is used for SOC architecture, you know SOC system on a chip. So, VRTX is suitable for system on a chip or SOC architectures. It is also very much suitable for traditional board based embedded systems.

So, originally the VRTX its size was just 4KB, originally it was 4KB in size. Now, since this size is very small, it did not support many required functionalities only the basic functionalities it has supported, it could not be used as a real-time operating system because its size is less, it does not support many required functionalities, so it could not be used as a full real-time operating system. So, this VRTX actually comes in two versions like VRTX SA and VRTX MC. Let us first see VRTX MC then we will see about VRTX SA.

(Refer Slide Time: 29:35)



VRTX MC (MICRO CONTROLLER)

- Optimized for low power consumption and ROM and RAM sizes:
 - Kernel occupies only a few KBs.
 - Intended for embedded applications such as a cell phone.
- First commercial RTOS certified by Federal Aviation Authority (FAA).
 - For safety-critical applications such as avionics.

The slide features a dark blue header with the title in white. The main content is on a white background with blue and red accents. A video inset on the right shows a man in a suit. At the bottom, there are logos for a university and NPTEL.

VRTX MC stands for VRTX micro controller, this VRTX MC it is optimized for low power consumption and ROM and RAM sizes. The kernel in VRTX MC it occupies just only a few kilobytes. So, why it was intended VRTX MC it was intended for embedded applications such as a cell phone. This was the first commercial real-time operating system certified by what, FAA, Federal Aviation Authority for safety critical applications such as avionics.

So, please remember very much important, this VRTX MC was the first commercial real-time operating system certified by FAA Federal Aviation Authority for which type of applications, for safety critical applications such as avionics. I hope you have already known what is a safety critical application? Examples like your avionics systems, a rocket is flying or the aero plane is flying it is what a navigation system these are all examples of safety critical applications.

(Refer Slide Time: 30:41)

VRTX MC cont...

- Targeted for products:
 - Where memory efficiency and low power consumption are important.
- Product examples:
 - Set-top boxes, digital cameras, video game systems, routers and handheld computing devices.
 - Many Motorola cell phones use VRTX MC.

So, what are the targeted products for VRTX MC? VRTX MC was targeted for the products where the memory efficiency and low power consumption is important. So, the cases or the products where memory efficiency is very much important, the power consumption should be very low. For those types of cases, VRTX MC was targeted.

Let us see some of the quick examples of the products where VRTX MC was used. So, the product examples where VRTX MC can be used are such as set-top boxes, digital cameras, video game systems, routers, and handheld computing devices. You can see that many Motorola cell phone systems they use VRTX MC. So, many of these cell phones are produced by Motorola they use VRTX MC.

(Refer Slide Time: 31:32)

VRTX SA (SCALABLE ARCHITECTURE)

- Functional super-set of VRTXmc
 - POSIX-RT compliant.
 - Designed for medium and large applications.
 - Extensive I/O support.
 - Memory protection.
 - Reduced interrupt and task-switch latency.

Then quickly let us see at VRTX SA. So, here SA stands for what, SA stands for scalable architecture. So, this VRTX SA is a functional superset of the VRTX MC which you have already seen. So, besides that, this VRTX SA it is POSIX-RT compliant, we have already seen what POSIX is.

This VRTX SA is designed for medium and large-scale applications, VRTX SA is designed for medium and large applications. So, in VRTX, there is extensive IO support, memory protection is also provided in VRTX SA. There is reduced interrupt and text switch latency in VRTX SA, these are some of the unique characteristics of VRTX SA.

(Refer Slide Time: 32:17)

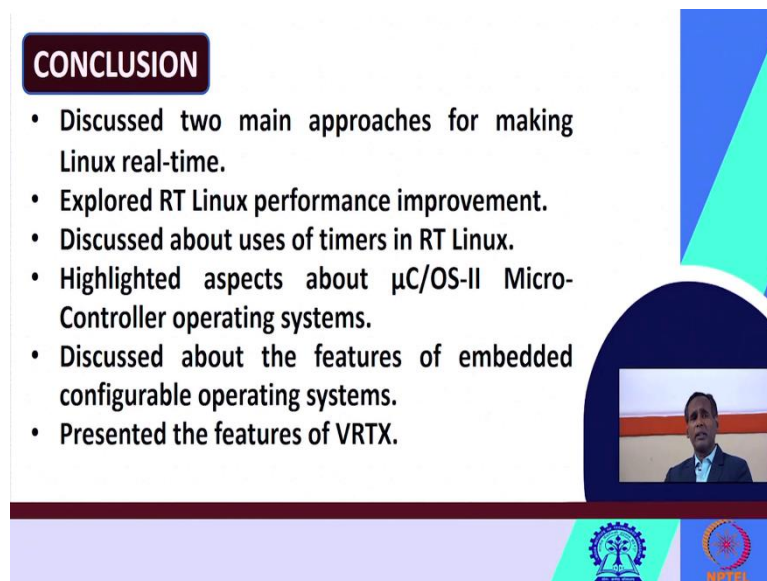
VRTX SA (SCALABLE ARCHITECTURE) cont ...

- Other features:
 - Support for priority inheritance and multiprocessing.
 - System calls fully preemptible and deterministic.
- Example use:
 - Hubble space telescope.

The slide includes a small image of the Hubble Space Telescope in orbit and a video inset of a speaker. Logos for IIT Bombay and RPTEL are visible at the bottom.

Some of the other features let us see of VRTX SA. It supports for priority inheritance and multi-processing. So, I hope what is priority inheritance etcetera you have already known earlier. VRTX SA supports for priority inheritance and multi-processing. Here, in VRTX SA the system calls fully preemptible and deterministic. In VRTX SA the system calls are fully preemptible and deterministic. So, one of the examples use of VRTX SA is the Hubble Space Telescope, you can see a telescope here, this Hubble Space Telescope in this telescope this VRTX SA real time system was used.

(Refer Slide Time: 32:58)



CONCLUSION

- Discussed two main approaches for making Linux real-time.
- Explored RT Linux performance improvement.
- Discussed about uses of timers in RT Linux.
- Highlighted aspects about μ C/OS-II Micro-Controller operating systems.
- Discussed about the features of embedded configurable operating systems.
- Presented the features of VRTX.

The slide features a dark red header with the word 'CONCLUSION' in white. Below the header is a list of six bullet points. To the right of the text is a video inset showing a man in a suit speaking. At the bottom of the slide, there are two logos: the Indian Institute of Technology (IIT) logo on the left and the NPTEL logo on the right. The slide has a decorative background with blue and green geometric shapes.

So, today we have discussed the two main approaches for making Linux real-time, we have explored the real-time Linux performance improvement, we have discussed about the uses of timers in RT-Linux, also how message passing can be done, how inter process communication can be done in real-time Linux we have seen, we have seen what two other earlier time operating system such as μ C operating system 2, Micro C operating system 2 and VRTX, we have highlighted the different aspects about μ C, Micro C operating system 2.

We have also discussed about the features of the embedded configuration operating system or ECOS. We have also presented another operating system VRTX, the different features of VRTX, two variants VRTX MC and VRTX SA, those things we have discussed today. So, that is what we have discussed on today. You have mainly discussed some of the, some survey on the commonly available real-time operating systems in the market.

(Refer Slide Time: 34:00)



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. A video inset in the bottom right corner shows a man in a suit speaking. The slide is decorated with a blue and green geometric design on the right side and a dark red horizontal bar at the bottom. Logos for IIT Bombay and NPTEL are visible in the bottom right corner.

We have taken those things from these two books. Thank you very much.