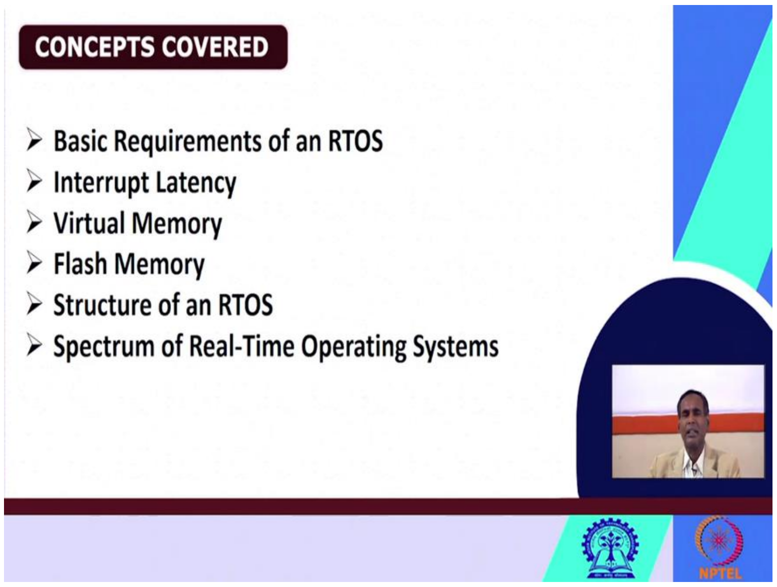
Real-Time Systems

Professor Durga Prasad Mohapatra
Department of Computer Science and Engineering
National Institute of Technology Rourkela

Lecture 37: A Few Basics in Real-Time Operating Systems

Good morning to all of you. Now, let us today, we will start the new topic, a new chapter on Real-Time Systems. That is, today, we will discuss about some of the commercial real-time operating systems available. But before going to the commercial, the available commercial real-time systems, we should first look at some of the few basics in real-time operating systems, then we will go to the popularly available some of the commercial real-time operating systems.

(Refer Slide Time: 00:50)

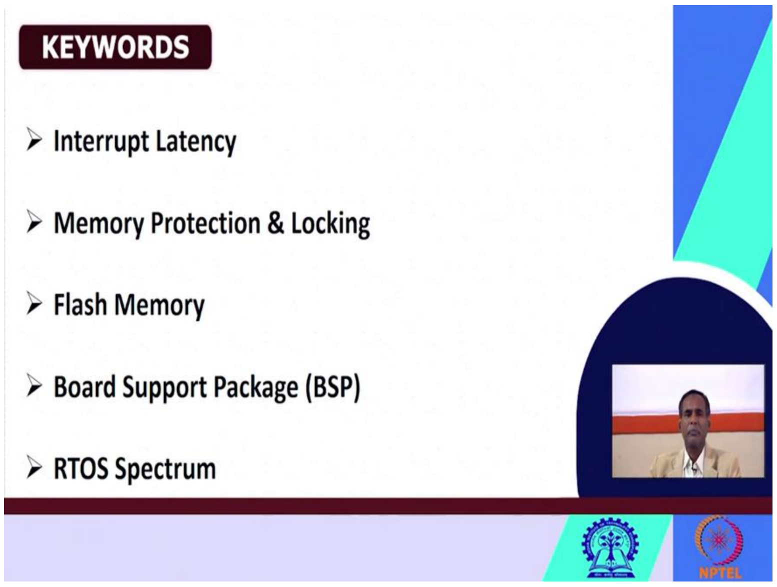


CONCEPTS COVERED

- Basic Requirements of an RTOS
- Interrupt Latency
- Virtual Memory
- Flash Memory
- Structure of an RTOS
- Spectrum of Real-Time Operating Systems

Today, we basically concept, first, what are the basic requirements of an RTOS, that means, a real-time operating system should support what kinds of things, which requirements. For example, interrupt latency, virtual memory, flash memory, et cetera. Then, we will discuss a structure of an real-time operating system, and we will discuss then the spectrum of real-time operating systems how the real-time operating systems can be categorized into different classes, and some of the popularly available time operating systems.

(Refer Slide Time: 01:22)

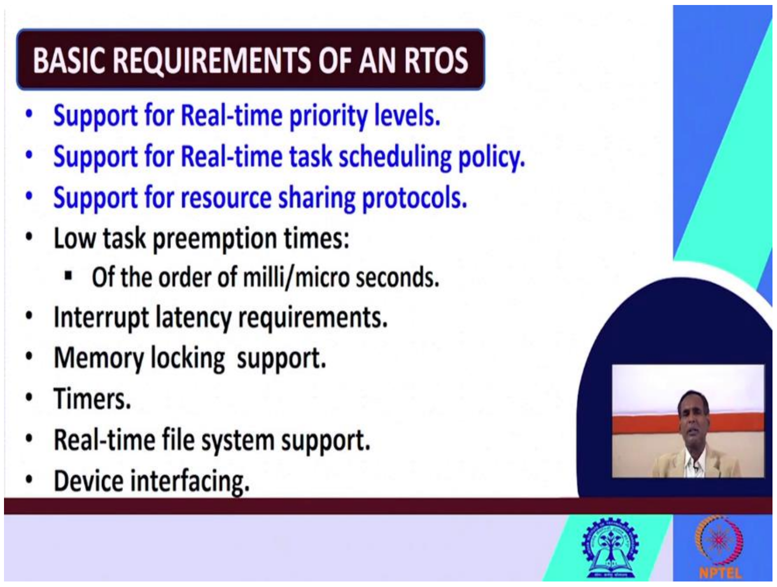


KEYWORDS

- Interrupt Latency
- Memory Protection & Locking
- Flash Memory
- Board Support Package (BSP)
- RTOS Spectrum

So, some of the key words you we will use are, what do you mean by interrupt latency? What do you mean by memory protection and memory locking? What is s flash memory? What is BSP, BSP stands for the Board Support Package. What is the spectrum of RTOS, et cetera, these things we will use?

(Refer Slide Time: 01:39)



BASIC REQUIREMENTS OF AN RTOS

- Support for Real-time priority levels.
- Support for Real-time task scheduling policy.
- Support for resource sharing protocols.
- Low task preemption times:
  - Of the order of milli/micro seconds.
- Interrupt latency requirements.
- Memory locking support.
- Timers.
- Real-time file system support.
- Device interfacing.

So, now, let us first start with the basic requirements an ROTS. I hope, you must have read a paper on operating systems in your earlier semesters. So, you might think of this operating system as a prerequisite for the any advanced operating system such as real-time operating system or distributed operating system or graphics operating system for any advanced real-time operating systems you must have to know the basic concepts of operating systems.

So, I requested, those who have not studied they are forgotten please have a look on the basics of the operating systems then it will be easier for you to understand the concepts of the real-time operating system. So, now let us first see, what should be the basic requirements upon a real-time operating system?

First, so the real-time operating system must support for real-time priority levels. I hope, in earlier classes you have known what do you mean by priority levels. How the priority levels can be assigned to the real-time tasks et cetera. So, this is one of the important thing every real-time operating system or support for real-time priority levels.

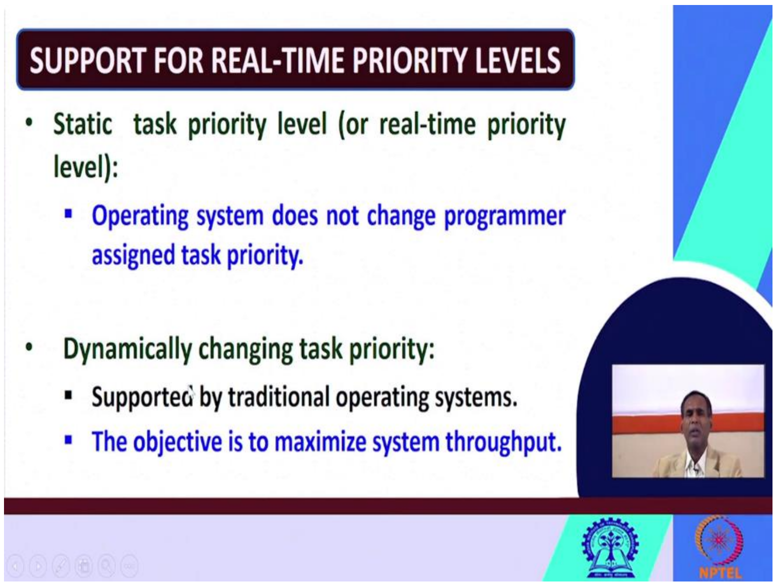
Similarly, every real-time operating system or support for what real-time task scheduling policy you have known dependent real-time task scheduling policies such as RMA, EDF et cetera. So, it must, the real-time operating system or support for the real-time task scheduling policy. Similarly, it must support for resource sharing protocols. Various resource sharing protocols, you have seen like PIP, HLP, PCP, et cetera. So, the operating system, the real-time operating system or support for these resource sharing protocols.

Similarly, the preemption times you might have heard about these normal operating system. So, for real-time operating system, it should have low task preemption times may be of the order of some million microseconds, it should not be more than that. And then let us see what should support the interrupt latency requirements. We will see, whether it should be lower or higher I will discuss in the subsequent slides. So, what is the interrupt latency requirements?

Then it should support the memory locking, it should support the better memory management. And since, the important component of real-time services is the real time. So, what should the time services? So, we must know about the time services that the RTOS must support for, we should know the timers, also, how the file systems can be supported.

Basically, it should be real-time file system support, so the RTOS must support real-time file system. And similarly, the real time operating system must support the device interfacing. So, these are some of the basic requirements for an operating system. Now, let us go one by one.

(Refer Slide Time: 04:25)



SUPPORT FOR REAL-TIME PRIORITY LEVELS

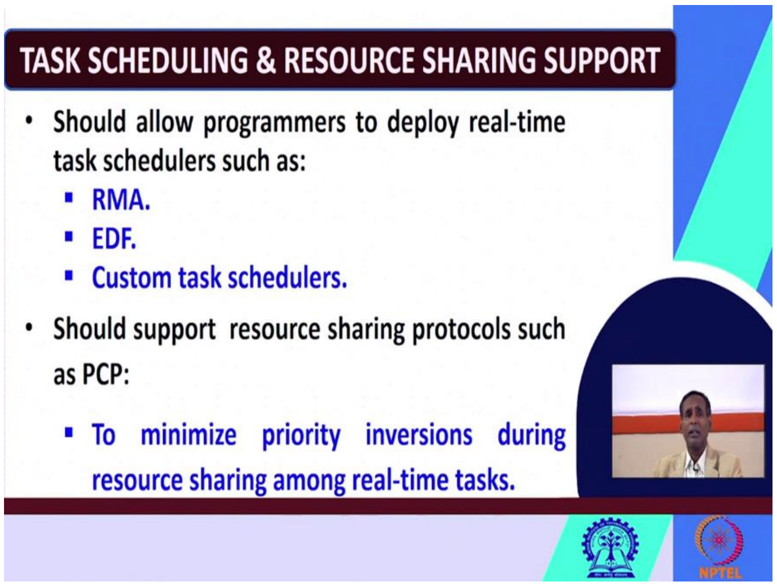
- Static task priority level (or real-time priority level):
  - Operating system does not change programmer assigned task priority.
- Dynamically changing task priority:
  - Supported by traditional operating systems.
  - The objective is to maximize system throughput.

First, we will see, the first requirement is support for real-time priority levels. So, you know traditionally about static task priority to level and dynamic task priority level. In static task product level, actually, this is explicitly made for the real-time priority level or this is meant for the real-time systems. So, the operating system, so in static task priority level, what is happening, the operating system, does not change the program assign task priority.

Once the programmer has assigned some priority to the task, the operating system will not change the priority. This is called a static task priority level, so, this is also known as real-time price level. Let us see, what happens in dynamically changing tax priority. So, as its names are just dynamic. So, here, once the programmer assigns a priority to the task, the operating system can change it.

So, that is why, this is known as dynamically changing tax priority. And this feature was supported in traditional operating systems, but for modern real-time operating systems we require static task writer level. So, why the traditional operating systems they were supporting this dynamically changing task priority, because, the objective was to maximize the system throughput, that is why the traditional operating systems they are supporting the dynamically changing tax priority.

(Refer Slide Time: 05:44)



TASK SCHEDULING & RESOURCE SHARING SUPPORT

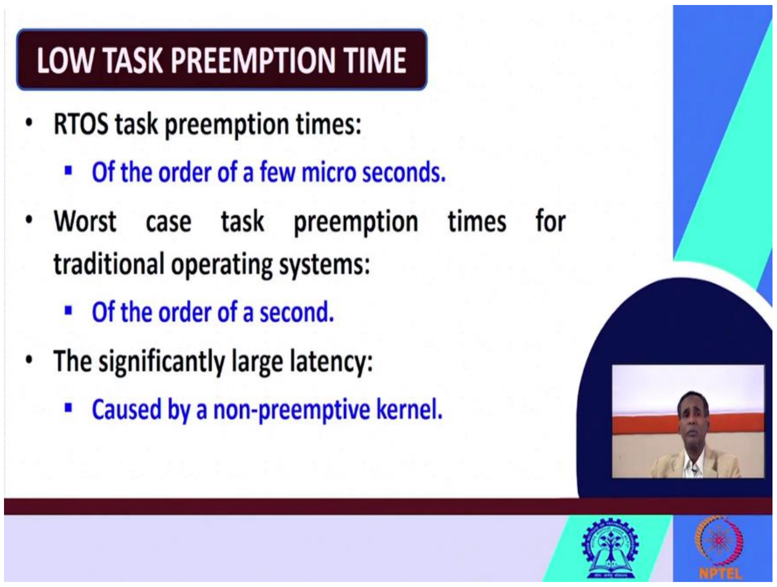
- Should allow programmers to deploy real-time task schedulers such as:
  - RMA.
  - EDF.
  - Custom task schedulers.
- Should support resource sharing protocols such as PCP:
  - To minimize priority inversions during resource sharing among real-time tasks.

Now, let us go to the next feature. I have already told you, real-time operating systems should support task scheduling, and they should support resource sharing worth services, resource sharing protocols. So, the real-time operating system must allow programmers to deploy real-time task schedulers, and examples of real time should last you have also already known. RMA Rate Monitoring Algorithm, EDF Earliest Deadline First or any customized task schedulers they should be supported by your real-time operating systems.

Similarly, your real time operating system must support resource sharing protocols such as PCP, that you have already known. So, why the real-time systems, operating systems must support these resource sharing protocols because they should support the resource sharing protocols in order to minimize the priority inversions during the resource sharing among the real-time task, so that is why the real-time operating system they should support PCP.

What is PIP, what is HLP, what is PCP, we have already known in the earlier classes, so we will not discuss here. Only you should note that, the real-time operating system should support these types of resource sharing protocols.

(Refer Slide Time: 06:52)



LOW TASK PREEMPTION TIME

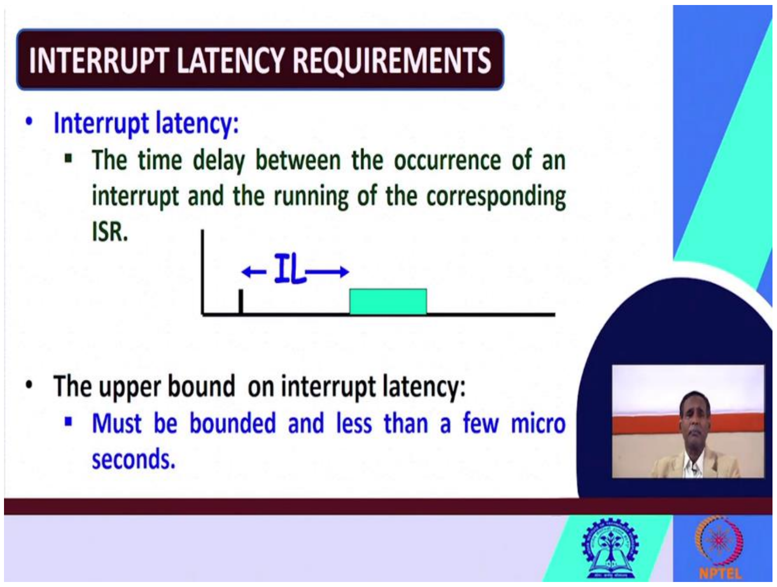
- RTOS task preemption times:
  - Of the order of a few micro seconds.
- Worst case task preemption times for traditional operating systems:
  - Of the order of a second.
- The significantly large latency:
  - Caused by a non-preemptive kernel.

Now, next, next requirement is preemption time. Now, let us say, what should be the task preemption time in the real-time operating systems? So, in real-time task preemption in real time systems the tax preemption time should be low, it should be very low. So, why? Let us say, the real time operating system task preemption time should be of the order of a few microseconds, that means, it should be very low.

So, the worst case task preemption times for traditional operating systems it was of the order of a second, it was a little bit high. In case of a traditional operating systems the worst case task preemption was of the order of a second, which was high, but in real-time operating system the preemption time should be low, it should be of the order of a few microseconds.

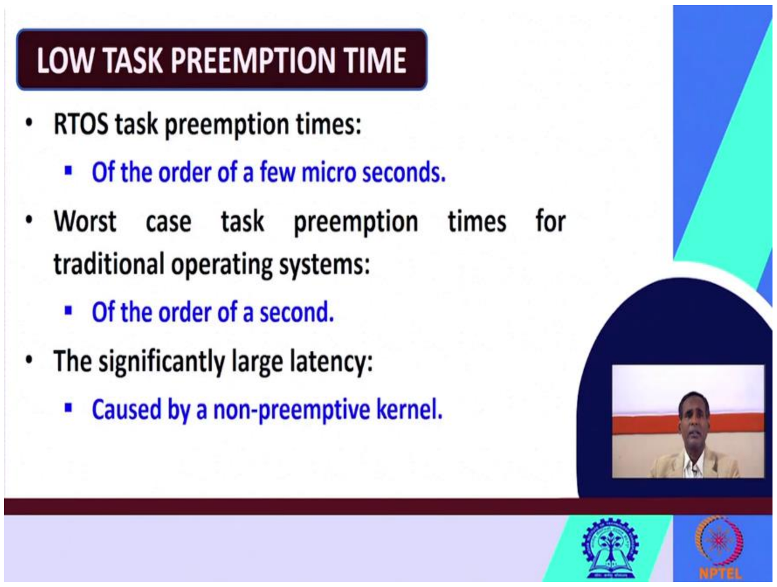
So, why in traditional operating systems there is a significantly larger latency? How? Why it was caused? The significantly larger latency, it was caused why, by a non-preemptive kernel. So, since we are using non-preemptive kernels, so that is why, the latency was a little bit large. The significantly larger latency was caused by using or due to use of a non-preemptive kernel.

(Refer Slide Time: 08:06)



INTERRUPT LATENCY REQUIREMENTS

- Interrupt latency:
  - The time delay between the occurrence of an interrupt and the running of the corresponding ISR.
- The upper bound on interrupt latency:
  - Must be bounded and less than a few micro seconds.



LOW TASK PREEMPTION TIME

- RTOS task preemption times:
  - Of the order of a few micro seconds.
- Worst case task preemption times for traditional operating systems:
  - Of the order of a second.
- The significantly large latency:
  - Caused by a non-preemptive kernel.

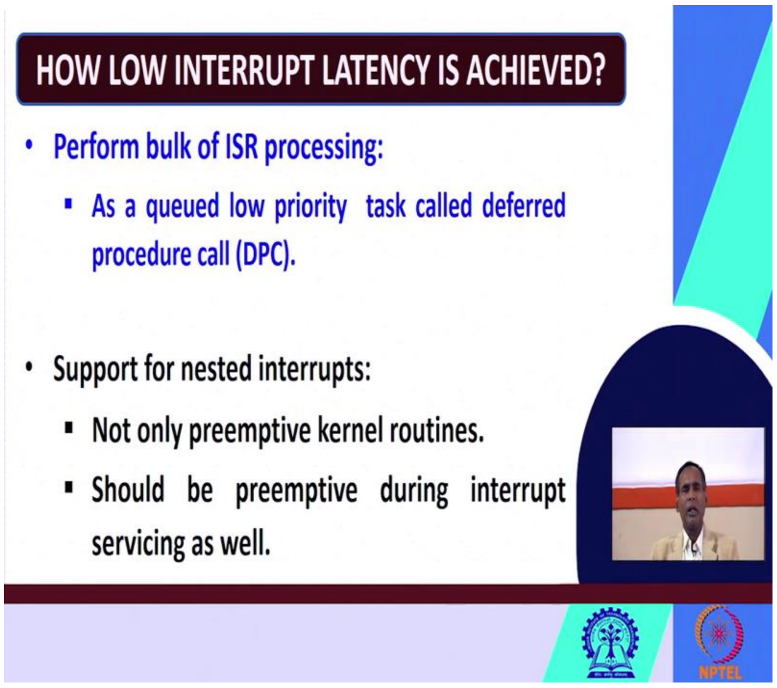
Now, let us say, the interrupt latency requirements. What do you mean by interrupt latency? First, let us understand. So, interrupt latency means, it is the time delay between the occurrence of one interrupt and running up the corresponding ISR. What is ISR? Interrupt service routine. So, the time delay between the occurrence of an interrupt.

I hope, you have always known, what is an interrupt, what is ISR in the traditional operating systems. Only I am extending to real-time operating system. The time delay between the occurrence of an interrupt and the running up, the corresponding interrupt service routine is called interrupt latency. You can show it graphically like this. So, interrupt latency can be shown graphical like this. The time delay between the occurrence of an interrupt and.

So, suppose, here, the interrupt has occurred, and when the corresponding ISR has started. Say, at this point the corresponding interrupt service routine has been started running. So, this period, this gap is known as interrupt latency. What should be the upper bound of the interrupt latency? The upper bound on interrupt latency must be bounded, and less than a few microseconds.

I have already told you in the earlier slide that this. This is reading preemption time. So, but regarding latency requirement it should be low. The upper bound, in case of real-time operating systems the upper bound on interrupt latency it should be, it must be bounded, and it should be and it must be less than a few microseconds.

(Refer Slide Time: 09:35)



HOW LOW INTERRUPT LATENCY IS ACHIEVED?

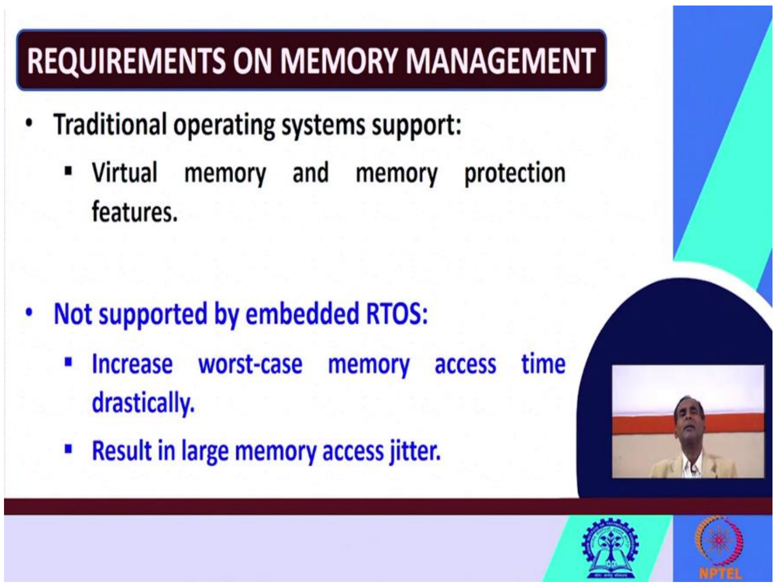
- Perform bulk of ISR processing:
  - As a queued low priority task called deferred procedure call (DPC).
- Support for nested interrupts:
  - Not only preemptive kernel routines.
  - Should be preemptive during interrupt servicing as well.

Then, how it can be achieved? How low interrupt latency can be achieved? So, in order to achieve low interrupt latency, what do you do, perform a significant chunk of ISR or perform a backup ISR processing, how, as a queued low priority tasks, which is called the deferred procedure caller DPC. I think, DPC you must have known earlier in operating systems.

So, in order to achieve low interrupt latency, what do you do, perform the backup ISR processing or perform a backup ISR processing or perform a significant chunk of ISR processing by as a queued lower priority task, which we call our default procedure call or DPC.

Then this, RTOS must support for nested interrupts. Not only the preemptive kernel routine. So, we know that in this RTOS actually, we are using this preemptive kernels, so the real-time operating systems should support for nested interrupts, not only, the preemptive kernel routines, it should be preemptive interrupt servicing as well. So, it should support for nested interrupts. Not only for this preemptable kernel routines, also should be preemptive during the interrupt servicing, as well.

(Refer Slide Time: 10:50)



REQUIREMENTS ON MEMORY MANAGEMENT

- Traditional operating systems support:
  - Virtual memory and memory protection features.
- Not supported by embedded RTOS:
  - Increase worst-case memory access time drastically.
  - Result in large memory access jitter.

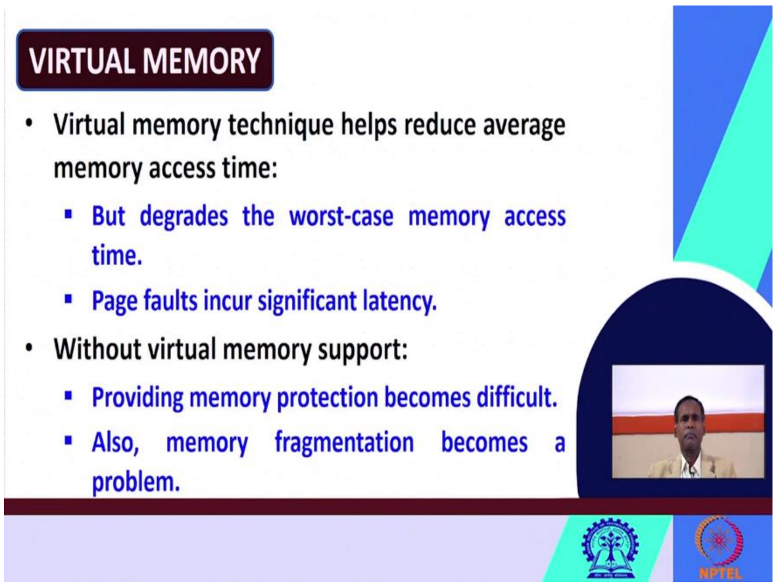
Now, let us see, what are the requirements of memory management. What kind of things should be supported by RTOS on memory management? Let us see what happens first in the traditional operating system. In traditional operating system the, we are using what, the virtual memory and memory protection schemes, is it not it.

So, traditional operating systems, they support virtual memory, and memory protection features. But in embedded real-time operating systems you will see these features, such as, virtual memory and memory protection features, these are not supported by embedded real-time operating systems. Why? Because these features that is virtual memory and memory protection features, they increase the worst-case memory access time drastically, and they also result in larger memory access dealer.

They will result in what? They will result in large or very large memory access jitter. What is jitter? Jitter you can say roughly it is delay. You have known already what a jitter may. What do you mean by jitter, you have known earlier. Roughly, you can say jitter means, delay. So, these are the two reasons why virtual memory and memory protection features these are not supported by the embedded real-time operating systems.

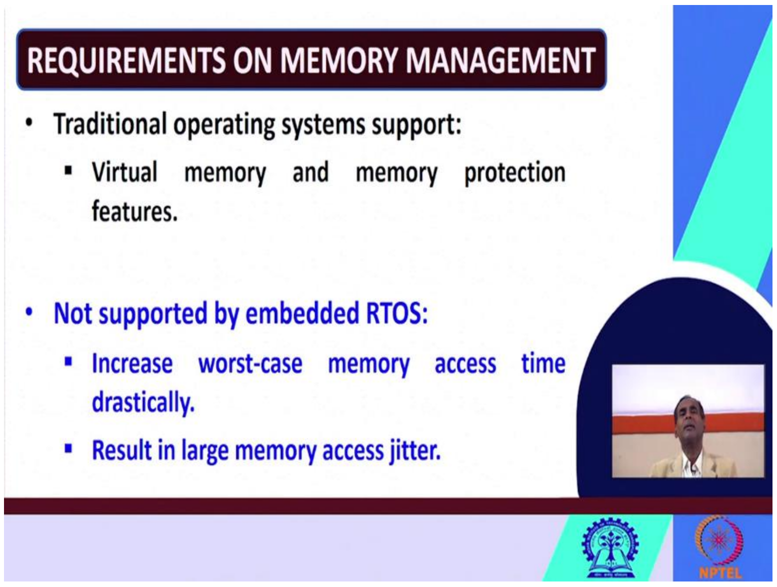
So, let us say, since in these traditional operating systems, they use virtual memory and memory protection features, and they are not supported by embedded RTOS. Let us first see, what do you mean by virtual memory?

(Refer Slide Time: 12:13)



VIRTUAL MEMORY

- Virtual memory technique helps reduce average memory access time:
  - But degrades the worst-case memory access time.
  - Page faults incur significant latency.
- Without virtual memory support:
  - Providing memory protection becomes difficult.
  - Also, memory fragmentation becomes a problem.



REQUIREMENTS ON MEMORY MANAGEMENT

- Traditional operating systems support:
  - Virtual memory and memory protection features.
- Not supported by embedded RTOS:
  - Increase worst-case memory access time drastically.
  - Result in large memory access jitter.

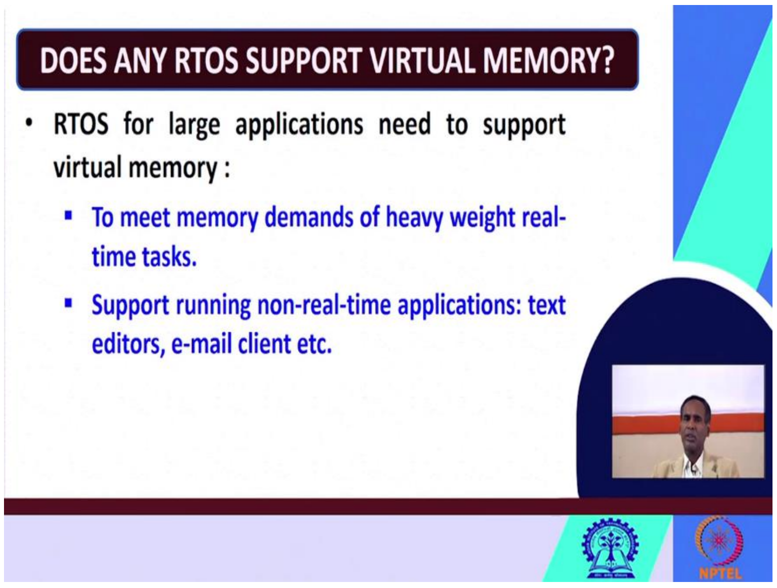
So, virtual memory means, this virtual memory technique it helps reduce the average memory access time. The virtual memory, why it was used in the traditional operating systems? Virtual memory techniques, they help reduce the average memory access time that is why they are used in traditional operating systems.

Since virtual memory techniques, they help reduce average memory access time that is why they were using traditional operating systems. But the drawback is that, the virtual memory it degrades the worst-case memory access time, that is one of the drawbacks. Perhaps, this is one and the second drawback is that, a page faults incur significant latency. Several page faults will occur, these page faults they incur significant latency.

Due to these two reasons, perhaps, so that is why virtual memory is not used in embedded real time operating systems. So, now, let us say, without virtual memory support what will happen? Without virtual memory support providing memory protection becomes difficult that you have already known in traditional operating systems. So, without virtual memory support, providing the memory protection it becomes much more difficult. Also, the memory fragment becomes a problem.

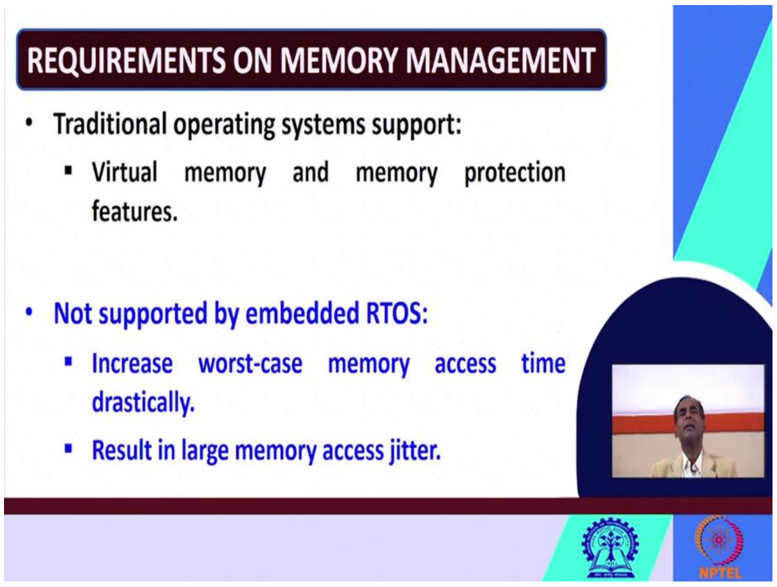
I hope, you have already known memory fragmentation in traditional operating system. So, without virtual memory support the memory fragmentation, it also becomes a problem. So, these are some of the problems that we may face if we will not use, or since virtual memory is not used in this embedded real-time operating systems.

(Refer Slide Time: 13:43)



DOES ANY RTOS SUPPORT VIRTUAL MEMORY?

- RTOS for large applications need to support virtual memory :
  - To meet memory demands of heavy weight real-time tasks.
  - Support running non-real-time applications: text editors, e-mail client etc.



REQUIREMENTS ON MEMORY MANAGEMENT

- Traditional operating systems support:
  - Virtual memory and memory protection features.
- Not supported by embedded RTOS:
  - Increase worst-case memory access time drastically.
  - Result in large memory access jitter.

Now, I will ask the question. Does any real-time operating system virtual memory? Because I have told you that the embedded what real-time operating systems, they do not support virtual memory. But still the question is, does any real time operating system support virtual memory? And the answer is, yes. Which kind of real-time operating system they support virtual memory?

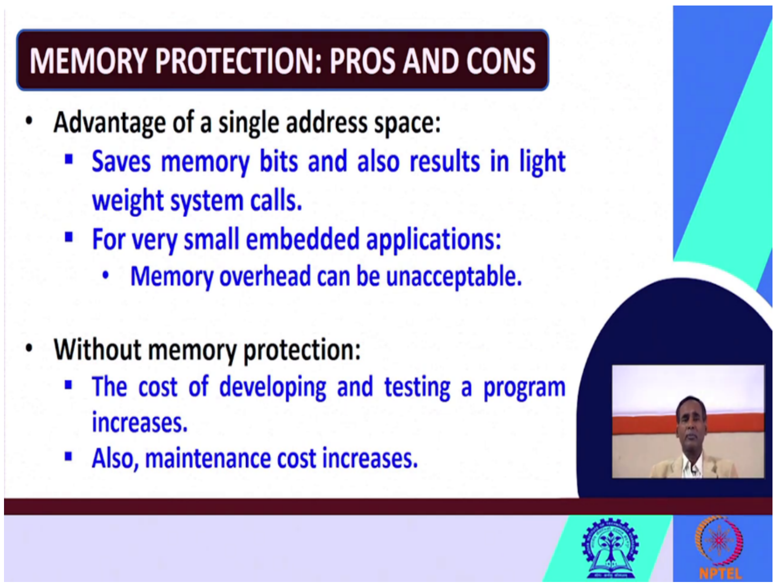
Let us say, the real-time operating system for large applications they need to support virtual memory. If you are developing some real-time system for very large applications, they need to support virtual memory. Why? What will be the problem, if they will not support the virtual memory? Because of the following two reasons.

So, the real-time system for large applications, they should support virtual memory in order to meet the memory demands of the heavyweight real-time tasks. So, what will be the memory demand? I hope, you have already known lightweight real-time tasks and heavyweight real-time tasks. So, in order to meet the memory demands, the memory requirements of heavyweight real-time tasks, the real-time operating systems for large applications, they should support virtual memory.

Similarly, another region is that, support this virtual memory, they support running non-real time applications such as a text-editors, email clients et cetera. So, if you are developing a real-time system for large applications, they might require what running non-real time applications such as text-editors an email client et cetera.

So, in order to support the running up non-real time applications such as text editors and email clients et cetera, the real-time systems for large applications must support virtual memory. So, these are some of the reasons, why the large application, the real-time operating system for large applications, they should support virtual memory.

(Refer Slide Time: 15:30)



MEMORY PROTECTION: PROS AND CONS

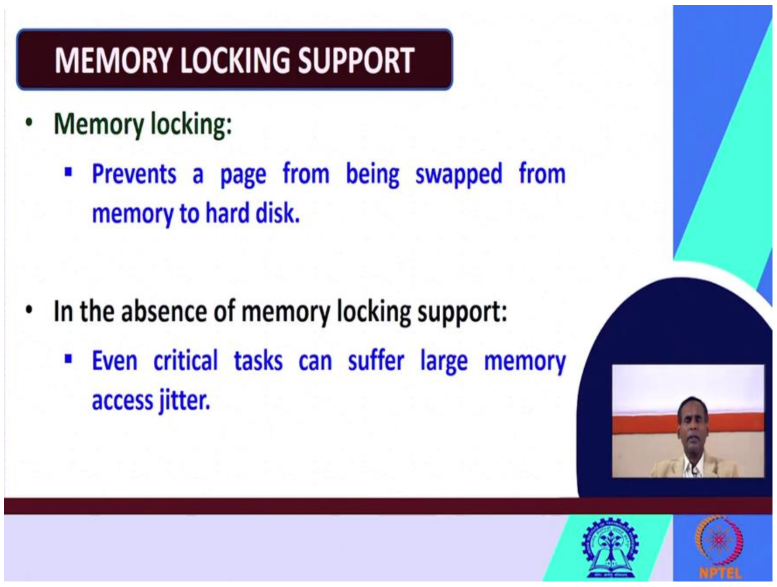
- Advantage of a single address space:
  - Saves memory bits and also results in light weight system calls.
  - For very small embedded applications:
    - Memory overhead can be unacceptable.
- Without memory protection:
  - The cost of developing and testing a program increases.
  - Also, maintenance cost increases.

Now, let us go to the memory protection aspects of these real-time operating systems, what are the pros and cons. So, advantages of a single address space are as follows. So, if you will use a single address space, it saves the memory bits, and also, results in light weight system calls. We require the light weight system call. So, if you will use single address space, it saves memory bits and also it results in a light weight system calls.

For very small embedded applications the memory overhead can be unacceptable. But if we will use this single address space. So, for very small embedded applications memory overhead can be unacceptable. Without memory protection what will happen? Without memory protection, the cost of developing and testing a program increases.

So, it will not use memory protection then the cost of developing and testing a program will increase tremendously. Also, the maintenance cost increases. If we will not use any memory protection scheme, the maintenance cost of the software, the maintenance cost of the real-time operating system or software it will increase.

(Refer Slide Time: 16:39)



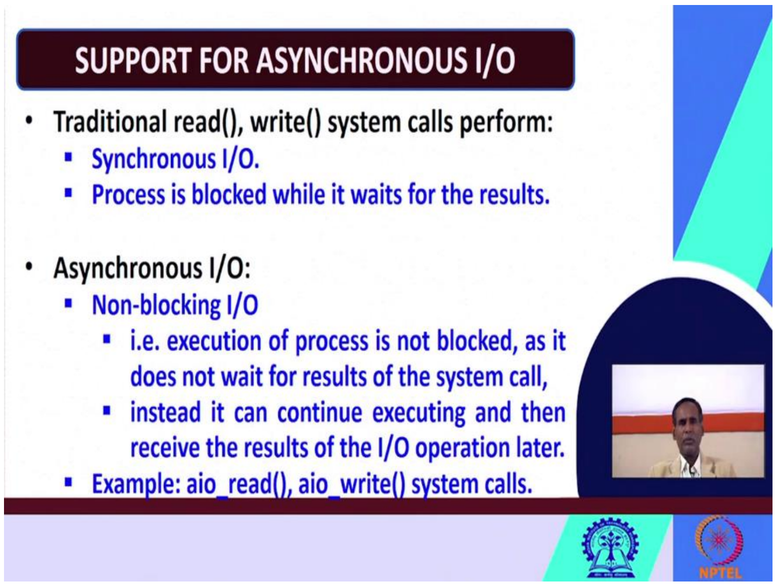
MEMORY LOCKING SUPPORT

- Memory locking:
  - Prevents a page from being swapped from memory to hard disk.
- In the absence of memory locking support:
  - Even critical tasks can suffer large memory access jitter.

Now let us go to the next aspect, that is, memory locking support. I hope, memory locking you have already seen in case of traditional operating systems. So, in memory locking, what does it do? Why it is used? The memory locking, it prevents a page from being swapped from memory to hard disk. So, if you are using memory locking, then you can prevent a page from being swapped from where, from memory to hard disk, that is why what this memory locking is there.

In the absence of the memory locking support what will happen? In the absence of memory locking support even critical tasks can suffer large memory access jitter. So, if you do not have memory locking support in your operating system, then, even these critical tasks, they may suffer from a larger memory access jitter. So, that is why that memory locking what facility should be supported by real-time operating systems.

(Refer Slide Time: 17:33)



SUPPORT FOR ASYNCHRONOUS I/O

- Traditional read(), write() system calls perform:
  - Synchronous I/O.
  - Process is blocked while it waits for the results.
- Asynchronous I/O:
  - Non-blocking I/O
    - i.e. execution of process is not blocked, as it does not wait for results of the system call,
    - instead it can continue executing and then receive the results of the I/O operation later.
  - Example: aio_read(), aio_write() system calls.

Now, let us say, the support for asynchronous I/O. I hope, you know two kinds of I/O operations, synchronous I/O operations and asynchronous I/O operations. Those things you must have studied in the normal operating system. The traditional system calls such read and write they perform synchronous I/O operations. So, the traditional system call, such as, read write et cetera they perform synchronous I/O.

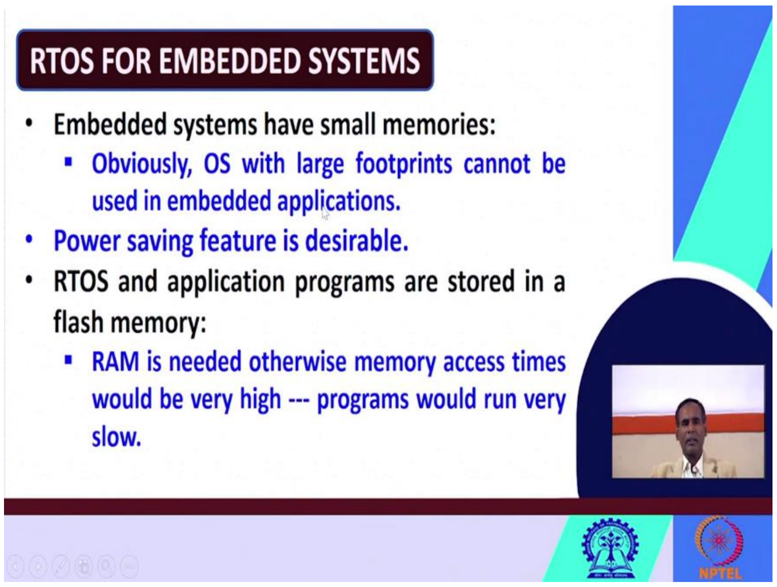
I hope, you know the difference between synchronous and asynchronous I/O in case of your normal operating system. Those who have forgotten, please again look at them. So, these traditional read and write system calls. When you are using the traditional read and write system calls here the process is blocked highly it waits for the results.

The process that is running it is blocked while it is waiting for the result. This is what is happening in synchronous, what, I/O operations. But in asynchronous I/O operations, what, these are, so in synchronous the process is blocked, but in asynchronous I/O the processes are non-blocked, they are non-blocking I/O.

What do you mean by non-blocking I/O? That is, the execution of a process it is not blocked, as it is happening in the case of the traditional systems or when you are using the read/write system calls in the traditional operating systems. So, in contrast in a synchronous I/O they are non-blocking, that is, execution of a process it is not blocked as it does not wait for the results of the system call. Since it does not wait for the results of the system call the execution process is not blocked.

Instead, what it does? Instead, it can continue or instead, it continues executing, and then, it receives the results of the I/O operation later on when they are available. So, examples of asynchronous I/O. In synchronous, you are using just read and write, in a synchronous I/O the examples are aio. So, aio stands for synchronous I/O read and asynchronous I/O write, these are the system calls, these are examples of asynchronous I/O. So, your real-time operating system must support for these asynchronous operations.

(Refer Slide Time: 19:45)



RTOS FOR EMBEDDED SYSTEMS

- Embedded systems have small memories:
  - Obviously, OS with large footprints cannot be used in embedded applications.
- Power saving feature is desirable.
- RTOS and application programs are stored in a flash memory:
  - RAM is needed otherwise memory access times would be very high --- programs would run very slow.

Now, let us quickly look at the real time operating system for embedded systems. So, embedded systems they have small memories, is it not it. The embedded systems you are using, they are, normally, they have small memories. Why? Because the operating system with large footprints cannot be used in embedded applications.

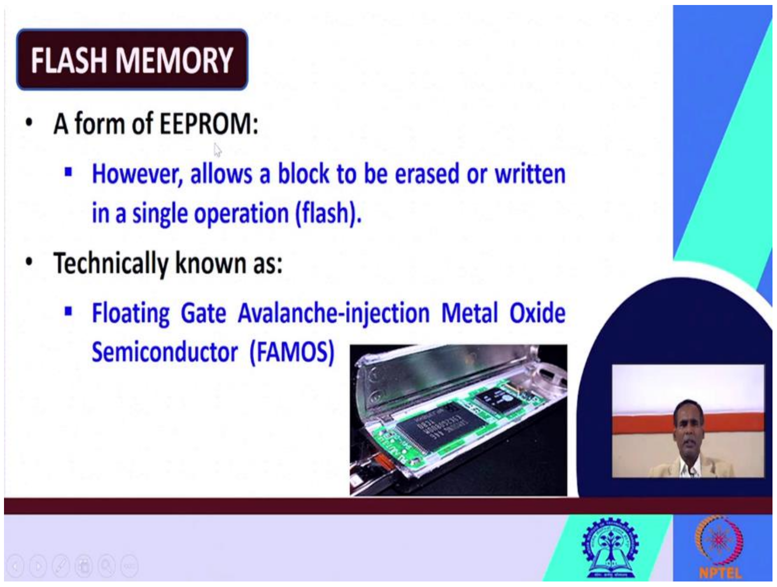
Take the example of remember your mobile phone. Your mobile phone is a very good example of embedded real-time system. And if your embedded that for your mobile phone you know how much memory space you are using their mobile phone. If it will be of very high capacity then what will the problem?

So, normally, the operating system with large footprints, they cannot be used in the embedded application. So, that is why, the embedded systems, they have small memories. And also, power saving feature is desirable in what the real-time system, real-time operating systems for embedded systems, the memory should be less as well as there should be power saving feature, it is desirable. Real-time operating system and application programs are stored in a flash memory.

So, normally, the real-time operating system and the application programs, they are stored in a flash memory, still, RA M is needed otherwise, the memory access time would be very much high. And you know, if the memory access times would be very high then what will the consequence? The programs would run very slow.

So that is why, what RAM is also needed otherwise the memory access times would be very high and the programs would run very slow. Now, as I have already told you. RTOS and the application programs, they are stored in a flash memory. We should know how to do flash member. Let us look at quickly what is your flash memory.

(Refer Slide Time: 21:27)



FLASH MEMORY

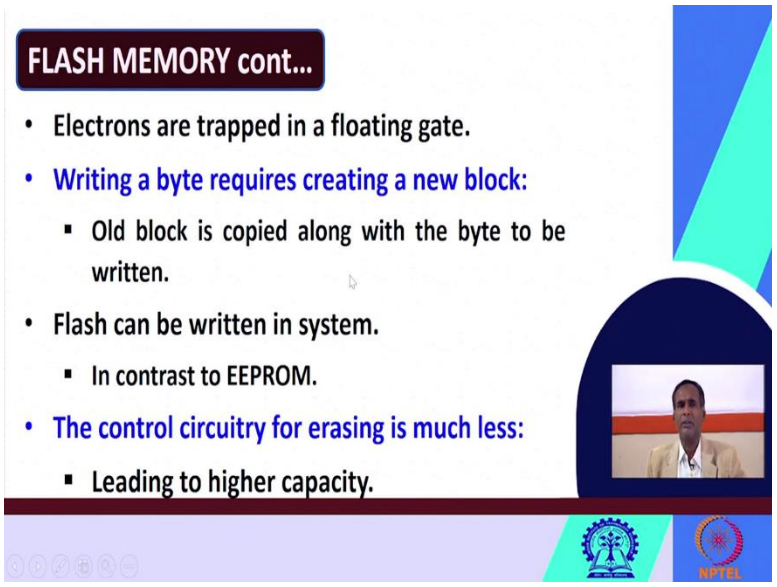
- A form of EEPROM:
  - However, allows a block to be erased or written in a single operation (flash).
- Technically known as:
  - Floating Gate Avalanche-injection Metal Oxide Semiconductor (FAMOS)

So, flash memory is a form of EEPROM. I hope, you know this hierarchy of the read-only memory, ROM then there is a read-only memory then PROM programmable read-only memory, then EEPROM is it not it, erasable programmable read-only memory then EEPROM or we call as E square PROM. Electrically erasable programmable read-only memory.

So, this flash memory, these are basics of what memories you must have read somewhere else earlier. So, this flash memory, it is a form of EEPROM. However, what is the difference between class flash and EEPROM? So, it is a form of EEPROM, but it allows a block to be erased or retain in a single operation or a single flash, that is why it is known as flash memory.

So, here, this memory flash it allows a blog to be erased or written in a single operation as a single flash that is why this is known as flash memory. So, this flash memory technically this is known as Floating Gate Avalanche-Injection Metal Oxide Semiconductor or FAMOS. So, this is a technically known as FAMOS.

(Refer Slide Time: 22:35)



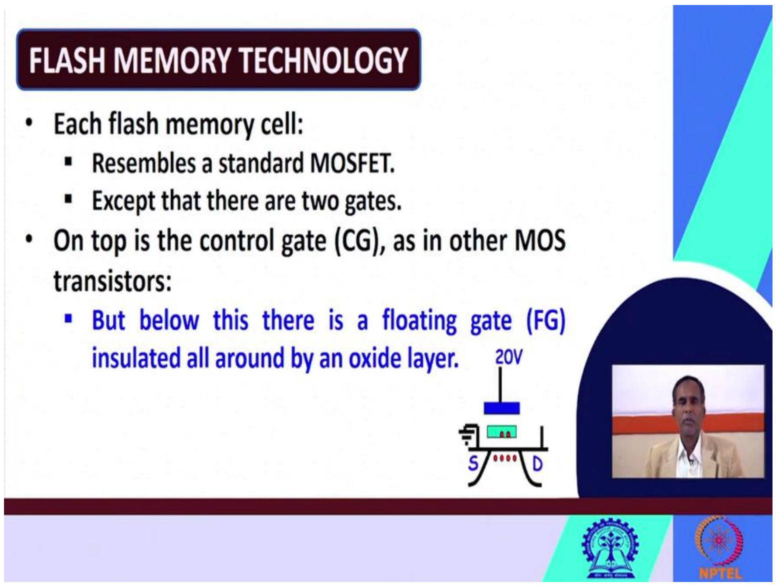
FLASH MEMORY cont...

- Electrons are trapped in a floating gate.
- Writing a byte requires creating a new block:
  - Old block is copied along with the byte to be written.
- Flash can be written in system.
  - In contrast to EEPROM.
- The control circuitry for erasing is much less:
  - Leading to higher capacity.

So, here in flash memory, the electrons are trapped in a floating gate. Writing a byte requires what, creating a new block. If you want to write a byte, then writing a byte requires creating a new block. The old block is copied along with the byte to be written. So, here, the old block that is copied along with the byte to be written, so the flash can be written in a system.

Here, the flash can be written in a system in contrast to E square PROM. So, in E square PROM this was not possible, but in flash memory, flash can be written in this system. The control circuitry for erasing is much less in comparison to EEPROM leading to higher capacity. So, these are some of the basic concepts of flash memory, other details you may see yourself from any other sources.

(Refer Slide Time: 23:23)



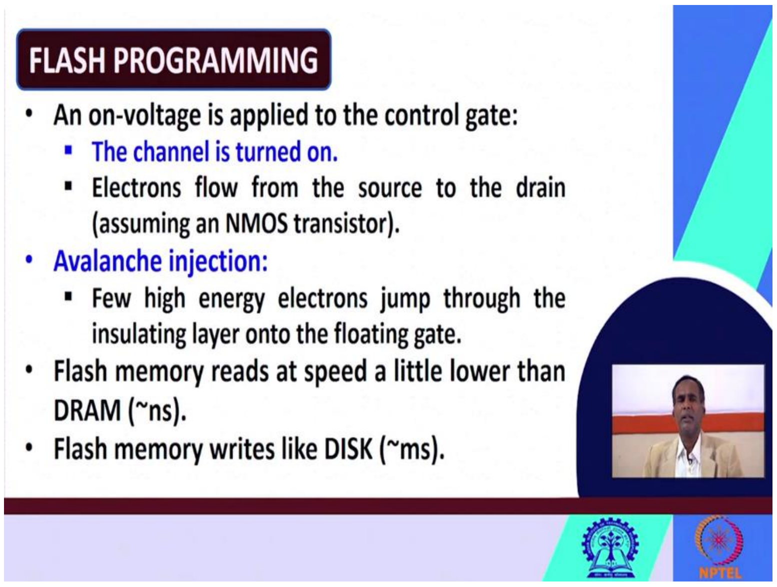
FLASH MEMORY TECHNOLOGY

- Each flash memory cell:
  - Resembles a standard MOSFET.
  - Except that there are two gates.
- On top is the control gate (CG), as in other MOS transistors:
  - But below this there is a floating gate (FG) insulated all around by an oxide layer.

Now, let us look at quickly flash memory technology. Even if what, this is very fundamental, I thought, this will be interesting. You must know, what do you mean by flash memory? What is the technology used in flash memory? So, let us quickly look at the flash memory technology. Each flash memory cell, it resembles just like a standard MOSFET transistor. So, except that here there are two gates.

So, on the top, there is one gate, and on the bottom, there is another gate. On top is the control gate, we call a CG as in other MOS transistors, but below, this there is a floating gate, and with insulated all around by an oxide layer. So, please see, how this flash memory it is what being built. So, this is a simple diagram showing you about this flash memory.

(Refer Slide Time: 24:14)



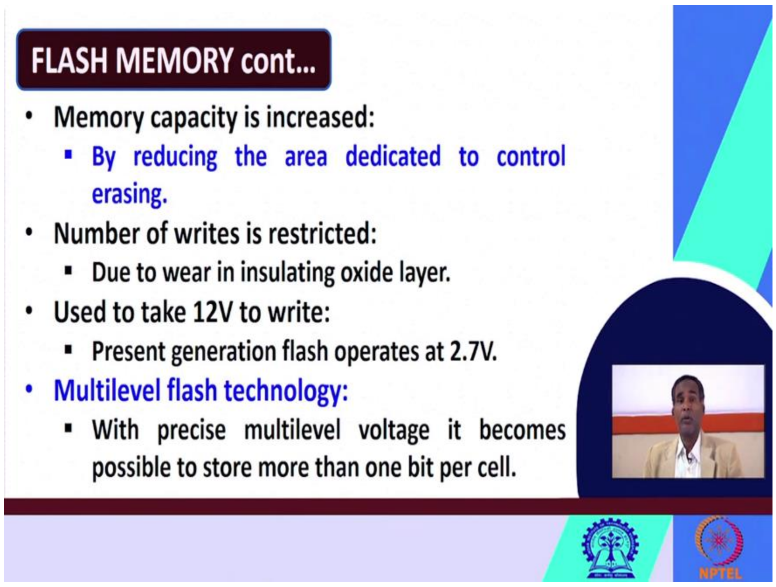
FLASH PROGRAMMING

- An on-voltage is applied to the control gate:
  - The channel is turned on.
  - Electrons flow from the source to the drain (assuming an NMOS transistor).
- Avalanche injection:
  - Few high energy electrons jump through the insulating layer onto the floating gate.
- Flash memory reads at speed a little lower than DRAM (~ns).
- Flash memory writes like DISK (~ms).

So, let's go to look what about the flash programming. So, in flash programming an on-voltage is applied to the control gate, the channel is turned on, then the electrons flow from the source to the drain assuming that an NMOS transistor is used. So, regarding the avalanche-injection who can say the followings.

So, here, a few high energy electrons, they jump through the insulating layer onto to the floating gate. So, the flash memory it reads at a speed a little bit lower than the DRAM, Dynamic RAM may be of the order of nanoseconds, and flash memory writes. So, it reads at a speed of little lower than on DRAM, that is, at the order of nanosecond, whereas, flash memory it writes like DISK, which is of the maybe milliseconds.

(Refer Slide Time: 25:07)



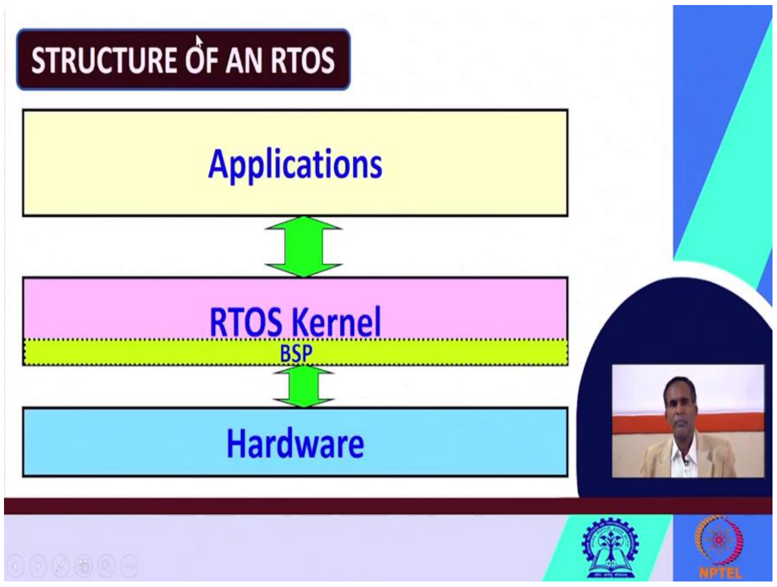
FLASH MEMORY cont…

- Memory capacity is increased:
  - By reducing the area dedicated to control erasing.
- Number of writes is restricted:
  - Due to wear in insulating oxide layer.
- Used to take 12V to write:
  - Present generation flash operates at 2.7V.
- Multilevel flash technology:
  - With precise multilevel voltage it becomes possible to store more than one bit per cell.

Memory capacity it is increased. How? Memory capacity is increased by reducing the area dedicated to control erasing. The number of writes is restricted. Why? The number of writes is restricted due to what we are in insulating the oxide layer. So, in flash memory, it used to take 12-volts to write. Flash memory used to take 12-volts to write, present generation Flash they operate at 2.7 volt.

So, let us see one more technological logic multi-level flash technology. With precise multi-level voltage, what happens, it becomes possible to store more than bit per cell. So, if required to store more than bit per cell, then what this precise multi-level voltage you can use, and with the use of precise multi-level voltage it becomes possible to store more than one bit per cell. So, this is regarding little bit fundamental of flash memory. You can read the more details from any other sources.

(Refer Slide Time: 26:10)


STRUCTURE OF AN RTOS

Applications

RTOS Kernel

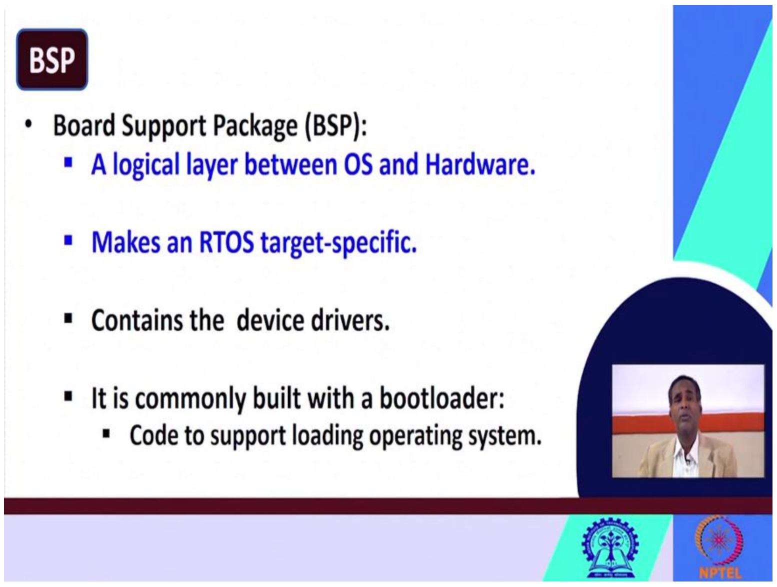
BSP

Hardware

Now, let us quickly look at to what the other important aspect, that is, the structure upon an RTOS. Normally, this is a typical structure of a real-time operating system. The bottom you know, the hardware level is there and then the top, the applications we are interacting with the applications. Then below the applications level, the kernel, they are just like in other operating systems there is a kernel.

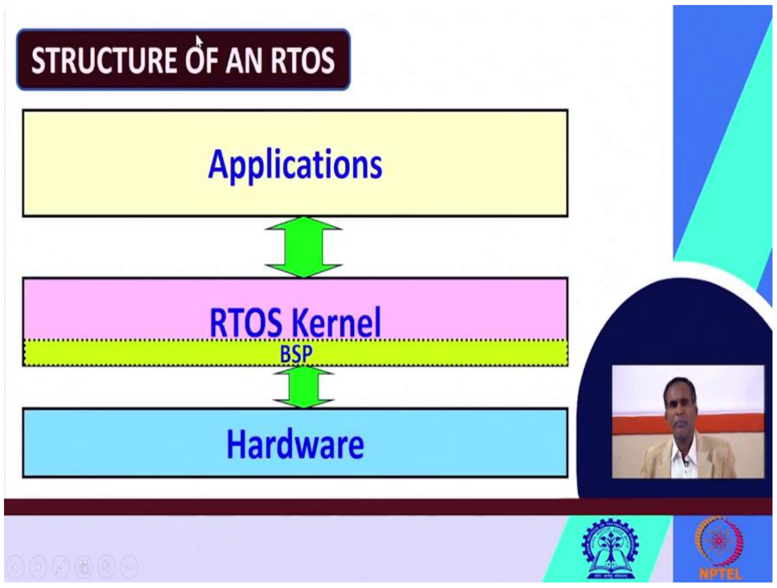
Here also below the application there is a kernel called as RTOS kernel, real-time operating system kernel. So, in between the real-time operating systems, kernel layer and hardware layer there is another layer called logic BSP. So, you know the basics of what is hardware, what is the RTOS kernel and what is applications. We will only discuss here what do you mean by BSP?

(Refer Slide Time: 26:53)



BSP

- Board Support Package (BSP):
  - A logical layer between OS and Hardware.
  - Makes an RTOS target-specific.
  - Contains the device drivers.
  - It is commonly built with a bootloader:
    - Code to support loading operating system.



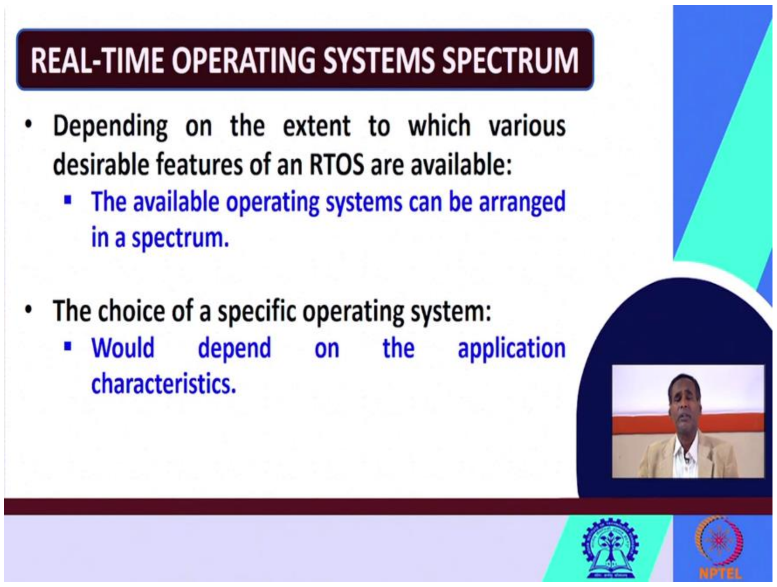
STRUCTURE OF AN RTOS

BSP stands for Board Support Package, this is your logical layer. This is not a physical, so you can see that is why we have given what like this what structure. So, this is a logical layer present in between the operating system, is it not it, between the operating system kernel on this hardware.

So, this package, Board Support Package BSP, it makes an RTOS target-specific. So, due to the presence of the BSP, it makes a real-time operating system target-specific, then this BSP also contains the device drivers. We require several device drivers. I have already told you one of the features that is required for real-time operating system is handling device drivers.

So, this BSP, this contents the device drivers. It is commonly built with a bootloader. So, this BSP it is commonly built with a bootloader, that is, with some code to support the loading operating system. So, bootloader et cetera, you had known earlier. So, this BSP it is commonly built with a bootloader with code to support the loading operating system. This is a little bit fundamental about the BSP.

(Refer Slide Time: 28:05)



REAL-TIME OPERATING SYSTEMS SPECTRUM

- Depending on the extent to which various desirable features of an RTOS are available:
  - The available operating systems can be arranged in a spectrum.
- The choice of a specific operating system:
  - Would depend on the application characteristics.

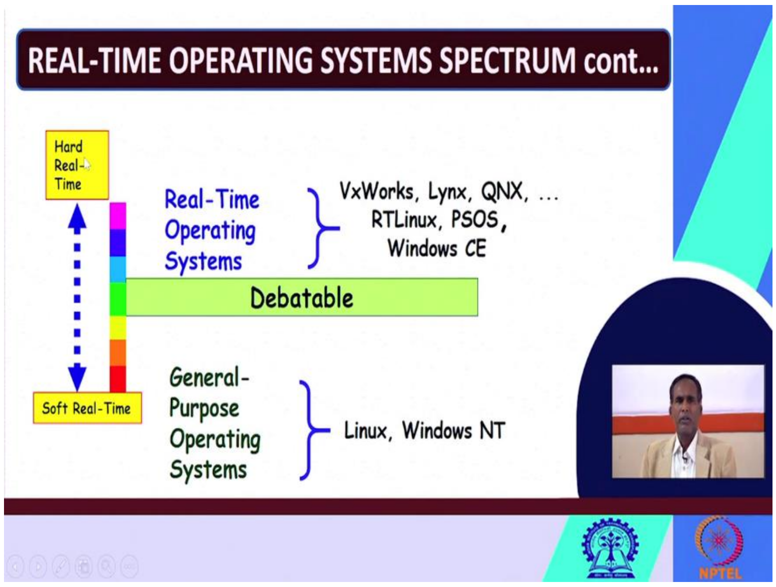
Now, let us quickly look at the last component of today's discussion, that is, the spectrum for real-time operating systems. What is this spectrum? Let us quickly look at the spectrum for real-time operating systems. So, depending on the extent to which various desirable features of an RTOS are available, the available operating systems can be added in the spectrum.

I have already told you several, what required features of real-time operating systems such are those priority levels and it should support, what is resource sharing. It should support, schedulers and then latency time, preemption time et cetera, memory management. These are the requirements I have already told you.

So, depending on the context to which various desirable features of an real-time operating system, whether they are available to what extent in any real-time operating system, the available real-time operating systems can be arranged just like in the spectrum. Then, how to select? Which operating system will be suitable for you? The choice of a specific operating system would depend on the application characteristics.

So, what is the, what are the characteristics of your application? So, depending on the application characteristics, you can choose or the choice of a specific operating system can be met. So, let us see, what is the spectrum for the real-time operating systems

(Refer Slide Time: 29:24)



REAL-TIME OPERATING SYSTEMS SPECTRUM cont…

This is the real-time operating system spectrum. So, you know about hard real-time tasks and soft real-time talks earlier. So, now, there are different real-time systems. Broadly, if you see at the bottom level, we can say them are the general purpose operating systems, and at the top level we, will see that, we call them as real-time operating systems. So, we are migrating from top to bottom.

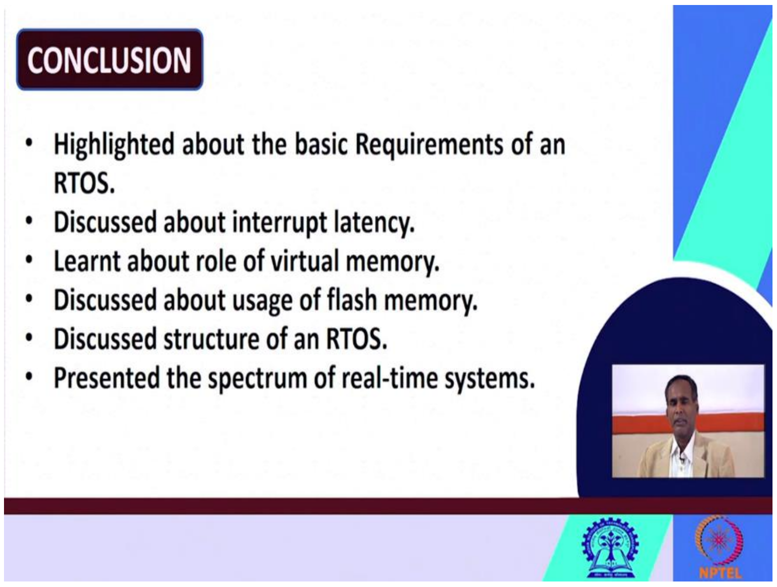
If you will see, in top level, we are dealing with the hard real-time task or this, we are dealing with hard real-time task. And while we are gradually coming down, we are dealing with the soft real-time task. I hope, you have already known hard real-time task and soft real-time task. And so, on the middle, so the top portion of the spectrum we call them as real-time operating systems, and at the bottom side of the spectrum, we call them are the general purpose operating systems.

So, examples of general-purpose operating systems are like, Linux, Windows NT, et cetera which can be used also as some of the real-time operating system. They have, they may have some features of real-time operating systems, but the upper ones they are specifically meant for real-time systems. These are popularly known as real-time operating systems.

The examples are like VxWorks, Lynx, QNX, RTLIinux, PSOS, Windows CE, et cetera, these are real-time operating systems, whereas, Linux, Windows NT et cetera these are general-purpose operating systems. So, in between these real-time operating systems and the general-purpose operating system, so some gap here, there.

So, this also, the operating system present in these layers, they are actual debatable whether you will call them real-time or not that is debatable, but this is the broader spectrum of the real-time operating systems. In the latter part of this classes we will discuss at least one, one or at least what few popular real-time operating systems available in the market.

(Refer Slide Time: 31:27)



CONCLUSION

- Highlighted about the basic Requirements of an RTOS.
- Discussed about interrupt latency.
- Learnt about role of virtual memory.
- Discussed about usage of flash memory.
- Discussed structure of an RTOS.
- Presented the spectrum of real-time systems.

REFERENCES

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

So, today, we have discussed the basic requirements of an real-time operating system, we have discussed, some of the basic requirements which are interrupt latency, virtual memory your memory locking, flash memory et cetera, you have seen. We have also discussed what is the basic structure of an RTOS. We have also presented the spectrum of the real-time systems, starting from general-purpose operating systems up to the real-time operating systems.

So, these are the things, we have seen. So, one more important thing we have left, that is one more important requirement is time services that should be provided by real-time operating systems. And the most important component is timer, clock resolution, so those are coming under time services, so that requirement we will discuss in the next class. Thank you very much.