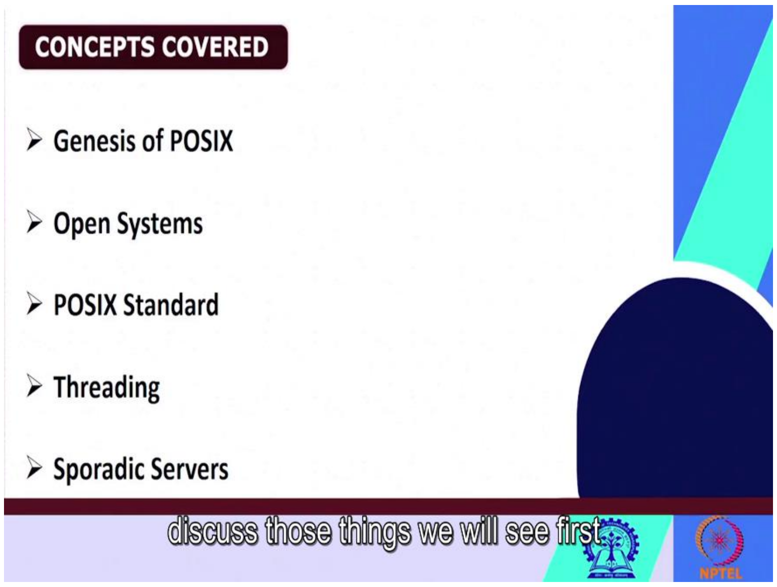
Real Time Systems
Professor Durga Prasad Mohapatra
Department of Computer Science and Engineering
National Institute of Technology Rourkela
Lecture 42
POSIX

Good afternoon to all of you. Today we will discuss a standard called as POSIX. So let us see what is meant by POSIX. What are the different parts of POSIX? What are the advantages of using POSIX? Let us discuss those things.

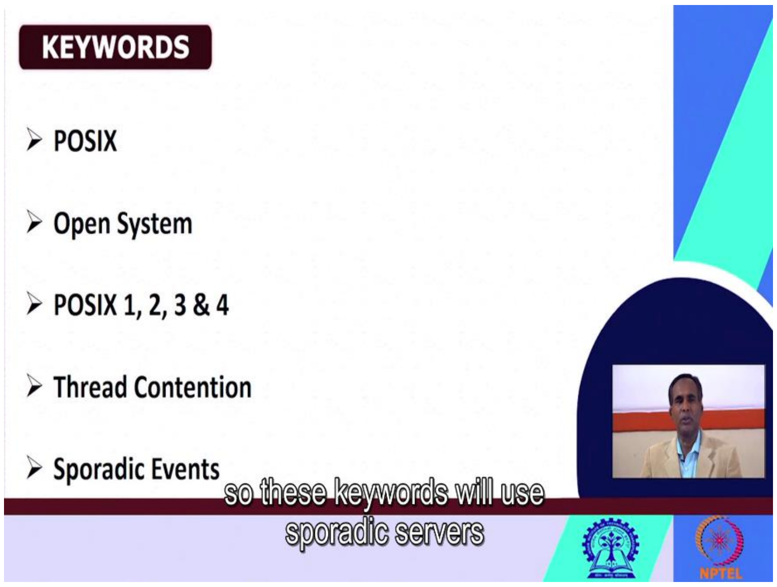
(Refer Slide Time: 00:34)



CONCEPTS COVERED

- Genesis of POSIX
- Open Systems
- POSIX Standard
- Threading
- Sporadic Servers

discuss those things we will see first



KEYWORDS

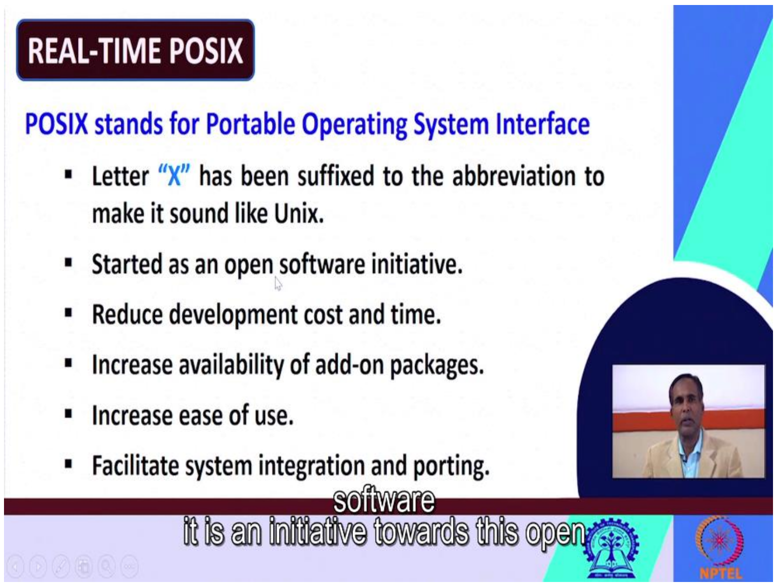
- POSIX
- Open System
- POSIX 1, 2, 3 & 4
- Thread Contention
- Sporadic Events

so these keywords will use sporadic servers

We will see first the genesis of POSIX, then what do you mean by open systems, what do you mean by open software, then the details about POSIX standard, the threading concepts whether

POSIX supports threading, multi-threading support regarding the multi-threading support, and we will finally see about the sporadic servers. So these keywords we will use.

(Refer Slide Time: 00:59)


REAL-TIME POSIX

POSIX stands for Portable Operating System Interface

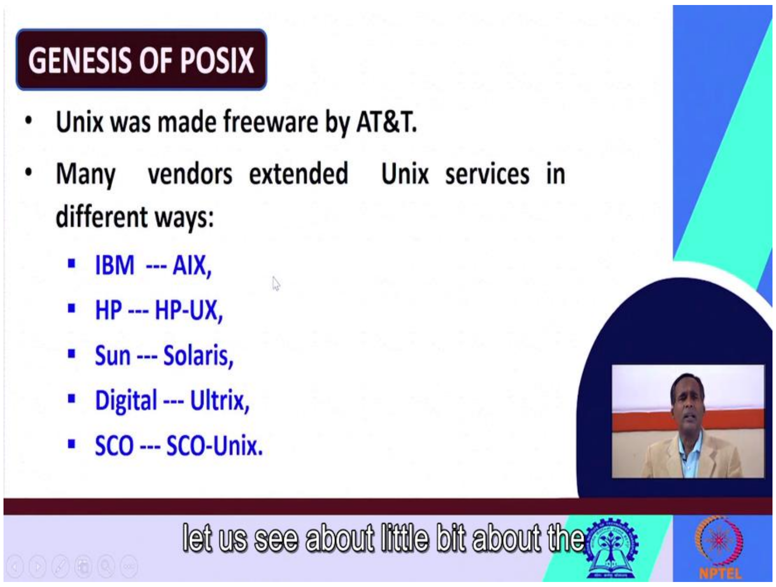
- Letter "X" has been suffixed to the abbreviation to make it sound like Unix.
- Started as an open software initiative.
- Reduce development cost and time.
- Increase availability of add-on packages.
- Increase ease of use.
- Facilitate system integration and porting.

Now let us see POSIX stands for what? So POSIX stands for Portable Operating System Interface. Then why does this x is there? It is just a portable operating system interface. This suppose the letter X has been added to the abbreviation to make it sound like Unix just like as Unix, Genix etcetera.

So similarly, just it will sound like Unix this letter X has been added to this what abbreviation. So this POSIX it was started as an open software initiative. So, you know, what are the advantages of open software? So, these are few of the advantages of using open software. For example, it will, by using open software, it will reduce the development cost and time, it will increase the availability of several add on packages, it will increase the ease of use. And also using this open software it will facilitate system integration and porting.

So, system integration will easier porting from one environment to another environment, it will be easier, these are the advantages of using open software. So, this what POSIX it is an offshoot of this open software, it is an initiative towards this open software.

(Refer Slide Time: 02:21)



GENESIS OF POSIX

- Unix was made freeware by AT&T.
- Many vendors extended Unix services in different ways:
  - IBM --- AIX,
  - HP --- HP-UX,
  - Sun --- Solaris,
  - Digital --- Ultrix,
  - SCO --- SCO-Unix.

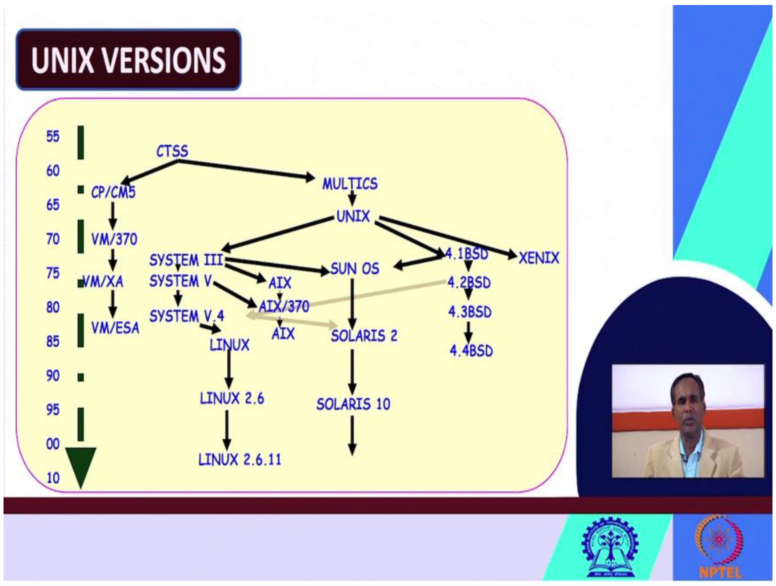
let us see about little bit about the

Let us see about little bit about the genesis of POSIX. This you know that this Unix was made freeware by AT&T. So first this Unix was made by freeware this AT&T and at that time or later on many vendors they extended the Unix services in different ways.

So, there are different vendors, they have extended this Unix in different ways for example, IBM has extended and produced a what a software called AIX, HP has developed HP-UX, Sun has developed sun Solaris, Digital has developed Ultrix, SCO has a version of Unix called as SCO-Unix. So, in this way many vendors they have extended the Unix services in different ways.

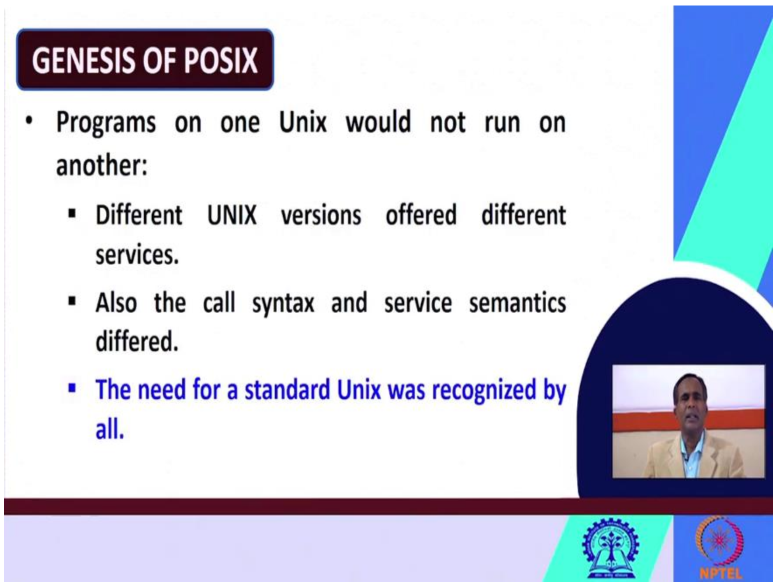
(Refer Slide Time: 03:06)

UNIX VERSIONS



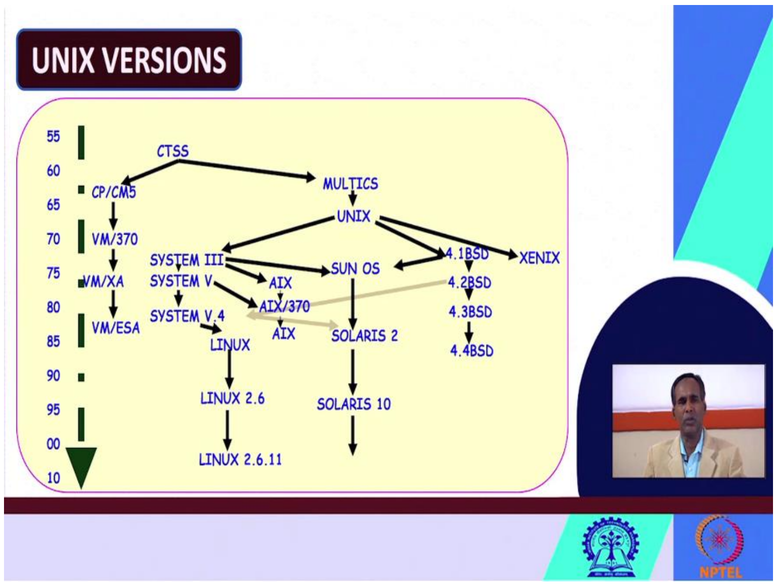
Now you can see look what the different versions of Unix it will see these are the different versions of Unix. Like as I have already told you, it was a Unix on the Sun operating system then Sun from Sun there is another version Solaris 2, Solaris 10 and like the System 3, System 4, System version System 5, System version V.4, like this similarly Xenix. So, these are different Unix versions you see the evolution has been shown here. So these are the different versions of Unix. This is for your better understanding I have listed all those versions of Unix maybe so far.

(Refer Slide Time: 03:48)



GENESIS OF POSIX

- Programs on one Unix would not run on another:
  - Different UNIX versions offered different services.
  - Also the call syntax and service semantics differed.
  - The need for a standard Unix was recognized by all.

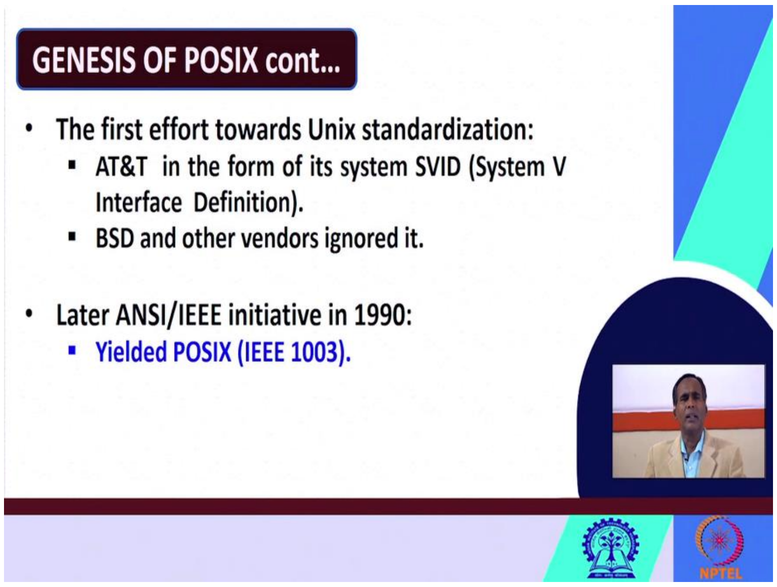


UNIX VERSIONS

Now let us again go back to the genesis of POSIX, you know that the programs on one Unix would not run on another, there is a problem, may, it may work, may not work. So, the programs on one Unix would not run on another. Did the different Unix versions they offer different services is not it?

So we have seen so many Unix versions see, there are so many Unix versions we have seen and now so the programs on one Unix would not run on another. Why? Because different Unix versions they offer different services, also on the call syntax and the service semantics of these different Unix versions they are also different. So there is a need for the standard Unix it was felt, so it was felt there should be a standard Unix for all, so the need for a standard Unix was recognized by all or all the users or all the vendors.

(Refer Slide Time: 04:47)



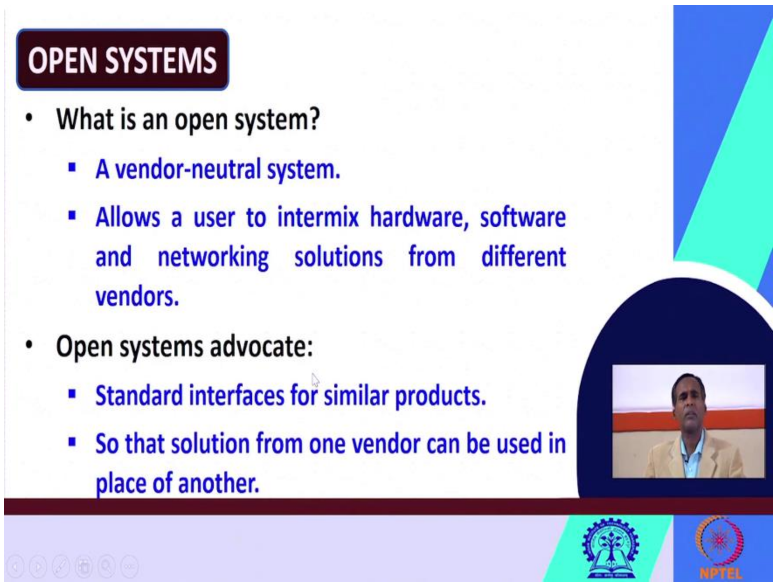
GENESIS OF POSIX cont…

- The first effort towards Unix standardization:
  - AT&T in the form of its system SVID (System V Interface Definition).
  - BSD and other vendors ignored it.
- Later ANSI/IEEE initiative in 1990:
  - Yielded POSIX (IEEE 1003).

Then in this direction first came forward AT&T, the first effort towards Unix standardization was made by AT&T in the form of its system SVID. You know SVID stands for System V Interface Definition, but at that time this BSD and other various vendors they have ignored it.

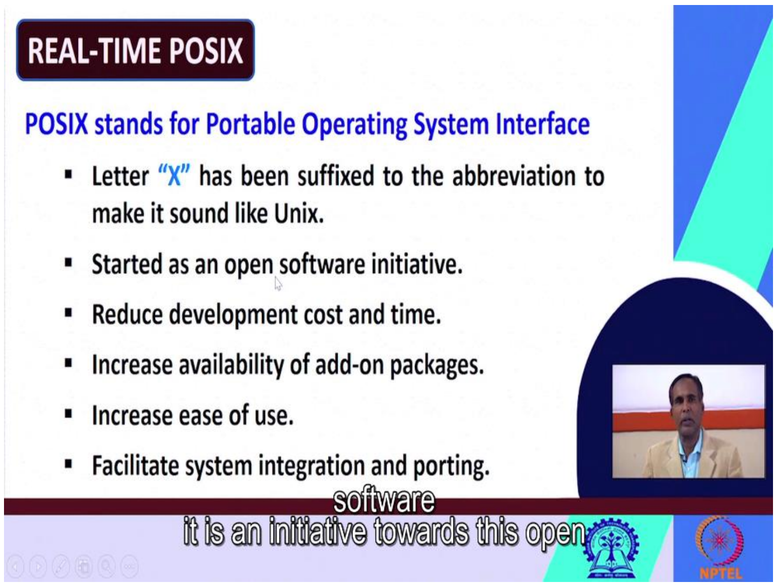
This Unix this AT&T has made some effort towards Unix standardization in the form of its system SVID. So however, at that time the BSD and other vendors they have simply ignored it. Later on ANSI or IEEE they have taken initiative in 1990 and this initiative was yielded, this version called as POSIX which is a standard which is a common standard so and you can see this is known as IEEE 1003.

(Refer Slide Time: 05:37)



OPEN SYSTEMS

- What is an open system?
  - A vendor-neutral system.
  - Allows a user to intermix hardware, software and networking solutions from different vendors.
- Open systems advocate:
  - Standard interfaces for similar products.
  - So that solution from one vendor can be used in place of another.



REAL-TIME POSIX

POSIX stands for Portable Operating System Interface

- Letter "X" has been suffixed to the abbreviation to make it sound like Unix.
- Started as an open software initiative.
- Reduce development cost and time.
- Increase availability of add-on packages.
- Increase ease of use.
- Facilitate system integration and porting.

software
it is an initiative towards this open

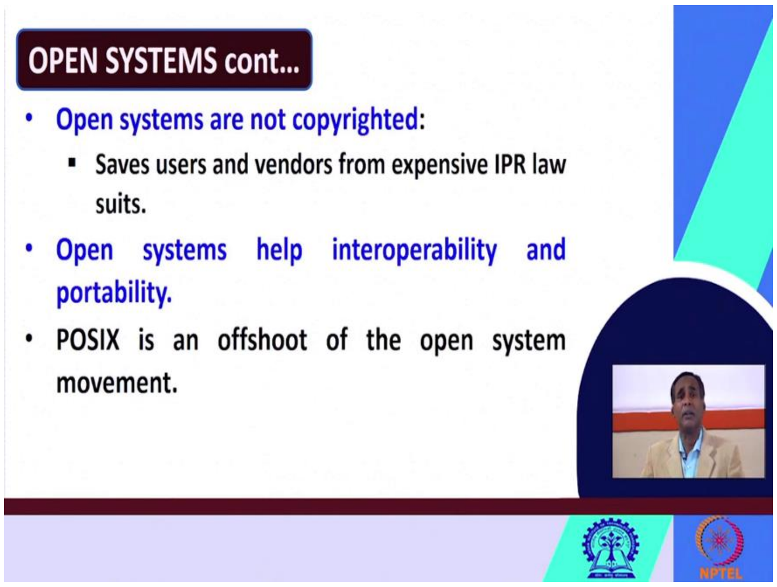
Now let us see, what is an open system? I have already told you this POSIX is now an open system initiative. So, what is an open system? POSIX, I have already told you somewhere else here you can see that this POSIX started as an open software initiative. So let us first see about this open system and then we will see about this open software.

What is an open system? A open system is a vendor neutral system, it is not clearly specific to any vendor, it is a vendor neutral system, it allows the user to intermix hardware, software and network solutions from different vendors.

So, if you are using an open system, it will allow you to intermix different hardware, different software and working and different networking solutions from different vendors and you can what intermix them. So, these open systems they advocate standard interfaces for similar products. So, these open systems they advocate for what? They advocate for standard interfaces for similar products.

So, that the solution from one vendor can be used in place of another, it is a, they want to make it standardized to the solution that you are getting from one vendor it can be used in place of another. So, these open systems they advocate standard interfaces for similar products, so that the solution from one vendor can be used in place of another.

(Refer Slide Time: 06:56)



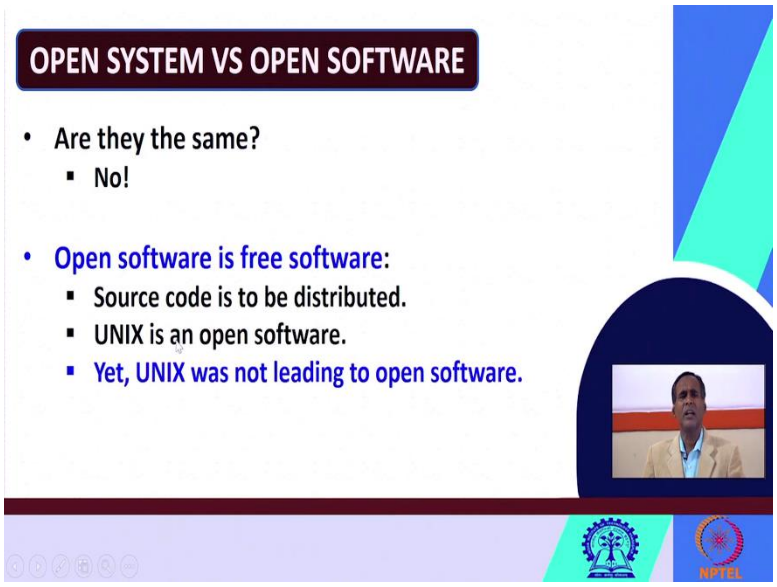
OPEN SYSTEMS cont...

- Open systems are not copyrighted:
  - Saves users and vendors from expensive IPR law suits.
- Open systems help interoperability and portability.
- POSIX is an offshoot of the open system movement.

The open systems are not copyrighted; I hope you know what is IPR, what is copyright etc. So, the open systems are not copyrighted. So, these open systems they save users and vendors from expensive IPR lawsuits. So, you know filing IPRs are very tedious, very much expensive.

So, these open systems they are not at all copyrighted and hence but they save the users and vendors are from my expensive IPR lawsuits. So, the open systems help interoperability and portability, another advantage of open of using open systems. So, the open systems they help interoperability and portability I have already told you that POSIX is an offshoot of the open system movement. So, this POSIX that we are using this is an offshoot of the open system movement.

(Refer Slide Time: 07:43)



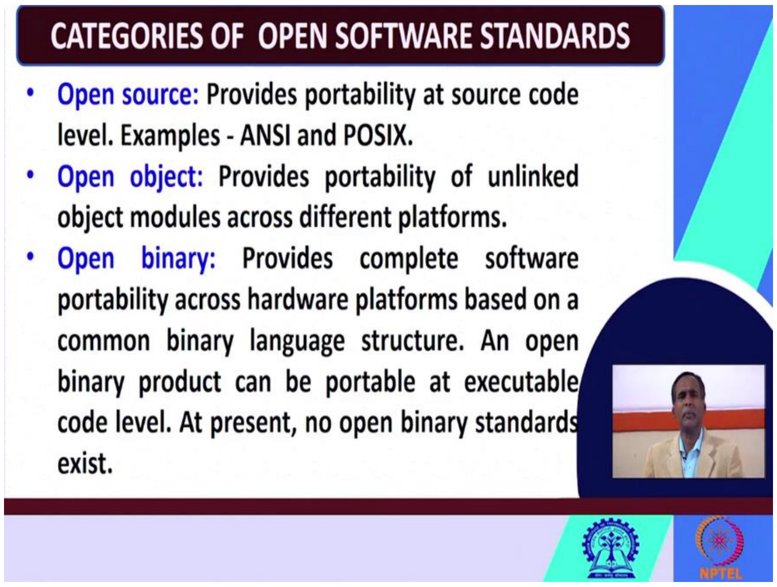
OPEN SYSTEM VS OPEN SOFTWARE

- Are they the same?
  - No!
- Open software is free software:
  - Source code is to be distributed.
  - UNIX is an open software.
  - Yet, UNIX was not leading to open software.

Now the question, whether open system and open software are the same? The answer is no. We have already discussed what is open system, on what is the advantage of using open system we have already seen. Now let us see what do you mean by open software. Open software means a free software.

Open software is free software which is freely available here the source code is to be distributed, so if you are developing a software you want to make it open software or free software you have to distribute the source code. This Unix is an open software, you will see that this Unix version open software, but this Unix was not leading to open software. Yet still Unix was not leading to open software.

(Refer Slide Time: 08:23)



CATEGORIES OF OPEN SOFTWARE STANDARDS

- Open source: Provides portability at source code level. Examples - ANSI and POSIX.
- Open object: Provides portability of unlinked object modules across different platforms.
- Open binary: Provides complete software portability across hardware platforms based on a common binary language structure. An open binary product can be portable at executable code level. At present, no open binary standards exist.

Now let us see what are the different categories of open software standards. There are mainly three categories of open software standards, one is the open-source, open object and another and the other one is open binary. So let us say first about open-source standards, these open-source standards they provide portability at source code level only.

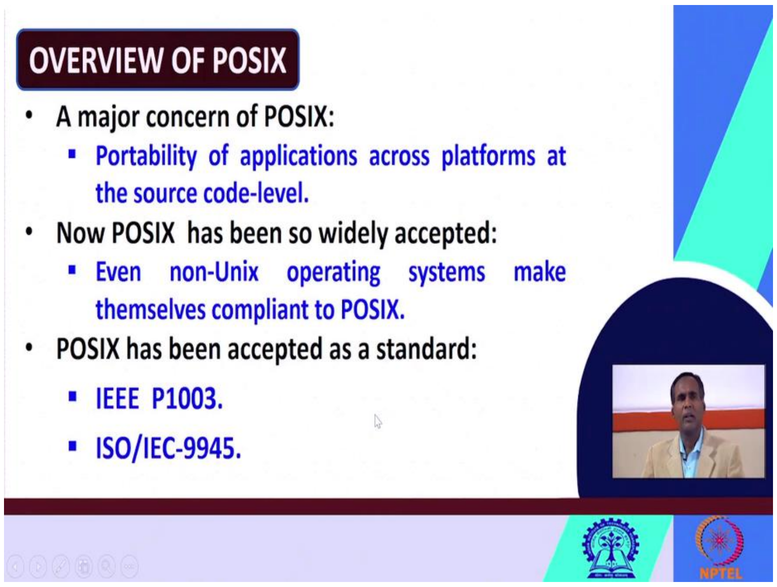
So, these standards open-source standards they provide portability, at source code level and the popular examples of open source are ANSI and POSIX. What is the open object? This open object standard it provides portability, where? It provides portability of only the object modules across different platforms. So, if you are using different platforms, so this open object standard it provides portability of unlink the object modules across the different platforms.

What is open binary? Open binary standard, this provides complete software portability, across what? Across hardware platforms. So open binary standard, it provides complete software portability across the hardware platforms based on what, based on a common binary language structure.

An open binary product it can be portable at executable code level, I have already told you this open source it provides portability and source code level. Similarly, open binary standard, an open binary product or an open binary standard they can be portable at where, at executable code level.

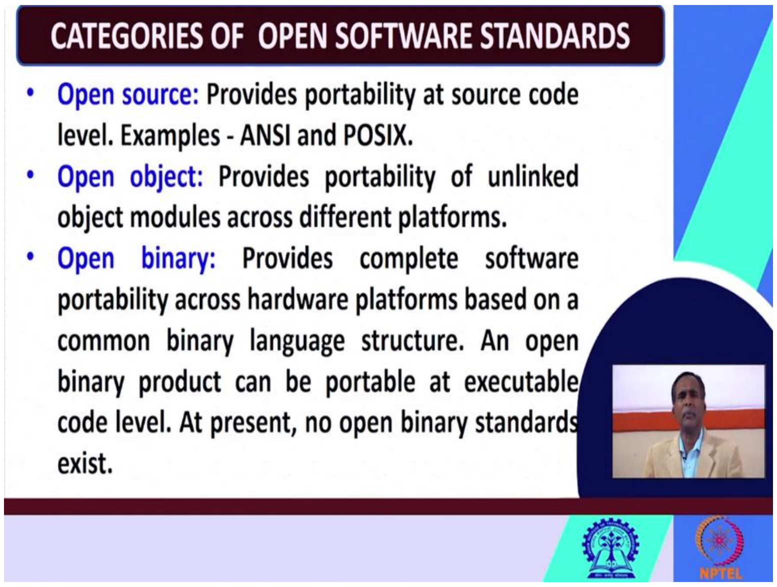
At present, if you will see examples, at present no open binary standards exist but you can still check the net if in the recent what few days if any open binary standards come up or not, I am not sure but as far as my knowledge is concerned at present no open binary standards exist. So, these are the classifications of different open software standards.

(Refer Slide Time: 10:14)



OVERVIEW OF POSIX

- A major concern of POSIX:
  - Portability of applications across platforms at the source code-level.
- Now POSIX has been so widely accepted:
  - Even non-Unix operating systems make themselves compliant to POSIX.
- POSIX has been accepted as a standard:
  - IEEE P1003.
  - ISO/IEC-9945.



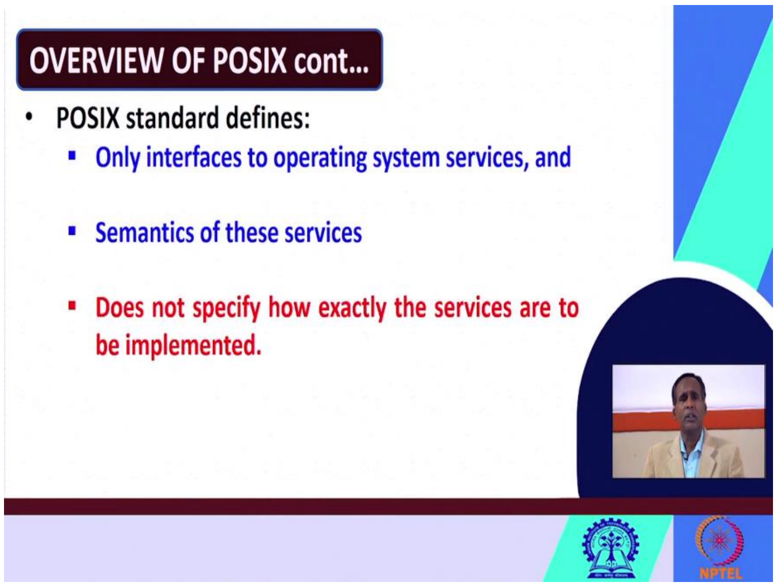
CATEGORIES OF OPEN SOFTWARE STANDARDS

- Open source: Provides portability at source code level. Examples - ANSI and POSIX.
- Open object: Provides portability of unlinked object modules across different platforms.
- Open binary: Provides complete software portability across hardware platforms based on a common binary language structure. An open binary product can be portable at executable code level. At present, no open binary standards exist.

Now we will go to our original discussion that is POSIX. Let us see the overview of POSIX. So, what is the major concern of POSIX? The major concern of POSIX is portability of applications across platforms are the source code level. I have already told you open source the example is POSIX and here this open source it provides portability at source code level.

So, this major concern of POSIX is to provide the portability of applications across what across platforms at the source code level, this is a POSIX. Now POSIX has been so widely accepted even non Unix operating systems they are also making themselves compliant to POSIX, this POSIX has become so much popular, now POSIX has been so widely accepted even the non-Unix operating systems they make themselves compliant to what? To the POSIX standard. POSIX has been accepted as a standard by IEEE P103 and ISO IEC 9945.

(Refer Slide Time: 11:14)



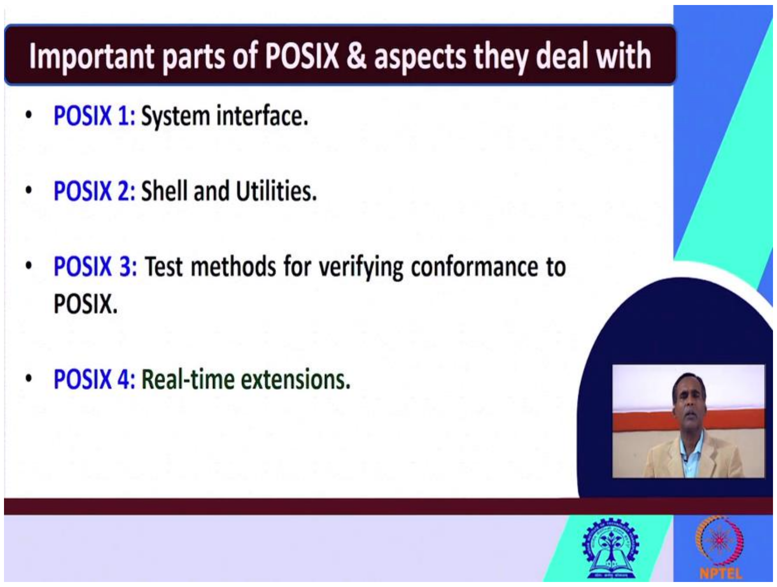
OVERVIEW OF POSIX cont…

- POSIX standard defines:
  - Only interfaces to operating system services, and
  - Semantics of these services
  - Does not specify how exactly the services are to be implemented.

So what does it define? POSIX standard defines only interfaces to operating system services. This POSIX standard it defines only the interfaces to the operating system services and the semantics to these services. So the POSIX standard it defines the interfaces to operating system devices only the interfaces to operating system devices and the semantics of the services.

So, POSIX standard defines only interfaces to operating system devices and the semantics of the services. But one problem with this POSIX, it does not specify how exactly these services are to be implemented. So, how exactly the services, these operating system services are to be implemented this POSIX standard does not specify anything regarding this.

(Refer Slide Time: 12:03)



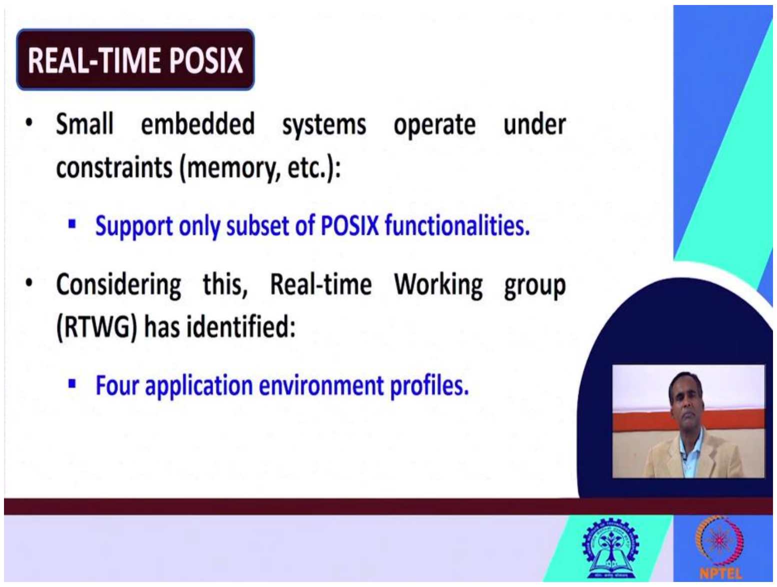
Important parts of POSIX & aspects they deal with

- POSIX 1: System interface.
- POSIX 2: Shell and Utilities.
- POSIX 3: Test methods for verifying conformance to POSIX.
- POSIX 4: Real-time extensions.

Now, quickly let us see the important parts of POSIX and aspects they deal with. You will see some of the important parts of POSIX I have listed here like POSIX 1, POSIX 2, POSIX 3, POSIX 4. POSIX 1 deals with system interfaces, POSIX 2 deals with the shell and the utilities, POSIX 3 deals with the test methods for verifying the conformance to the POSIX, whether the methods they are conformance to your POSIX or not.

Who will deal with this? POSIX 3. So, POSIX 3 deals with the test methods for verifying conformance to POSIX. And what is about POSIX 4, that we are most interested in,4 that is the real time extensions. So, the real time extensions are provided in part 4, POSIX 4. So, POSIX 4 deals with the real time extensions, it is known as POSIX RT. This POSIX 4 is also known as POSIX RT because it supports real time extensions it contains real time extensions.

(Refer Slide Time: 13:08)



REAL-TIME POSIX

- Small embedded systems operate under constraints (memory, etc.):
  - Support only subset of POSIX functionalities.
- Considering this, Real-time Working group (RTWG) has identified:
  - Four application environment profiles.

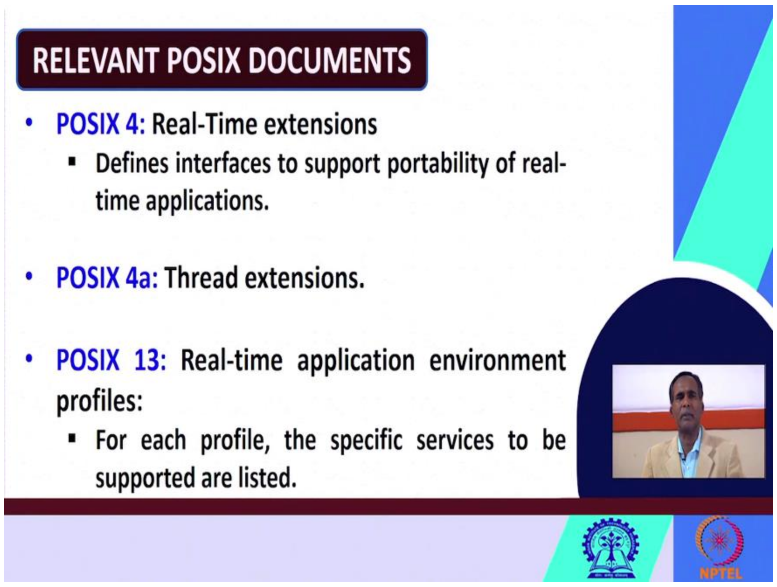
So real time POSIX that is the part 4, POSIX 4 this is known as also POSIX RT the small embedded systems they operate under the constraints, is not it? The small embedded systems they operate under several constraints such as the memory, limitations of the memory etc.

So, this small embedded system they support only a subset of POSIX functionality. There are a large number of POSIX functionalities, these small embedded systems they may not support all the POSIX functionalities, they support only a subset of POSIX functionalities.

So, considering this fact, there is a group called large real time working group RTWG, they have identified 4 application and environmental profiles. As I have already told you that small embedded systems, they operate under several constraints such as memory limitations etcetera.

So, they support only the subset of the available POSIX functionalities. So real time working group RTWG. So, based on this it has identified 4 application environment profiles. Let us see what are those 4 application environment profiles.

(Refer Slide Time: 14:16)



RELEVANT POSIX DOCUMENTS

- **POSIX 4:** Real-Time extensions
  - Defines interfaces to support portability of real-time applications.
- **POSIX 4a:** Thread extensions.
- **POSIX 13:** Real-time application environment profiles:
  - For each profile, the specific services to be supported are listed.

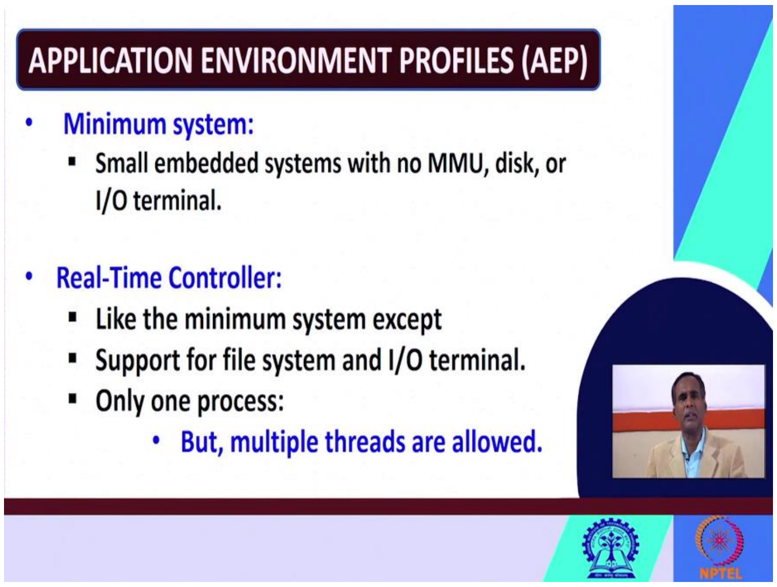
But before going to that let us first to see about the what are the different POSIX documents, so many POSIX documents are there, I have here shown only some relevant, some important POSIX documents. For example, POSIX 4, I have already told you POSIX 4, it deals with the real time extensions. POSIX 4 deals with real time extensions.

It defines the interfaces to support portability of real time application. So, this POSIX 4 it defines the interfaces in order to support portability of real time applications. Similarly, POSIX 4a, it deals with the thread extensions, POSIX 13, it deals with what this contains real time application environment profile.

So, POSIX 13, it contains the real time application environment profiles. So, in this POSIX 13 whatever what is happening as I have already told you this contains real time application environment profiles in which for each profile the specific services have to be supported, these are listed.

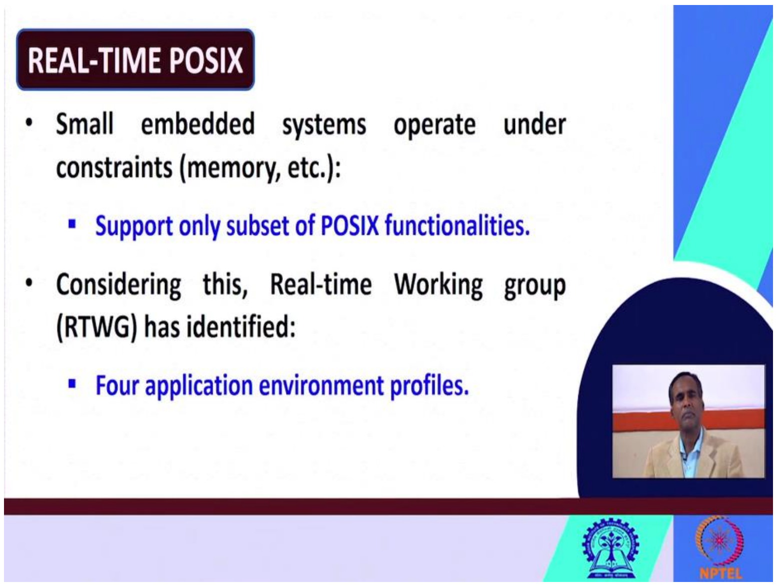
So, POSIX 13 it contains real time application environment profiles for each profile the specific services to be supported, they are listed. So, this is what is contained in POSIX 13. So, now we will go back to the 4 application environments.

(Refer Slide Time: 15:31)



APPLICATION ENVIRONMENT PROFILES (AEP)

- Minimum system:
  - Small embedded systems with no MMU, disk, or I/O terminal.
- Real-Time Controller:
  - Like the minimum system except
  - Support for file system and I/O terminal.
  - Only one process:
    - But, multiple threads are allowed.



REAL-TIME POSIX

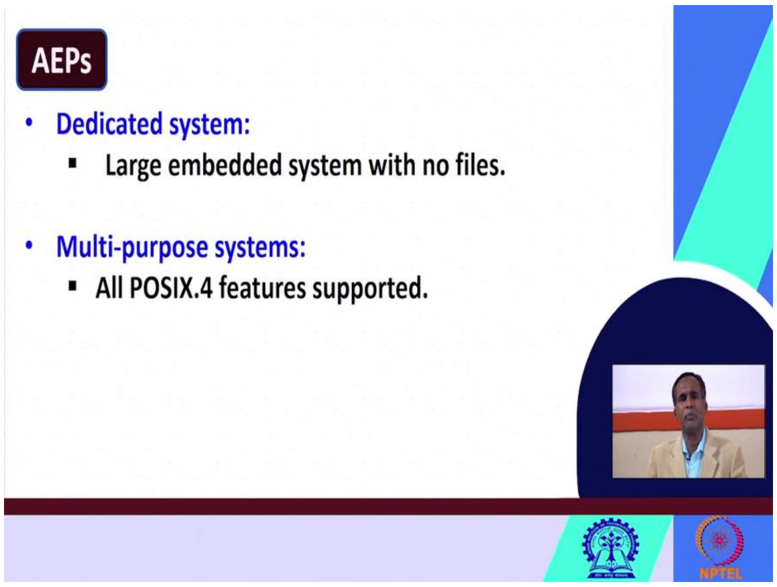
- Small embedded systems operate under constraints (memory, etc.):
  - Support only subset of POSIX functionalities.
- Considering this, Real-time Working group (RTWG) has identified:
  - Four application environment profiles.

As I have already told you, that RTWG has identified 4 application environment profiles. So, now let us see what are those four application environment profiles. First one is minimum system. This minimum system is for small embedded systems with no main memory unit disk or IO terminal because this is this minimum system is meant for small embedded systems with what.

So, these minimum sub systems it is meant for small embedded systems, this minimum system it supports small embedded systems with no main memory unit, disk or IO terminal then the next environment then the next application environment profile is real time controller. So, this is like minimum system except it also supports for file system and IO terminal, may we see IO terminal, it is not supported in minimum system.

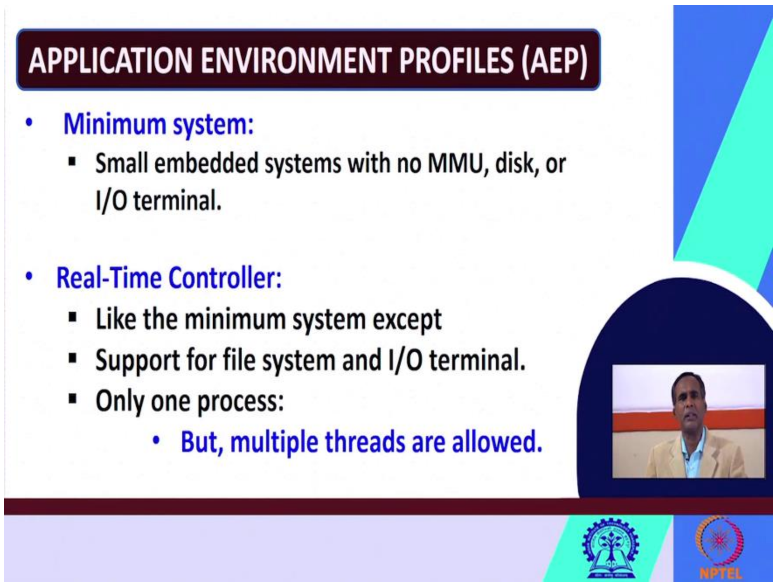
But here it this real time controller it supports IO terminal as well as it supports file systems. So, in real time controller only one process there is only one process. So, in real time controller there is only one process, but multiple threads are allowed, but here multiple number of threads are allowed this is about real time controller.

(Refer Slide Time: 16:43)



AEPs

- **Dedicated system:**
  - Large embedded system with no files.
- **Multi-purpose systems:**
  - All POSIX.4 features supported.



APPLICATION ENVIRONMENT PROFILES (AEP)

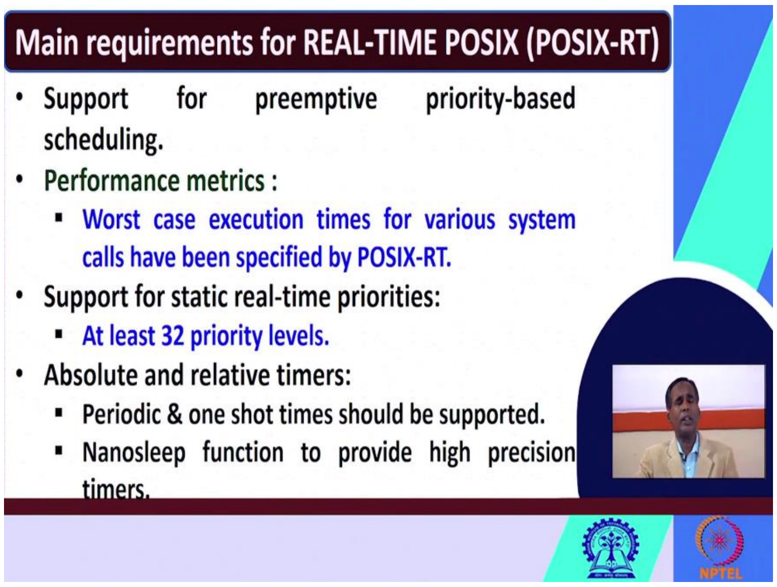
- **Minimum system:**
  - Small embedded systems with no MMU, disk, or I/O terminal.
- **Real-Time Controller:**
  - Like the minimum system except
  - Support for file system and I/O terminal.
  - Only one process:
    - But, multiple threads are allowed.

Then next application environment profile is dedicated system. Dedicated systems are made for large embedded systems with no files and the last application environment profile is multipurpose system. So, all POSIX. So, in multipurpose systems, I have already told you there are several features, several functionalities of all POSIX.

So, all these AEPs, application environment profiles may not support all of them, but this multipurpose system it supports all POSIX 4 feature. So, how many POSIX 4 or that means

POSIX real time POSIX features or functional which are there, they are supported in multi-purpose systems. So, in multi-purpose systems all POSIX 4 functionalities and features are supported.

(Refer Slide Time: 17:26)


Main requirements for REAL-TIME POSIX (POSIX-RT)

- Support for preemptive priority-based scheduling.
- Performance metrics :
  - Worst case execution times for various system calls have been specified by POSIX-RT.
- Support for static real-time priorities:
  - At least 32 priority levels.
- Absolute and relative timers:
  - Periodic & one shot times should be supported.
  - Nanosleep function to provide high precision timers.

Now, let us quickly look at the main requirements for the real time POSIX. So, what are the important requirements for real time POSIX? So, first it should support for preemptive priority-based scheduling.

So, this real time POSIX it should support what preemptive priority-based scheduling. I hope we have already known, what do you mean by preemptive scheduling, what do you mean by non-preemptive scheduling we have already known. So, these POSIX it must support preemptive priority-based scheduling.

So, now, next item is performance metrics. So, in case of these POSIX-RT, the worst-case execution times for various system calls, they have already been specified by POSIX RT. So, you do not have face any problem because this worst-case execution times are for various system calls have already been specified by this POSIX RT.

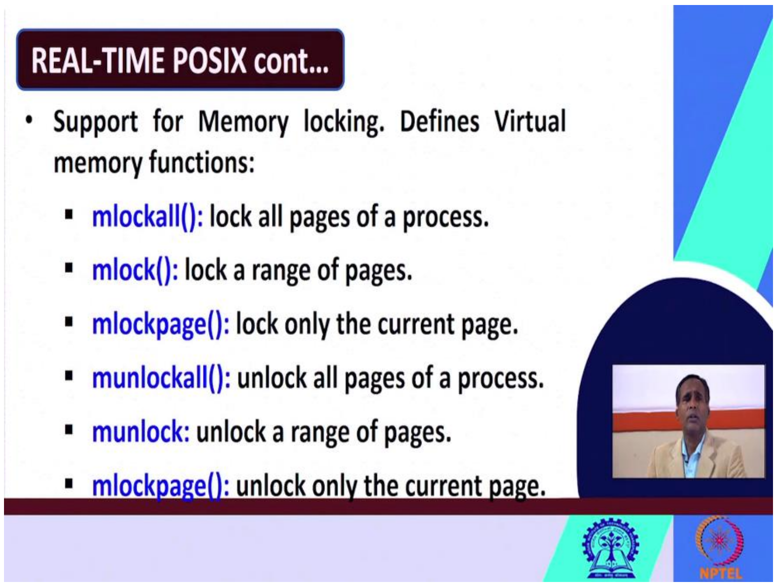
So, this is the second requirement that is called as performance metrics the worst-case execution times for various system calls have been specified by POSIX RT. Then the support for static real time priority, we have already told you earlier this one more feature for real time systems one is preemptive priority-based scheduling, another is using what static real time priorities.

So, in contrast to dynamic real time, dynamic priorities you should use static real time priorities. POSIX must support the static real time priority that is you this is one of the another important requirement and at least 32 priority levels must be provided. Then regarding the timers, we will discuss about absolute and relative timers.

The POSIX standards must support this periodic and one time, one shot timers. So, you know that I have already told you periodic timers and one shot timers in the previous classes, examples of one shot timer you have known watch dog timers.

So, these periodic and one-shot timers should be or they must be supported by this POSIX RT. And regarding this precision or this resolution nanosleep function should be used to provide high precision timer. So, this is another requirement nanosleep function it has to be used or to provide a high-resolution timer. So, these are some of the important requirements for real time POSIX few more requirements are there let us quickly see them.

(Refer Slide Time: 19:47)



REAL-TIME POSIX cont…

- Support for Memory locking. Defines Virtual memory functions:
  - mlockall(): lock all pages of a process.
  - mlock(): lock a range of pages.
  - mlockpage(): lock only the current page.
  - munlockall(): unlock all pages of a process.
  - munlock: unlock a range of pages.
  - mlockpage(): unlock only the current page.

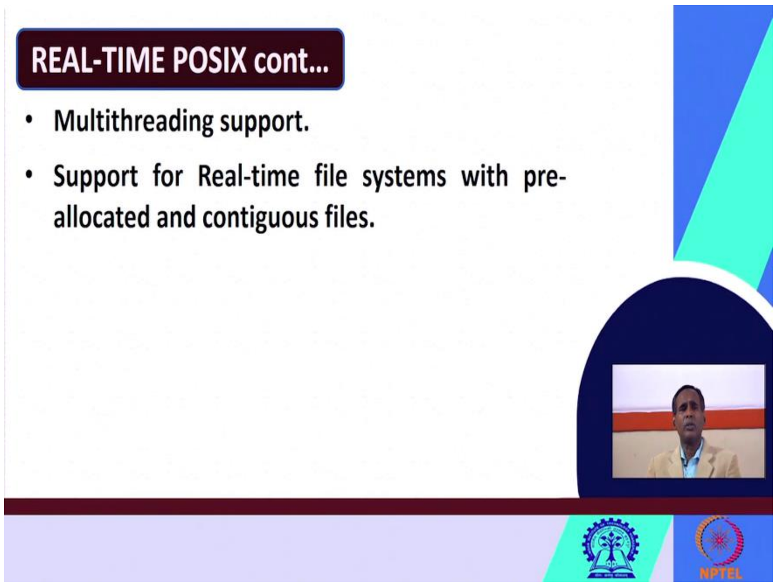
So, this real time POSIX or POSIX RT must support for memory locking. In POSIX RT what happens, it defines several virtual memory functions. Such as, so two important functions it provides locking and unlocking. For example, it provides a function called mlockall, what does it do? it is used to lock all the pages of a process.

Similarly, simply mlock it can be used to lock a range of pages. mlockpage it can be used to lock only the current pages. So, in this way this what memory locking functions are used or they are provided in real time POSIX or POSIX RT.

Similarly, you must this POSIX RT also defines various virtual memory functions for unlocking. For example, if you are using a munlockall it will be used to unlock all pages of a process, munlock it is used to unlock a range of pages, mlockpage it unlocks only the current page.

So, in this way, this real time POSIX or POSIX RT supports memory locking, it defines various virtual memory functions such as mlockall, munlockall, mlock, munlock and mlockpage and munlockpage. I think there is a typing mistake here. It should be munlockpage I am sorry, it should be munlockpage please correct.

(Refer Slide Time: 21:05)



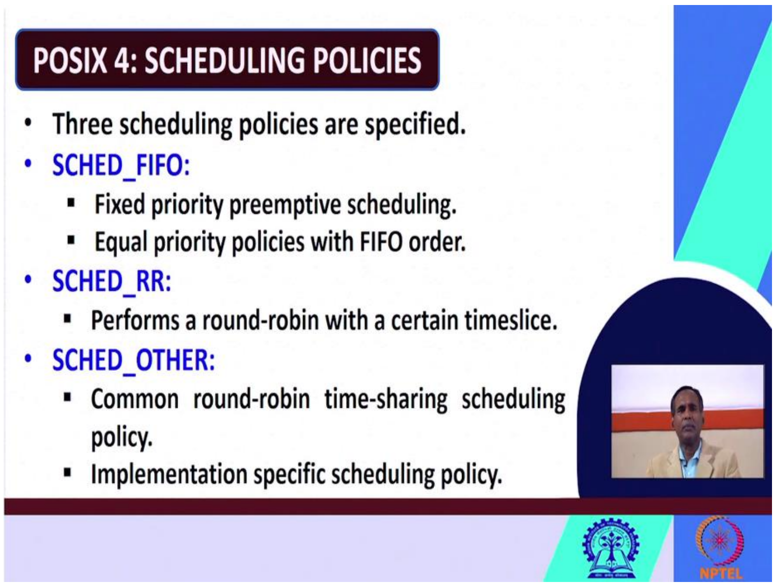
REAL-TIME POSIX cont...

- Multithreading support.
- Support for Real-time file systems with pre-allocated and contiguous files.

So, besides all those important requirements, two more other requirements are there, one is multi-threading support, the POSIX must have multi-threading support and similarly POSIX must support for real time file system. So, when you are using POSIX, one of the important requirements is that it should support real time file systems.

So, POSIX must support the real time file systems with pre allocated and contiguous files. So, these file systems they are you should be able to pre allocate files and it should use contiguous files. So real time POSIX support, must support for real time file systems with pre allocated and a contiguous file. So, these are some of the important requirements we have discussed which must be supported by POSIX or POSIX RT.

(Refer Slide Time: 21:50)



POSIX 4: SCHEDULING POLICIES

- Three scheduling policies are specified.
- SCHED_FIFO:
  - Fixed priority preemptive scheduling.
  - Equal priority policies with FIFO order.
- SCHED_RR:
  - Performs a round-robin with a certain timeslice.
- SCHED_OTHER:
  - Common round-robin time-sharing scheduling policy.
  - Implementation specific scheduling policy.

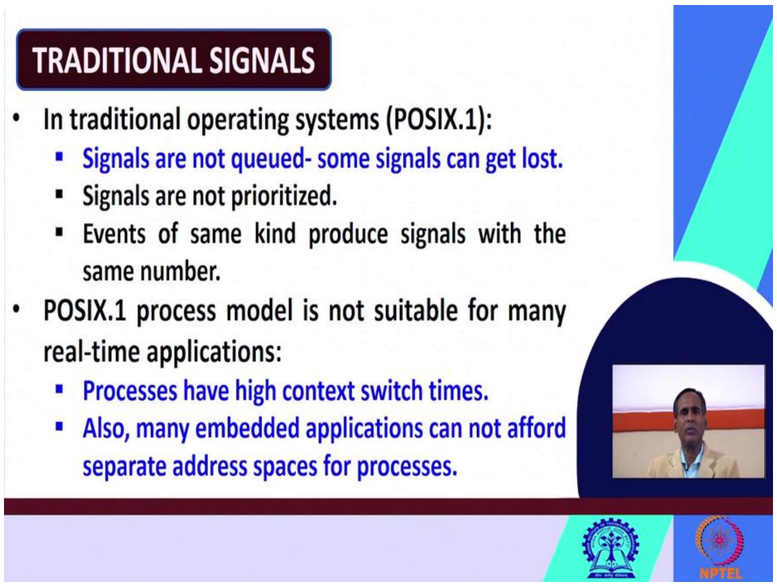
Now let us see what kind of scheduling policies POSIX RT or POSIX 4 for real time application this must support. So, three scheduling policies are specified in POSIX 4 or POSIX RT. One is scheduling in FIFO order, scheduling in Round Robin and scheduling others.

So, in the scheduling FIFO, what does it do? It uses fixed priority preemptive scheduling. In FIFO scheduling it uses fixed priority preemptive scheduling, here the equal priority policies with FIFO order. So, when you will see the tax are having equal priority policies with FIFO order. So, this the scheduling policy scheduled FIFO, this you just equal priority policies with FIFO order.

Then you will see, the next scheduling policy, that is scheduling Round Robin. So, this scheduling policy performs a round robin with a certain time slice. So, this policy it performs or it works in a round robin fashion with certain or specific time slice. And scheduling others, so they use common round robin timesharing scheduling policy. So, this policy scheduling order, it uses common round robin time sharing scheduling policy.

So, implementation specific scheduling policy. So, this policy scheduling order, it is a implementation specific scheduling policy that means this scheduling policy is very much implementation specific may differ from implementation to implementation.

(Refer Slide Time: 23:23)



TRADITIONAL SIGNALS

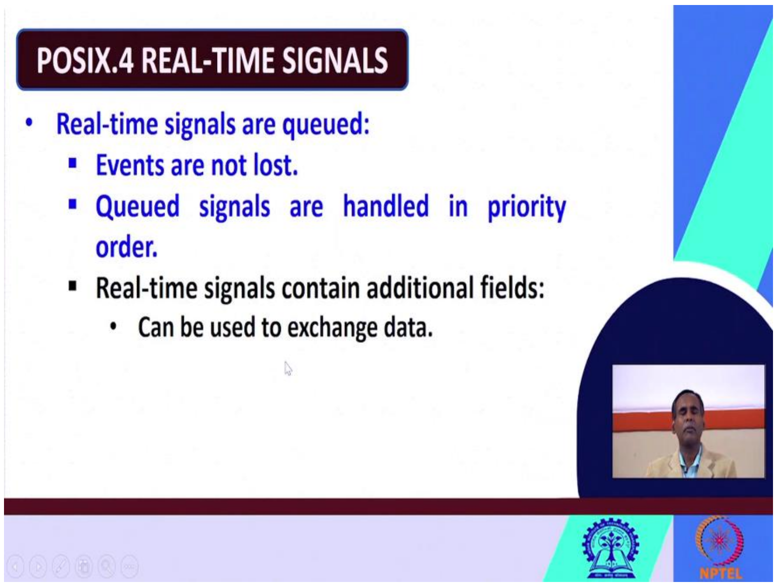
- In traditional operating systems (POSIX.1):
  - Signals are not queued- some signals can get lost.
  - Signals are not prioritized.
  - Events of same kind produce signals with the same number.
- POSIX.1 process model is not suitable for many real-time applications:
  - Processes have high context switch times.
  - Also, many embedded applications can not afford separate address spaces for processes.

So now let us see, look at about the traditional signals. In traditional operating systems like POSIX 1 2 etc. These signals they are not queued. So, if these signals are not queued what will happen? So some signals can get lost. The signal and another problem is the traditional operating system is that the signals are not prioritized and what will happen events of same kind so they may produce signal with the same number, is not it?

So, since these signals are not queued, they are not prioritized. So, the events of the same kind they may produce signals with the same number. So, this is the drawback that is happening in the traditional operating system. So now this POSIX 1 process model it is not suitable for many real time applications. I am discussing about the traditional like POSIX 1 POSIX 2 etc.

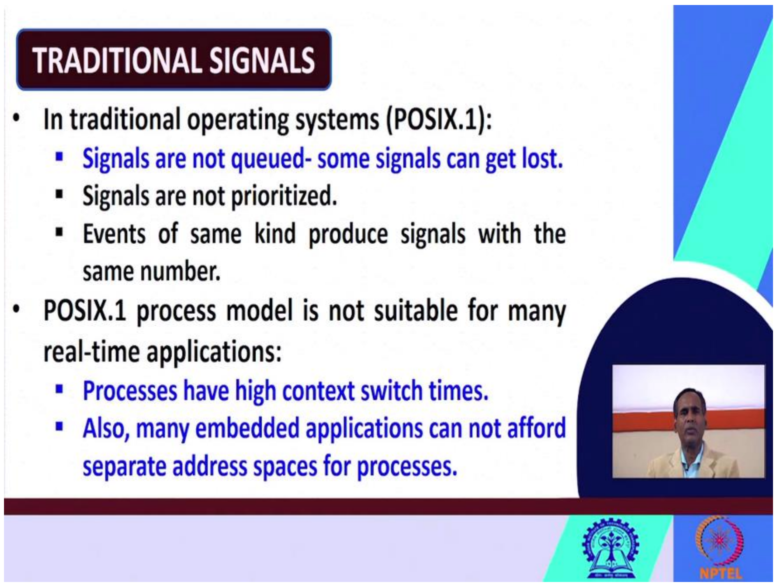
So, these POSIX 1 process model it is not suitable for many real time applications. Because here the processes have very high context switch times which is not desirable; also many embedded applications cannot afford separate address space for the processes. So, you will see this is the drawback of POSIX 1 because many embedded applications they cannot afford separate address spaces support for processes. So that is why this POSIX 1 process model is not suitable for many real time applications.

(Refer Slide Time: 24:41)



POSIX.4 REAL-TIME SIGNALS

- Real-time signals are queued:
  - Events are not lost.
  - Queued signals are handled in priority order.
  - Real-time signals contain additional fields:
    - Can be used to exchange data.



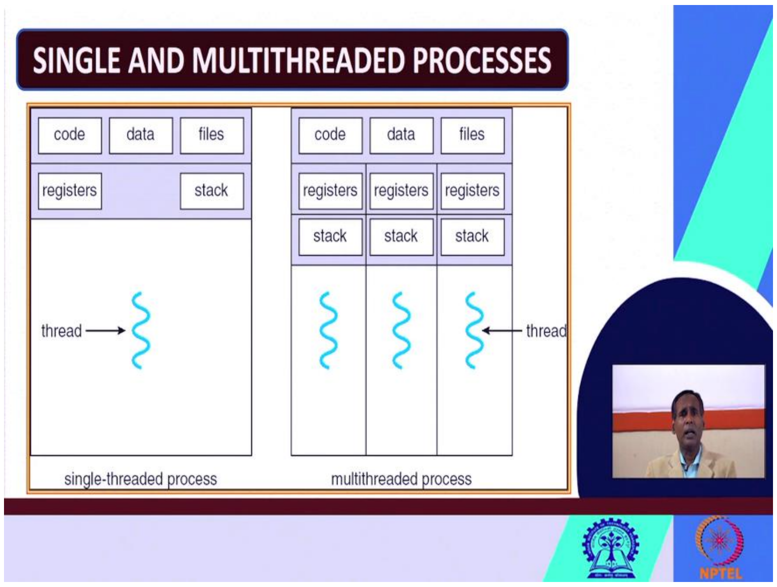
TRADITIONAL SIGNALS

- In traditional operating systems (POSIX.1):
  - Signals are not queued- some signals can get lost.
  - Signals are not prioritized.
  - Events of same kind produce signals with the same number.
- POSIX.1 process model is not suitable for many real-time applications:
  - Processes have high context switch times.
  - Also, many embedded applications can not afford separate address spaces for processes.

Now let us see in POSIX 4 how the signals are handled. In POSIX 4 the real time signals they are queued. I have already told you in POSIX 1, the signals are not queued. But here in POSIX 4, the real time signals are queued. As a result, no event is lost, the queued signals are handled in priority order.

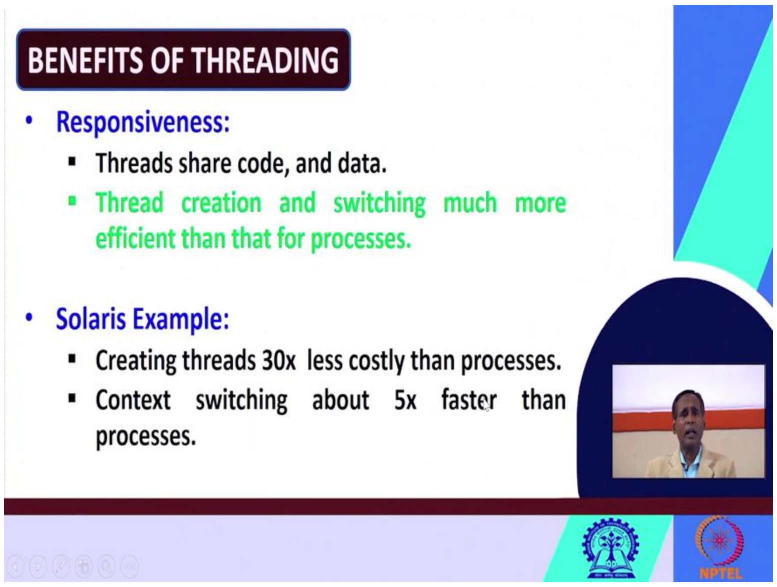
So, since these signals are sorted in a queue, they are put in a queue, the queued signals are handled in some priority order. So, each signal you will be given some priority and the queue signals are handled according to their priority. Real time signals contain additional fields. So, these real time signals they may contain additional fields which can be used to exchange data. So, these real time signals they may contain what, some additional fields which can be used to exchange the data.

(Refer Slide Time: 25:31)


SINGLE AND MULTITHREADED PROCESSES

I have already told you POSIX supports multi-threading or multi-threading support is there in what POSIX, this is the example how a single thread is there but in POSIX it has a multi-threading support.

(Refer Slide Time: 25:44)


BENEFITS OF THREADING

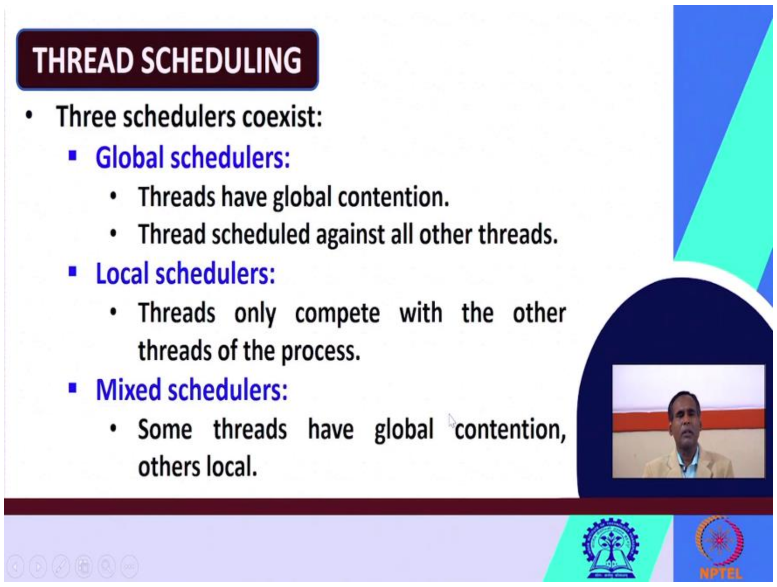
- **Responsiveness:**
  - Threads share code, and data.
  - Thread creation and switching much more efficient than that for processes.
- **Solaris Example:**
  - Creating threads 30x less costly than processes.
  - Context switching about 5x faster than processes.

Now, benefits of threading why POSIX supports multi-threading, what is the benefit of threading? So, there are several benefits of threading. For example, the responsiveness; the threads share code and data, the threads that you are using, you are using multiple threads, the threads, they share the code and not the data.

Thread creation and switching much more efficient than that for the processes. So, if you will compare that creating the threads or switching among the threads in comparison to creation of processes or switching up the processes. So, it is much more easier in case of threads, thread creation and switching they are much more efficient than that for the processes.

If you take this Sun Solaris as an example creating the threads in Sun Solaris it is 30 times less costlier than the processes. So here it is less costlier. So, in Sun Solaris creating the threads is 30 times less costly than the processes also the context switching is about 5 times faster than the processes. In Sun Solaris the context switching is about 5 times faster than the processes so these are the advantages. These are some of the advantages of using threading.

(Refer Slide Time: 26:56)



THREAD SCHEDULING

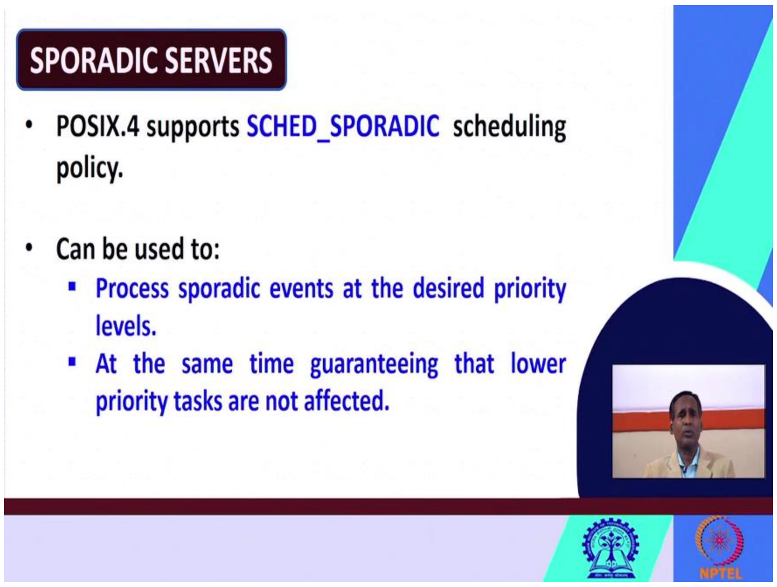
- Three schedulers coexist:
  - **Global schedulers:**
    - Threads have global contention.
    - Thread scheduled against all other threads.
  - **Local schedulers:**
    - Threads only compete with the other threads of the process.
  - **Mixed schedulers:**
    - Some threads have global contention, others local.

So what are the different thread scheduling techniques exists? So normally three schedulers exist maybe in case of these POSIX. One is global scheduler; another local scheduler; another is mixed scheduler. In global scheduler what is happening here? Here the threads have global contention as it name suggests, global schedulers, the threads of global contention, then the threads scheduled against all other threads.

If you since there is a global scheduler, so this thread is scheduled against all other threads. In local scheduler, what is happening? The threads only compete with the other threads of the same process. So that is why the name is local schedulers. Here are the threads, they only compete to the what? They only compete with the other threads of the same process. So here are the threads, they only compete with the other threads of the same process.

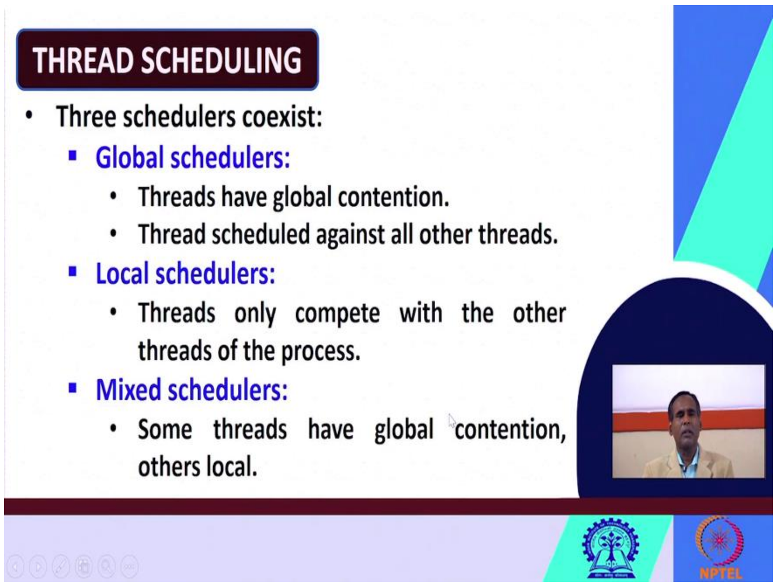
Then Mixed scheduler, what is happening? So, as it name suggests, mixed scheduler or hybrid scheduler, here some threads have global contention and others have local. That is why it is a mixed scheduler or hybrid scheduler. So here are some threads have global contention and others local. So that is why this is known as mixed scheduler or hybrid Scheduler.

(Refer Slide Time: 28:05)



SPORADIC SERVERS

- POSIX.4 supports SCHED_SPORADIC scheduling policy.
- Can be used to:
  - Process sporadic events at the desired priority levels.
  - At the same time guaranteeing that lower priority tasks are not affected.



THREAD SCHEDULING

- Three schedulers coexist:
  - Global schedulers:
    - Threads have global contention.
    - Thread scheduled against all other threads.
  - Local schedulers:
    - Threads only compete with the other threads of the process.
  - Mixed schedulers:
    - Some threads have global contention, others local.

Let us quickly look at these sporadic servers. I hope this for what do you mean by sporadic events you have already known earlier, in the earlier classes. So, POSIX 4 supports these schedulers, sporadic events, with using a policy scheduling policy called as scheduling sporadic.

POSIX 4 or real time POSIX supports a scheduling policy called as scheduled sporadic for the sporadic events. Let us see why this scheduling policy will be used. We are discussing about the scheduling policies is not it? Here I have already told these are these three schedulers coexist.

And we are discussing now the scheduling policy for the sporadic servers. Here I have already told you POSIX 4 supports a scheduling policy named as schedule or sporadic events or scheduled sporadic for the sporadic events. POSIX 4 supports a scheduling policy named as scheduled sporadic for the sporadic events, this schedule sporadic or this sporadic scheduling can be used to process the sporadic events.

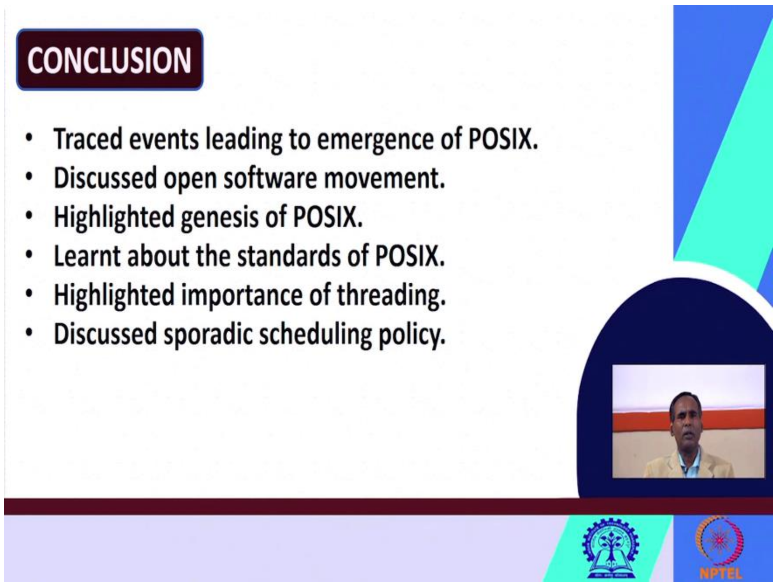
I hope you have already known as periodic events, aperiodic events, sporadic events, so this scheduling policy called as scheduled sporadic, it can be used to process the sporadic events where are the desired priority levels.

So, this scheduling policy named as schedule, scheduled sporadic this scheduling policy can be used to process this sporadic event at the desired priority level or the required or the interested or the intended priority levels.

At the same time, guaranteeing that lower priority tasks are not affected. In the same time this scheduling policy ensures that, guarantees that lower priority tasks are not affected. This scheduling policy, schedule sporadic, it can be used to process the sporadic events or the desired or intended priority levels. At the same time, it guarantees that the lower priority tasks, they are not affected.

So, this is how one scheduling policy known as scheduled sporadic this exists for the sporadic servers. So, this is how these sporadic events they can be scheduled using a policy known as scheduled sporadic.

(Refer Slide Time: 30:26)



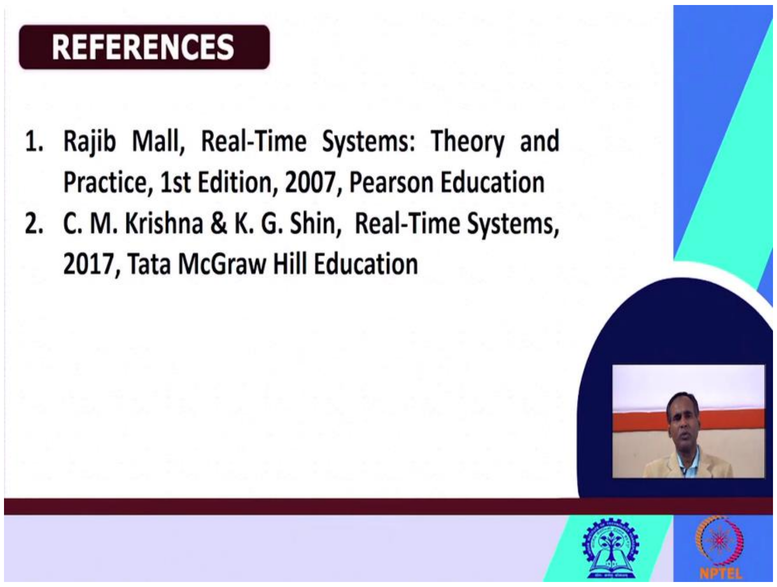
CONCLUSION

- Traced events leading to emergence of POSIX.
- Discussed open software movement.
- Highlighted genesis of POSIX.
- Learnt about the standards of POSIX.
- Highlighted importance of threading.
- Discussed sporadic scheduling policy.

So, today we have discussed about the emergence of POSIX or the genesis of POSIX. Today we have traced the events leading to emergence of POSIX. We have discussed the open software movement. We have highlighted the genesis of POSIX or we have highlighted the evolution of POSIX.

We have learned about the standards of POSIX. We have highlighted the different requirements for POSIX RT. We have highlighted the importance of threading in POSIX. We have also discussed the scheduling policy for sporadic events. These much things we have seen.

(Refer Slide Time: 31:01)



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

We have taken the references from these two books. Thank you very much.