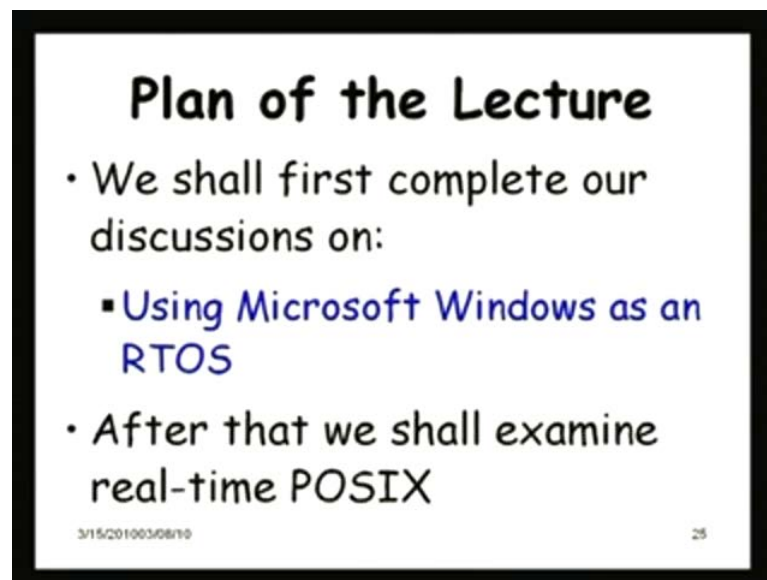


Real-Time Systems
Dr. Rajib Mall
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture No. # 25
Real-Time POSIX

Good morning. So, we will get started today, last time we were discussing about Windows and UNIX, and their use in real-time applications what difficulties will arise. So, today we will complete our discussion on using Windows and UNIX on real-time applications, we had left some parts, and then we will look at real-time POSIX which has become a standard in real-time operating systems. **Actually**, initially, it was designed for UNIX based operating systems, but now it is a standard for all operating systems; so, let us proceed.

(Refer Slide Time: 01:07)



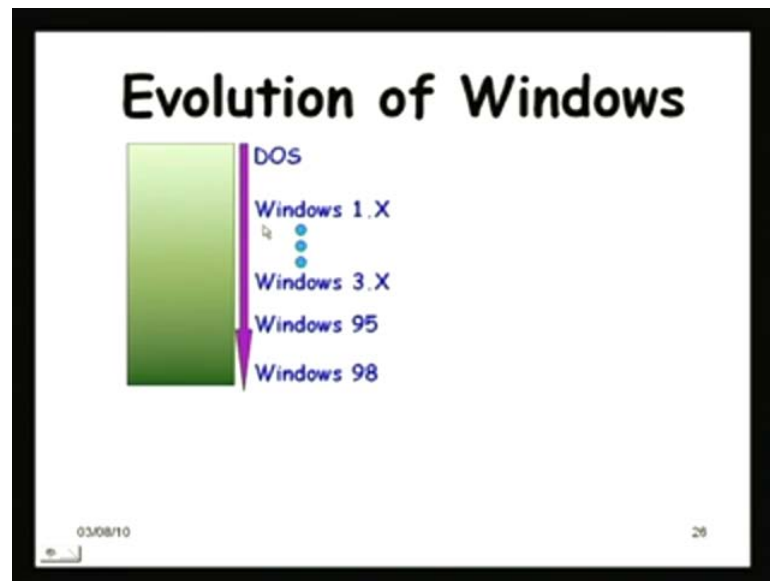
Plan of the Lecture

- We shall first complete our discussions on:
 - Using Microsoft Windows as an RTOS
- After that we shall examine real-time POSIX

3/15/2010 3:08:10 25

So, today we will first complete our discussions on using Windows on as real-time operating system, we had looked at UNIX, and then we will compare the Windows, and UNIX features for uses or real-time operating system, and after that we shall start discussing about real-time POSIX.

(Refer Slide Time: 01:30)



So, yesterday we were discussing about Windows and their evolution; we had seen that started in 1982, as a simple disk operating system, and kept on evolving, and major event was a graphical interface Windows 1.X, single tasking, and simple operating system, and slowly it evolved, and more and more features were incorporated.

And the code finally, become unmanageable, because it was basically a maintenance on some old code successive maintenance, the structure of the code degraded, and the code was discontinued Windows 98, but by that time, they had started the Windows NT, this was a freshly written new code, and as we were pointing out, it is a microkernel based operating system, the idea is to tackle the bugs.

Earlier we were saying I think Windows 95 10000 known bugs, but could not be debugged. So, Windows NT has become stable, it is a stable operating system does not crash, as Windows 95 and 98 used to do and that is keeping on evolving.

Now, let us see how this NT is suitable for real-time applications, the earlier series there was no question of using it in real-time applications, because it was unstable.

(Refer Slide Time: 03:19)

Microsoft's Windows

- Windows code was completely rewritten:
 - Windows NT system was developed.
 - More layered than microkernel design
- Windows NT system:
 - Much more stable (does not crash) than the earlier DOS-based systems.
- The later versions of Windows:
 - Descendants of Windows NT
 - The DOS-based systems were scrapped.

So, the Windows code was completely re written, and the the NT system was developed, and even though it is microkernel based design, but we will see that, it is more of a layered design than a microkernel design, we will, we will see that. It is much more stable, and all the later versions, these are descendants of the Windows NT code, and the dos based code was scrapped.

(Refer Slide Time: 03:58)

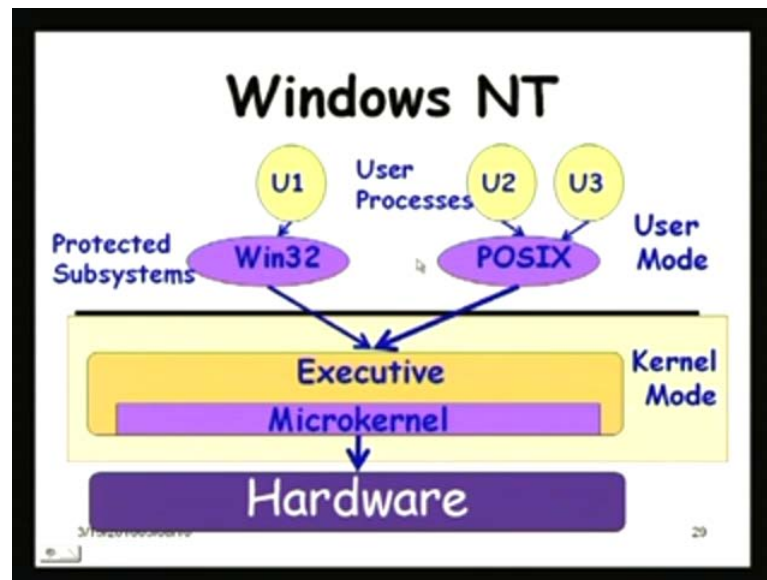
Windows NT

- NT Microkernel provides base services
- Executive:
 - Builds on base services to provide user-level services
- User-level services used by:
 - Privileged subsystems (parts of OS)
 - True user programs

So, let us see the Windows NT the main features, and let us examine how suitable it is for real-time applications. So, the microkernel here provides the basic services, and on

top of the microkernel we have the executive, which builds on the basic services to provide user level services, and these user level services are used, both by the privileged subsystem which is part of the OS and also by user programs.

(Refer Slide Time: 04:37)



So, if we look at the structure of Windows NT, we will see that in the kernel mode, we have the microkernel, and on top of, that is the executive.

In appear microkernel design, we were discussing, I think last time, that the kernel mode only the microkernel operates, and all other operating system components, they operate in user mode, that is, the basic of a microkernel based design; so, it is not purely a microkernel design.

So, you have design, see the executive also is operates in kernel mode, but it provide the higher level services than the microkernel, and the microkernel is the one which interacts with the hardware, and there are two types of interfaces here, the win 32 interface and the POSIX. We will, we will discuss about POSIX today, that is the main focus of today's discussion and the user level processes can run using both this interfaces.

(Refer Slide Time: 05:46)

NT Executive

- Provides higher level services than the microkernel.
- Runs in kernel mode:
 - But separate from the microkernel itself.
 - Makes it easier to maintain
- Built of independent modules:
 - All preemptible and pageable

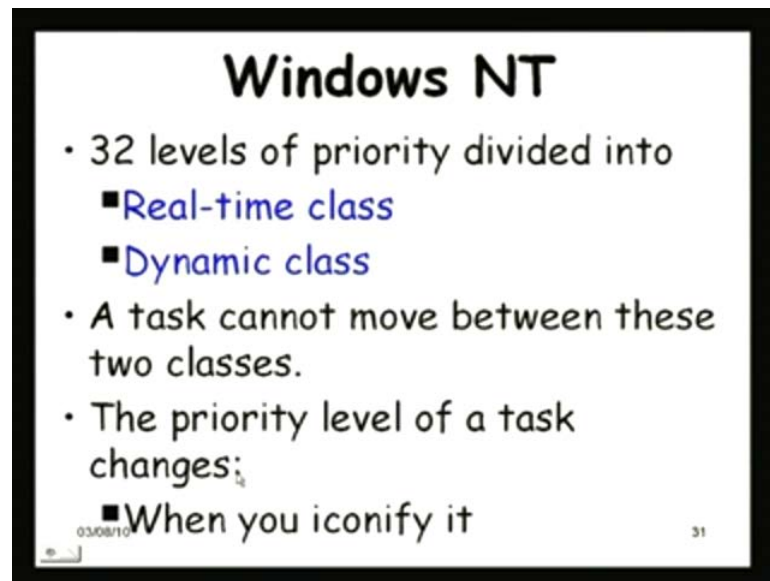
3/15/2010 03:08:10 30

So, the executive as we are saying, it provides higher level services than the microkernel, and the executive also runs in the kernel mode, which is not a purely microkernel based design, but the executive is completely separate from the micro kernel itself, it just calls the microkernel.

But still, because these are two separate things, the microkernel and the executive, it is easier to maintain, unlike the earlier code dos and from that 95, 98 which was a monolithic code, here is more layered here, and we have, but still both of them operating the kernel mode, and the executive is actually pre-emptible and page able, whereas the microkernel is not.

So, that is a difference, here also in the features of microkernel and executive, the microkernel is memory resident, it is loaded they start whereas, the executive is page able and also pre-emptible.

(Refer Slide Time: 07:10)



So, it provides, 32 levels of priority, which are divided into three classes of priority, sorry two, two classes dynamic and real-time class, and we also have a ideal class, we will come to that later.

So, there are two main classes here, the real-time and the dynamic class, and we know what is a real-time priority level is not it, what is a real-time priority level anybody would like to answer, what is a real-time priority level, and what is a dynamic priority level?

(()).

Sorry.

Static.

Static and dynamic.

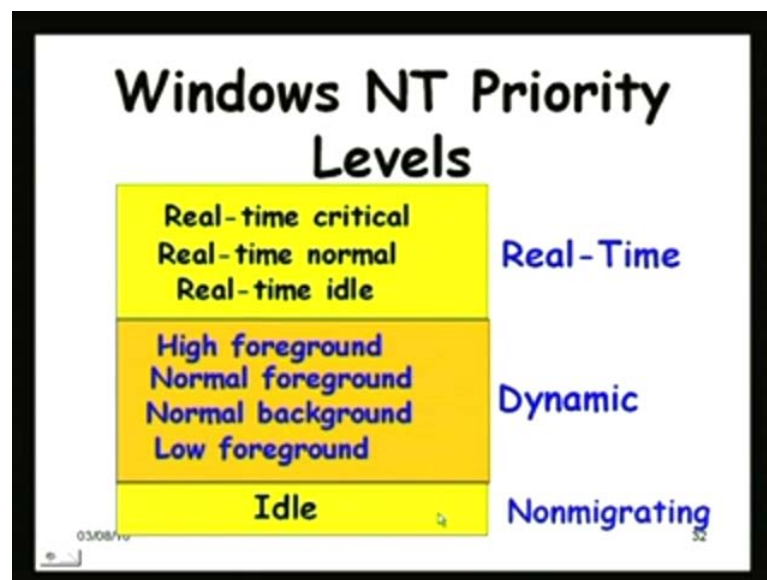
Change (()).

So, the real-time priority levels, they do not change with time, and these priority levels are assigned to the real-time tasks, where the other is throughput is not a concern, the main concern is meeting task deadlines, whereas in a dynamic class, the tasks that run here, they do not have deadlines, and the concern here is, other is throughput.

So, the operating system keeps on changing the priority of the tasks that we are discussing yesterday, and once a class, a task is assigned either of these priorities, it cannot move between these two. So, once it is a real-time priority, it will continue to be real-time priority, and for the **the** dynamic class, the priority level of the task keeps on changing.

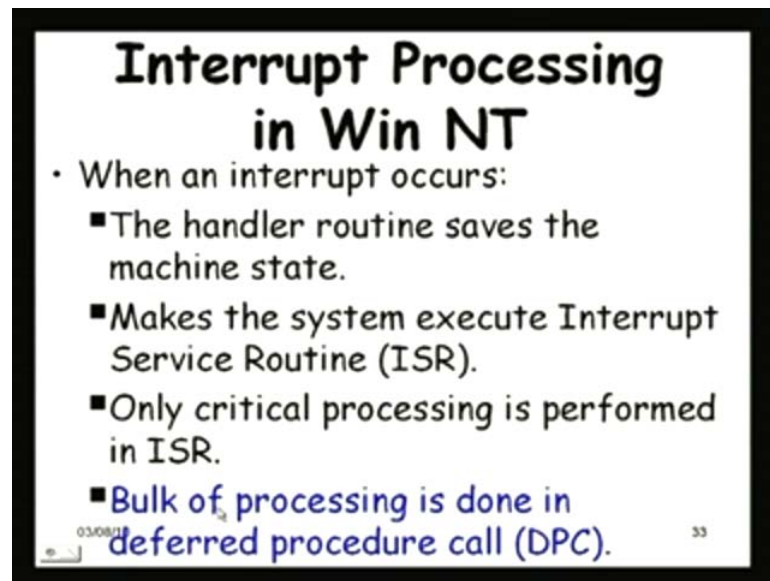
For example, if there is a task running iconifies, there will be a change in the priority, and also, it depends on whether it is using a CPU time or is doing IO, and besides that, if you iconify also, it might change its priority, it will change its priority.

(Refer Slide Time: 09:21)



Now, let us see the priority levels supported by NT. So, there are basically two main cut ever is a priorities. The real-time, there are sub classes, here real-time, idle normal and critical, we can assign priorities at this level, and in the dynamic high foreground, normal foreground, normal background, and low foreground, and within these the task, these ban the priority, the task operates unless you change a background task into foreground, and so on, and the lowest priority is the idle task, and this is a non migrating priority again.

(Refer Slide Time: 10:20)



**Interrupt Processing
in Win NT**

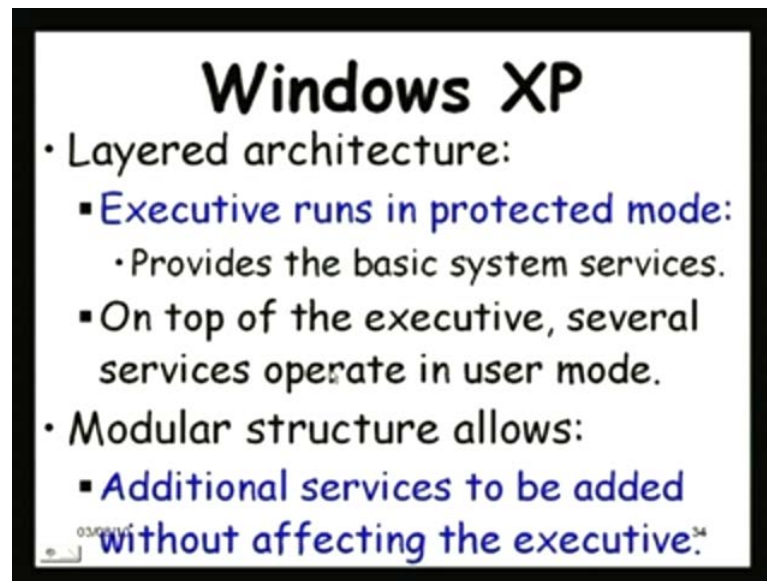
- When an interrupt occurs:
 - The handler routine saves the machine state.
 - Makes the system execute Interrupt Service Routine (ISR).
 - Only critical processing is performed in ISR.
 - Bulk of processing is done in deferred procedure call (DPC).

03/08/17 33

It is, like, it is also a static priority does not change here. Now, one important thing that we need to discuss is about interrupt handling in Windows NT, we will see that the feature that we are looking for, we were discussing is required for a real-time operating system is present here, but still there is a small problem here, let us see how the interrupts are handled here. So, once interrupt occurs, the handler routine saves the Machine state, and makes the system execute the interrupt service routine.

But in the interrupt service routine, only the critical processing is performed, and the bulk of the processing is cubed as a task in the form of a deferred procedure call or a DPC, so we are saying that, this is required for quick response to interrupts.

(Refer Slide Time: 11:24)



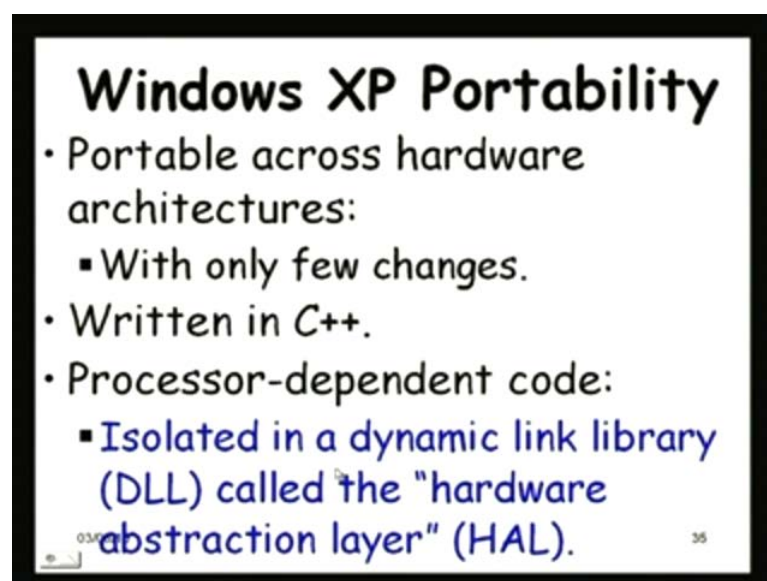
Windows XP

- Layered architecture:
 - **Executive runs in protected mode:**
 - Provides the basic system services.
 - On top of the executive, several services operate in user mode.
- Modular structure allows:
 - **Additional services to be added without affecting the executive.**

03/09/11

So, that feature is there. Now, let us look at the XP derivative again of the NT. So, layered architecture executive runs in protected mode, just like their NT provides the basic system services, and on top of the executive, several services operate on user mode and additional services can be added without affecting the executive, so which will operate in the user mode. So, based on the executive, more services can be added which will operate on the user mode.

(Refer Slide Time: 12:09)



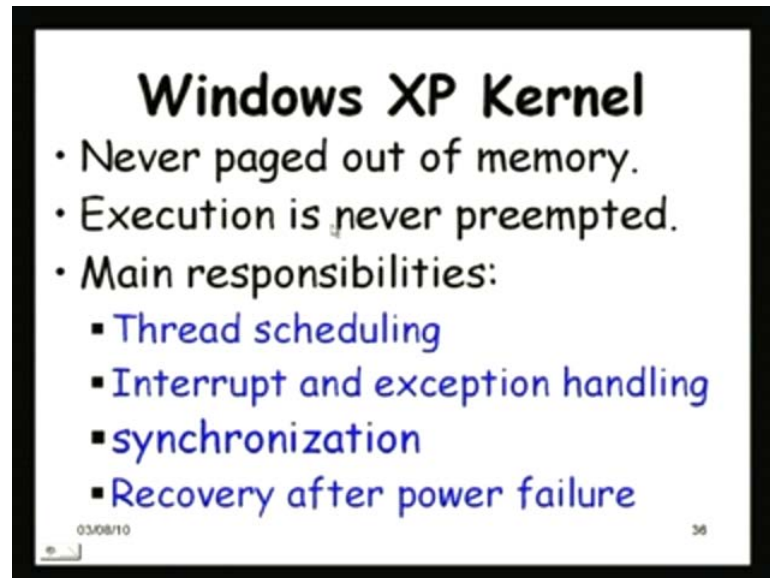
Windows XP Portability

- Portable across hardware architectures:
 - With only few changes.
- Written in C++.
- Processor-dependent code:
 - **Isolated in a dynamic link library (DLL) called the "hardware abstraction layer" (HAL).**

03/09/11 35

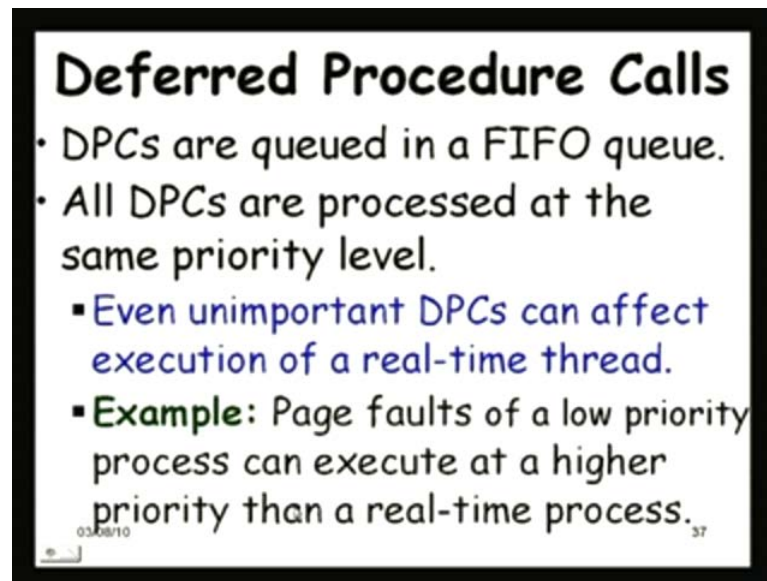
It is portable across hardware architectures with only few changes written in C plus, plus the processor dependent code, the microkernel basically is isolated in a dynamic link library called as a hardware abstraction layer.

(Refer Slide Time: 12:33)



Again the same feature is there, as NT the kernel is never paged out of memory, and its execution is not preempted, and the main responsibility of the microkernel is thread scheduling interrupt and exception handling synchronization, and recovery after power failure, and based on this the executive and the user tasks operate, similar feature just like n t, because after all, it is a derivative of NT; so, let us not spend time on discussing this.

(Refer Slide Time: 13:09)



Deferred Procedure Calls

- DPCs are queued in a FIFO queue.
- All DPCs are processed at the same priority level.
 - Even unimportant DPCs can affect execution of a real-time thread.
 - **Example:** Page faults of a low priority process can execute at a higher priority than a real-time process.

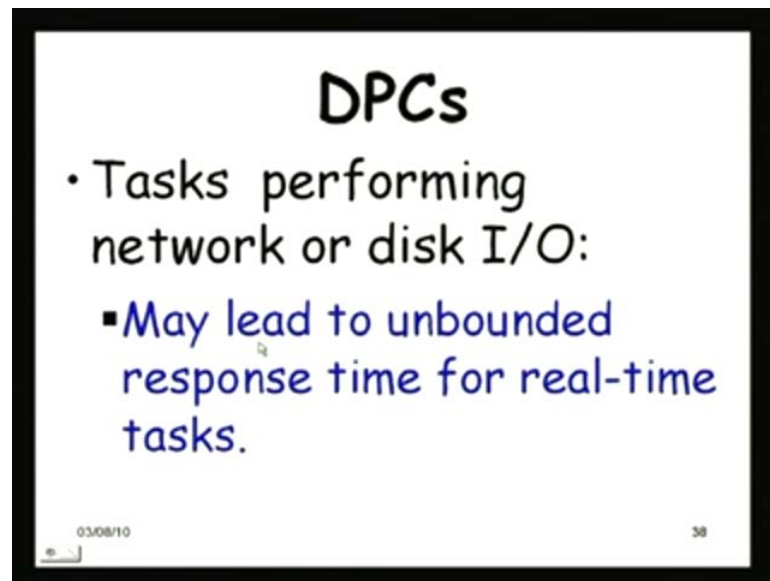
03/08/10 37

Now, let us look at the feature of the deferred procedure call, they said that the interrupt handling is much better than UNIX, the basic UNIX here the bulk of the processing is done in deferred procedure call, but let us examine it further.

So, once the interrupt is handled, the deferred procedure calls are queued in a FIFO manner, in a FIFO queue, and one catch here, just see here the DPCs are processed at the same priority level irrespective of which task or the interrupt corresponds to which task all of them are treated in the same manner or in other words, even unimportant DPCs can affect execution of a real-time thread, because these are independent of task priorities. So, the DPCs for a real-time thread will operate at the same priority as some unimportant task, for example, some e-mail or something or may be a disk I O, just logging something onto disk, they will operate at the same level this. Just an example, here the problem that might occur, the page faults of a low priority process, like logging or something can execute at a higher priority than a real-time process.

So, this is one of the main problems that you would face, if you want to use the NT or its derivatives in a real-time application.

(Refer Slide Time: 14:59)



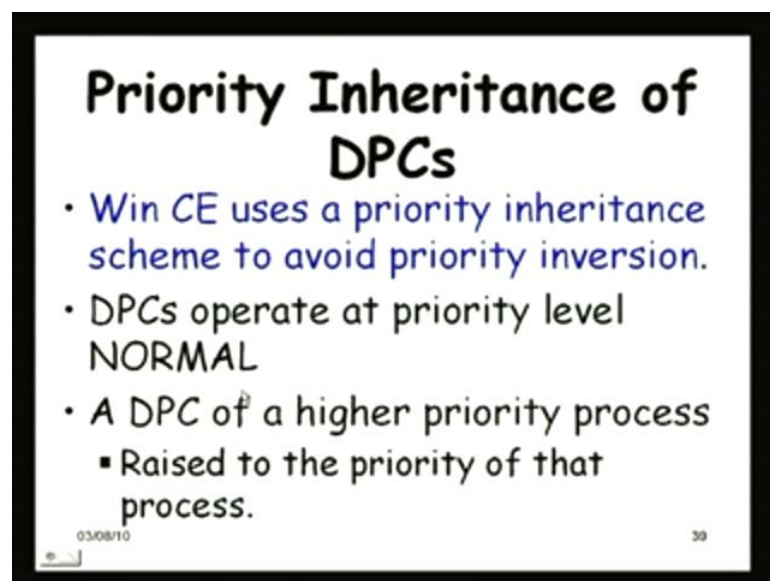
DPCs

- Tasks performing network or disk I/O:
 - May lead to unbounded response time for real-time tasks.

03/08/10 38

For example, if we have tasks performing network or disk I/O, unimportant tasks, then for the real-time tasks, they will translate into unbounded response time. So, using the even that NT series has a many good features, phrasely designed microkernel based, and DPCs is and so on, but still there are problems here, which can which prevent its use in hard real-time applications.

(Refer Slide Time: 15:39)



Priority Inheritance of DPCs

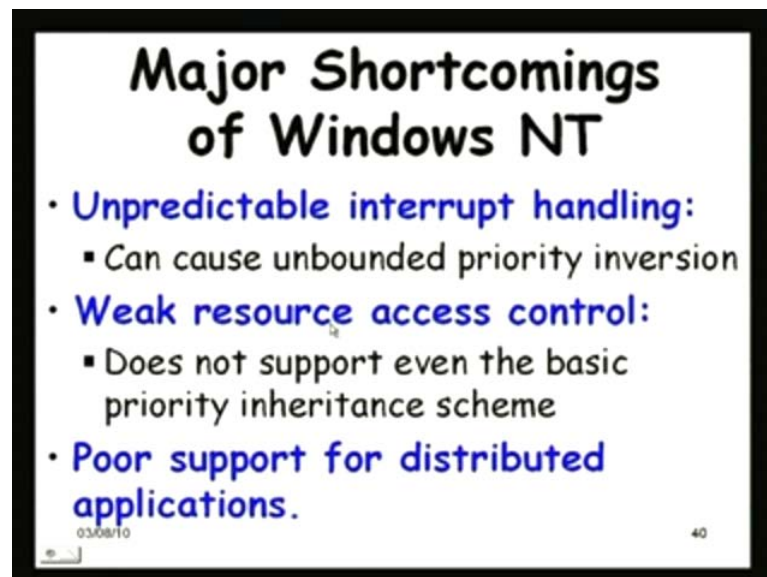
- Win CE uses a priority inheritance scheme to avoid priority inversion.
- DPCs operate at priority level NORMAL
- A DPC of a higher priority process
 - Raised to the priority of that process.

03/08/10 39

But the Windows e which we will discuss some extent later, which is used in embedded application, this provides a priority inheritance of the DPCs.

So, we had a look at the priority inheritance earlier, that a DPC of a higher priority task, so we inherit the priority of that task; so, typically the DPCs operate at a priority level normal, and if the task has higher priority than normal, then the DPCs will inherit the priority of that process.

(Refer Slide Time: 16:27)



So, the major shortcomings of Windows NT for real-time applications is unpredictable interrupt handling, because of this DPCs operate at the same level, weak resource access control; so, it is still possible for tasks to use each other resources, that is inherited from the dos series poor support for distributed applications, and the unpredictable interrupt handling we had seen that, it can cause unbounded priority inversions, and weak resource access control does not support, even the basic inheritance scheme.

(Refer Slide Time: 17:23)

Windows NT versus Unix		
Real-Time Feature	Win NT	Unix V
DPCs	Yes	No
Real-Time Priorities	Yes	No
Timer Precision	1msec	10msec
Asynchronous IO	Yes	No
Memory locking	Yes	No

So, there is no real-time resource, here in is not supported, but if you compare NT and UNIX for real-time applications, we will see that the DPCs are supported by Windows NT. So, the response time interrupt response time is much better here compare to UNIX system 5.

There are real-time priority classes, we had seen that, where as UNIX there are no real-time priorities, all are dynamic priority levels, timer precision is 1millisecond UNIX 10 millisecond, asynchronous I O supported in n t, not supported in UNIX five, we know all what is a asynchronous I O is not it.

Memory locking is supported. So, this is an important feature to reduce the g t r in memory access, if there is a page fault, then the memory access time can be really large and for real-time tasks, that would be unacceptable. So, the memory locking facility is also provided in Windows NT; so, has many positive features, so it is much suited.

Memory locking reduces (()).

So, I think let me just a elaborate on that, see last time we discussed about that since you are asking, we will let me just elaborate on that. So, we said that, if the data or the page that is, that is needed is resident in the memory, then it just you get it instantly is not it.

But if there is a page fault, then you have to fetch it from the hard disk. So, the time can be a second or may be a fraction of a second, so the real-time tasks, they can lock certain parts of their memory, which will never be phased, and they can even lock their entire address space; so, that gives a deterministic memory access time.

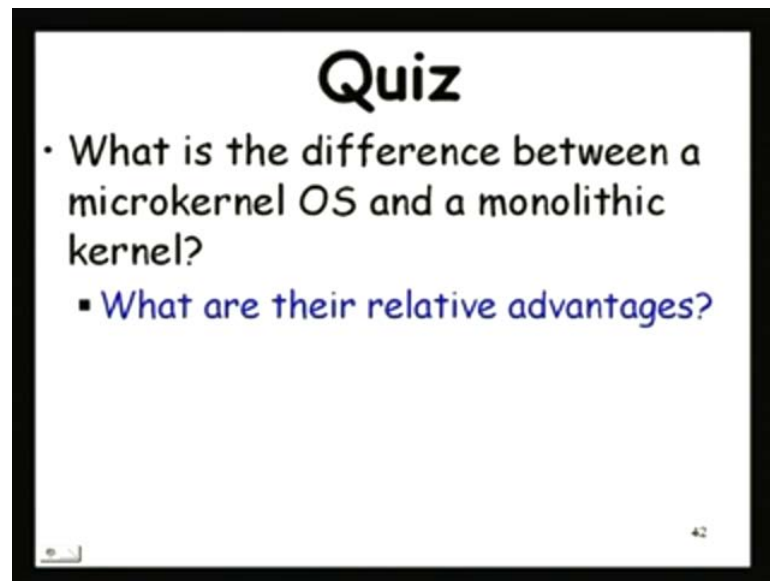
(Refer Slide Time: 19:33)

Windows NT versus Unix		
Real-Time Feature	Win NT	Unix V
DPCs	Yes	No
Real-Time Priorities	Yes	No
Timer Precision	1msec	10msec
Asynchronous IO	Yes	No
Memory locking	Yes	No

So, as I we can see from this comparison table, that if you are thinking of a real-time application, the NT is much suited, then UNIX **as it is**, but we will see that, again the Linux has a good features. So, this is the UNIX system 5, which has designed to be used in traditional applications and that to in server.

So, naturally, it **is**, was not designed for real-time application and it does not support those features.

(Refer Slide Time: 20:17)



So, let us just try to answer one or two questions. So, what is the difference between a monolithic kernel and a microkernel?

Monolithic kernel as entire thermal, entire thermal into the main memory, micro thermal has only basic kernel.

Anybody would like to answer differently.

There is a driver has only basic services in kernel mode.

Exactly, kernel mode; so, in the kernel mode, the basic services are provided in the microkernel based operating system, and all other operating system services are built on the microkernel operating system, the microkernel, but those operated the user mode, whereas **a**, in a monolithic kernel, all the operating system services are provided in the operating the kernel mode.

But what are the relative advantages?

(()), small part, small part resident.

But how does that help.

Easy, easy to maintain.

I mean, why is it easy to maintain?

That is only small part is ((.)).

Sorry.

((.)).

No, why is, see microkernel based operating systems are easier to change and extent true, but why?

((.)) sir, they will have the layered architecture and possibility.

No, even the traditional the monolithic kernels are also layered architecture, it is not that everything operates at the same layer; layered architecture is not the answer.

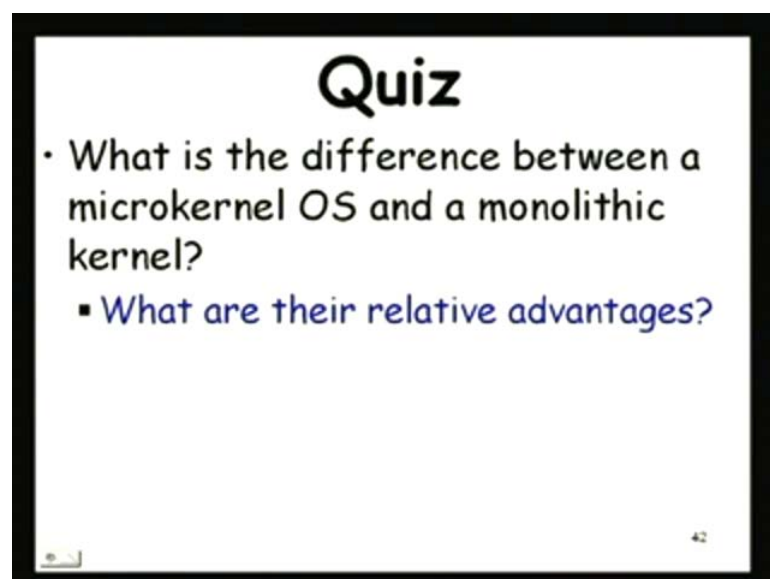
((.)).

Why, why debugging is easier, yes, correct debugging, sorry.

((.)).

Not really see anything that operates in the kernel mode debugging is difficult, because a kernel mode task can change anything, any data anywhere.

(Refer Slide Time: 22:28)



Quiz

- What is the difference between a microkernel OS and a monolithic kernel?
 - What are their relative advantages?

42

So, and also you cannot use a debugger in a kernel mode. So, debugging a kernel task is always difficult, and this is the small microkernel, which can be developed debugged and extended maintained etcetera, whereas in a monolithic kernel, just see everything operates the large operating system, millions of lines of code in the kernel mode, and it is really a nightmare to debug a monolithic kernel.

So, one is about developing maintaining etcetera, micro kernel is easier, because most of the services operate at user level, like tradition, like a normal application, but what are the, any other advantages of the micro kernel approach?

These are operating in two modes, which can make one mode as protective.

Not clear, can you elaborate your answer.

The present mode is protective mode.

Yes, it is protective, it is same in monolithic.

(()).

No, kernel mode, so, you are saying that, even without using kernel level locks, we can just disable the interrupts in a micro kernel, and still not have difficulty is it possible.

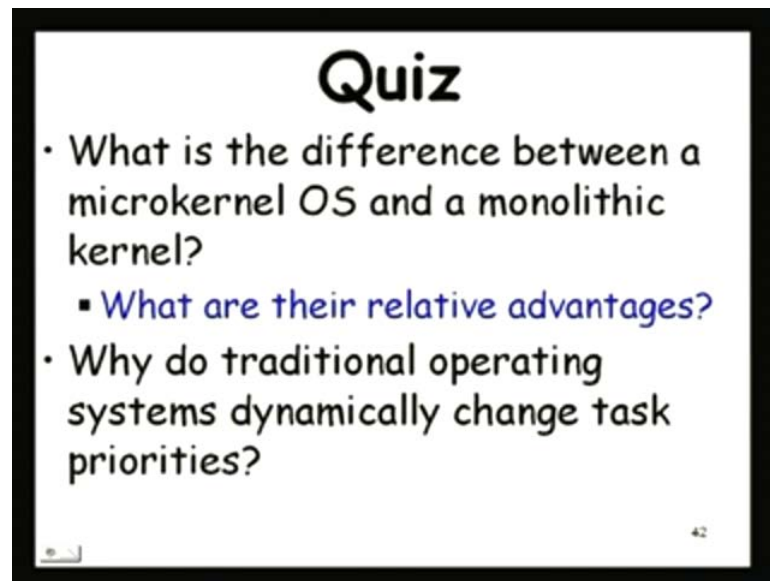
So, he says that in the micro kernel, we do not need the kernel level locks, because after all this provides only the basic services and that it does not take much time possible.

Yes, we will see that some of the operating systems, micro kernel based ones, they do not use locks, any other, any other advantage of a micro kernel operating system.

(()), that, that operate in user mode. So, it might help in a easy accessing of that (()).

No, see any service of, finally, the user mode component, after this is no component has to invoke the micro kernel. So, there has to be a mode change, it is not that you cannot, can give services without mode change.

(Refer Slide Time: 24:17)



So, **so**, another advantage can be a lower interrupt response time, because of the low code overhead, but what are the disadvantages?

It has to make two calls.

It is inefficient, the microkernel operating systems are inefficient, because it makes two function calls, two calls, one to the user level process which in turn calls.

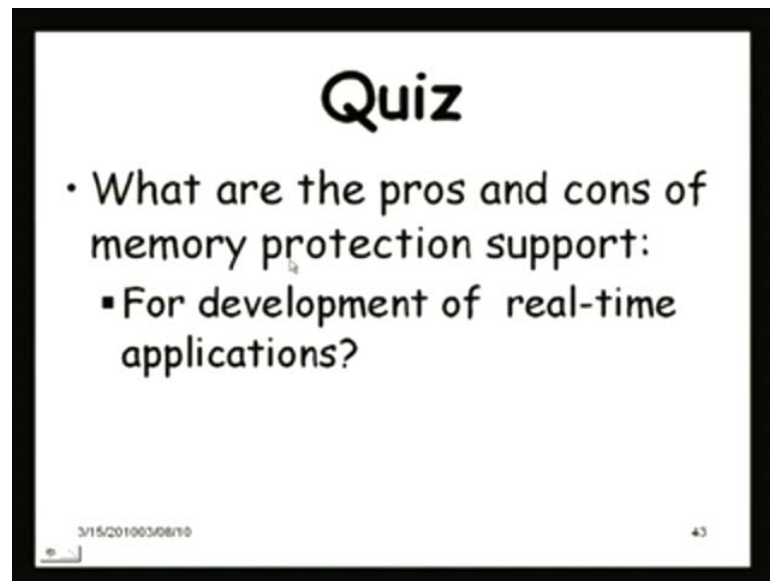
Kernel.

Kernel, the micro kernel, and also the mode change the more; here, now, let us go to the next question, why traditional operating systems dynamically change task priorities.

(()).

Exactly they change the task priorities at the every time slice, at the end of every time slice, because the I O bound processes have to given a high priority. So, if something is doing I O, its priority keeps on increasing, and if it is a CPU bound task, its priority keeps on decreasing the idea is that, the I O channels should be kept busy; so, that the other is response time improves.

(Refer Slide Time: 26:33)



So, let us look at this question, what is the advantage and disadvantage of having a memory support, memory protection support in a real-time application, advantages I think it is simpler question, what is the advantage of having memory protection support in a real-time application.

((.)).

No, many protection not memory locking memory protection, basically one task cannot access the memory, the data and the code of another task.

It is consistence.

It is easy to develop advantage is that, a, if there is a memory protection, it becomes easier to develop debug.

There is no memory protection; you do not know which data it corrupts, but what about the disadvantage, because we will see that, there are some operating systems which real-time, which do not provide memory protection. Then, for each task, we need to maintain separate its own memory that has to be bounce between ((.)).

No.

((.)).

No.

(()).

That is a disadvantage you are saying is it.

(()).

No, not like that, because, see if you make it work, once it will keep on working, it is not that you need to over head, each time you do something not necessary. So, any other advantage or disadvantage, let us look at the disadvantage, advantage, you are said easier to develop, what about disadvantage of memory protection.

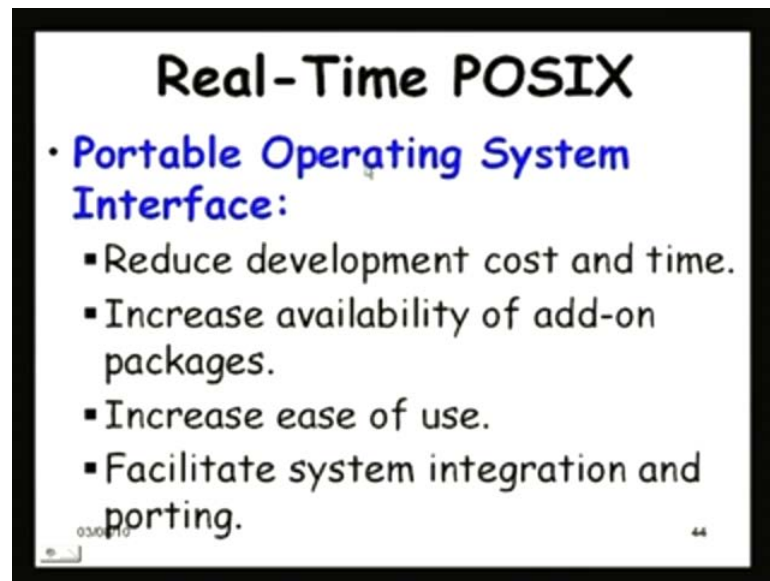
Sir, data sharing of the (()).

No, data sharing will be supported, that is irrespective of memory protection, that is, that is supported, as a I mean, see memory protection means you can also disable in memory protection, we can alive another tasks to access that is.

(()).

No, not really, so the simple thing not that difficult teach, that there is a overhead, each time will the operating system, checks about the protection data, you know the traditional operating system, a virtual memory environment, the protection data, the protection beat is checked, the protection beat is checked is not it.

(Refer Slide time: 29:09)



So, now let us look at the real-time POSIX, a POSIX stands for portable operating system interface, but that gives POSIX I is not it, POSIX but what about x.

((.)).

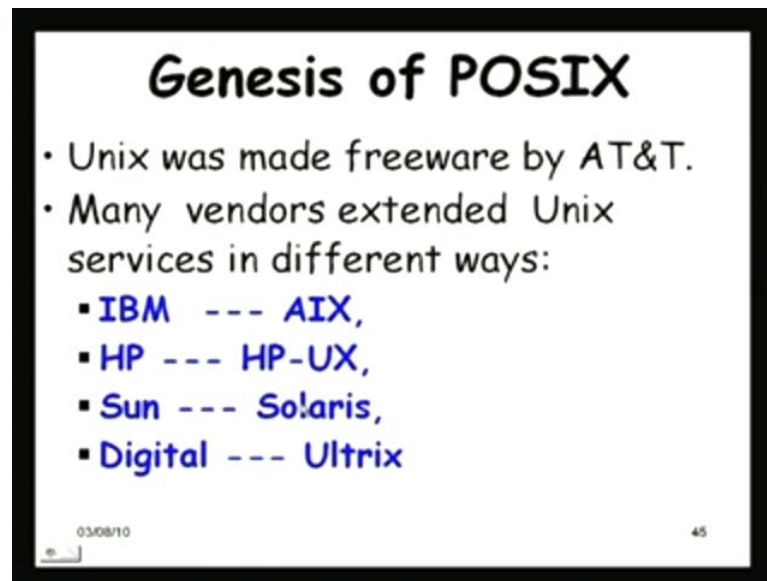
I think some of you know, this, this is, was developed to be a unique standard and to make it sound like a UNIX standard POSIX x was added here, portable operating system interface.

So, the idea here was to reduce the development cost and time, we will see why and how increase the availability of add on packages, increase ease of use and facilitate system integration and porting, that was the basic objective with which the POSIX was formulated.

To reduce any application development cost and time, later I will ask you how once we look at the real-time POSIX standard, how will this lead to reduce development cost and time how it will increase the availability of add on packages, how it will increase the ease of use, how it will facilitate system integration and porting.

So, just have these questions in mind, because I am going to ask you, after we look at the POSIX features just going to ask you, that how this occur, how this happen.

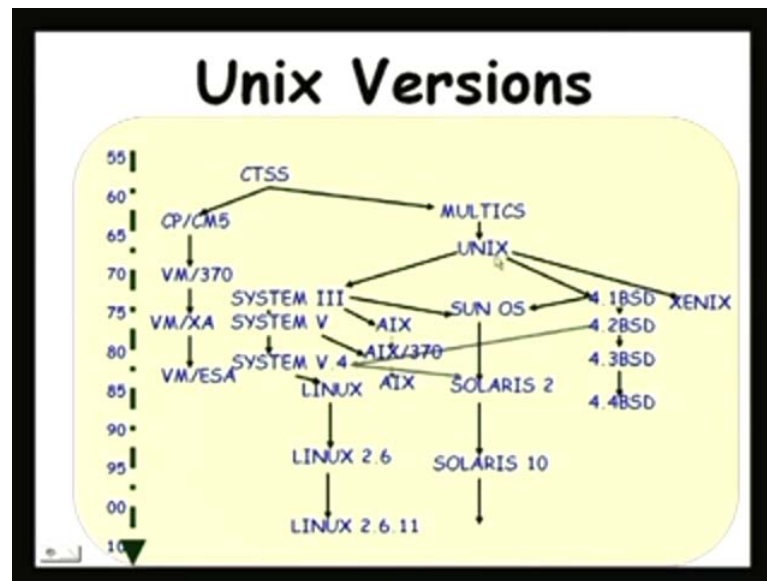
(Refer Slide Time: 31:02)



Let us look at the genesis of POSIX or the origin how the POSIX came about we had seen, so far that UNIX was developed at AT and T 1968 or 69, and those made a freeware 975, a source code was given away, and since the source code was available many vendors extended UNIX in different ways. IBM had its AIX operating system at derivative of UNIX, and if they added several other features, they thought appropriate, HP, HP-UX. Many features they thought appropriate, they added to the basic UNIX. A sun, they had the sun OS, and then, from that the Solaris which is again a extension of the UNIX in their own way. The digital electronic corporation, they had their Ultrix operating system, again a derivative of UNIX extended, whatever they thought; they are appropriate for, then they extended UNIX that way.

The sco, with the sco UNIX, but the thing is that, even they all are UNIX derivatives, they an application written on AIX will not work on HP X, because they use different services, not only this semantics, the syntax of the service is different, I mean, some services are not supported other operating system. And even the Solaris are supported, the syntax may be different or the semantics may be different.

(Refer Slide Time: 32:54)

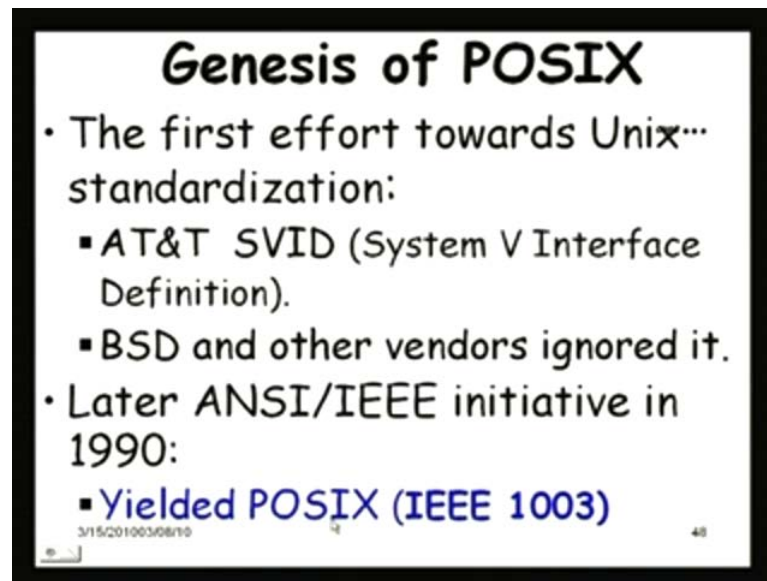


So, if you look at the UNIX versions, the UNIX had its origin in MULTICS, and which is a derivative of the CTSS operating system, time sharing, first time sharing operating system developed at M I T lab, it is not a commercial operating system, it is a educational institute developed it, and based on that, all these are derivatives..

See here from UNIX we had this BSD developed at university of California Dockley, the Genix, the sun SOLAREIS, the Linux, the **the** virtual memory operating system v n at IBM, and so on. So, again some of these arrows, I have not shown these are more complicated arrows, they have taken features from each other.

But the end thing is that, **they have become**, they have extended the basic UNIX here in very different ways, and they have become inconsistent, application written for Solaris will not run on Linux, will not run on system five, will not run on V M, so, that was the situation.

(Refer Slide Time: 34:32)



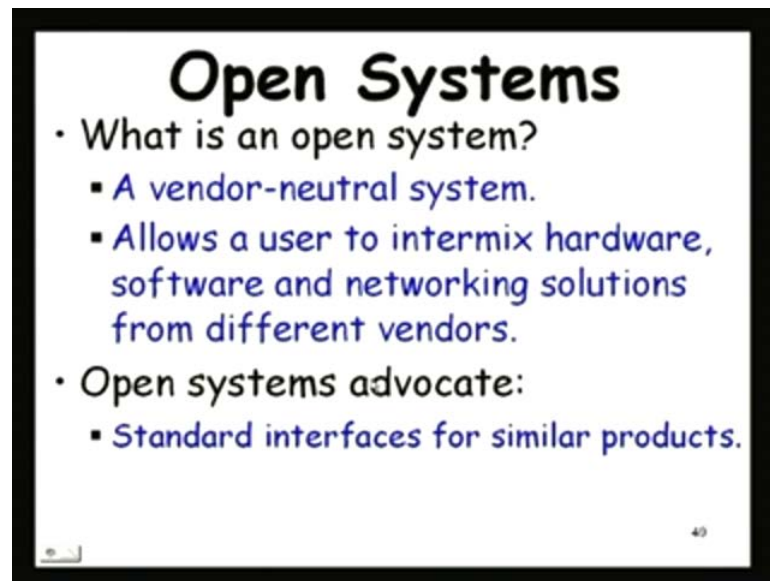
So, program written on one UNIX, one version of UNIX will not run on another, and the different versions offered different services. So, some services are supported in some versions, some services not supported, the syntax was different even the semantics was different and the need for a standard UNIX was recognized.

So, the first effort towards UNIX standardization was done by AT and T, so they found that even though they had made UNIX freeware, one source code they had distributed, but it has developed in many different ways.

So, they started this initiative system five interface definition, they said see since the UNIX was started by them. So, they said their own system five with of course, suggestions from others, other vendors of UNIX should become a standard, but the SVID was ignored by other vendors, even BSD and other vendors, they ignored this SVID initiative. They said see we have developed in our own way. So, there is no point why we should follow your own system five interface definition.

Later the ANSI, under ANSI and IEEE initiative in 1990, the POSIX movement came, the POSIX has become an IEEE standard, IEEE 1003.

(Refer Slide Time: 36:20)



But before we understand the POSIX, let us see what is a open system, what do you think is a open system, must have heard about open systems.

The context of computer, yes, anybody would like to say, what is a open system.

(()).

That is open source, anybody are tell would like.

(()).

Sorry.

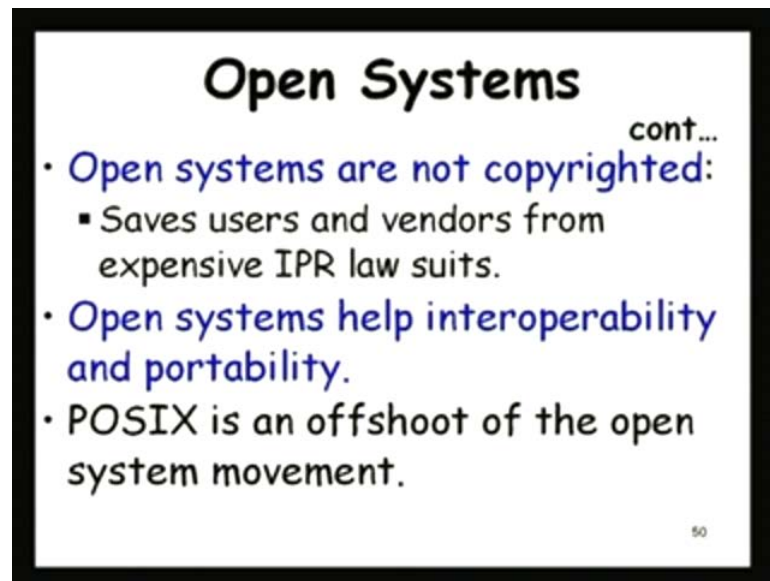
Flexible hardware support.

Flexible hardware support, what sort of flexibility.

That any kind of diverse can be (()), the minimal support on the (()).

Minimal support no not very convincing does not appear, anyway let say proceed, because these are important terms, you hear them once in a while, even in newspapers or in general conversation; so, let us see this.

(Refer Slide Time: 36:20)



The slide is titled "Open Systems" in a large, bold, black font. To the right of the title, the text "cont..." is written in a smaller, italicized font. Below the title, there is a list of three bullet points. The first bullet point is "Open systems are not copyrighted:", followed by a sub-bullet point "Saves users and vendors from expensive IPR law suits." The second bullet point is "Open systems help interoperability and portability." The third bullet point is "POSIX is an offshoot of the open system movement." The slide has a black border and a small number "50" in the bottom right corner.

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.

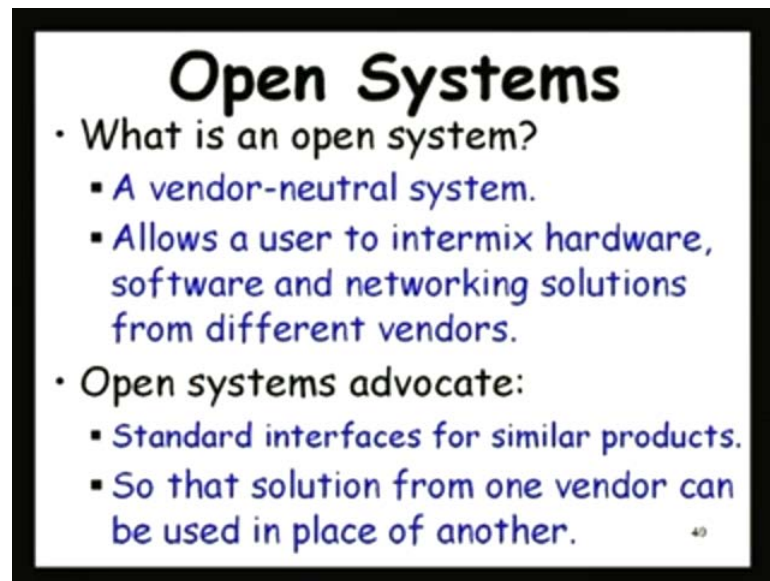
50

An open system is a vendor neutral system; it allows the user to intermix hardware software and networking solution from different vendors. So, if you have an open system, we can get part of one part from one vendor, another part from another vendor and so on, the user can use that can easily integrate this, they talk to each other, but if it is a not a open system, then they have to buy a specific component from specific vendor, other vendors cannot even manufacture.

So, there has to be standard interfaces for similar products, these are the advantages is not it, if there is a standard interface for similar products, then there is a advantage for example, just a common example, let us say, you have a bicycle, and know the knots, and or let us say the paddle, if the paddles of a different manufacturer is different know, you have to buy only that paddle, that will work on that cycle, other if you buy any other paddle, it will not work.

So, then it becomes difficult for the user is not it, and the cost also the user pays more, because that manufacturer he will hike his price; now, there are many manufacturers who manufacture that paddle; so, the cost will be less.

(Refer Slide Time: 39:04)



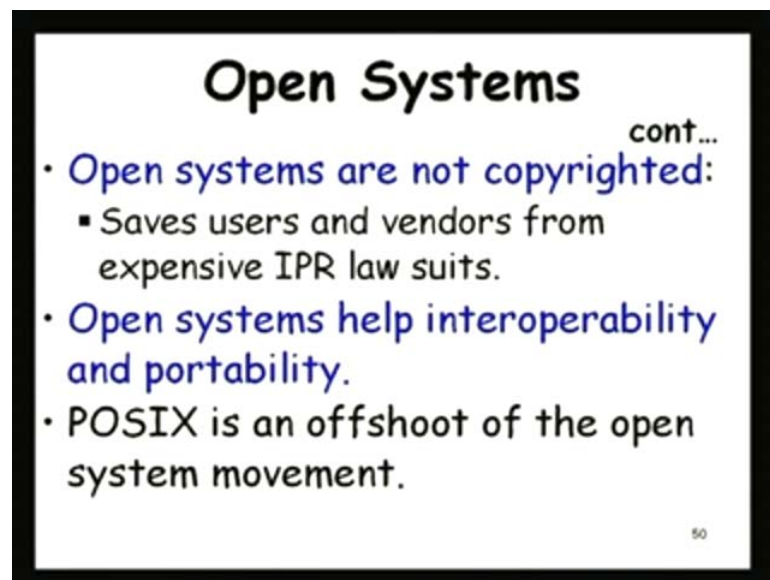
Open Systems

- 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.

49

So, the solution from one vendor can be used in place of another very powerful thing actually open systems.

(Refer Slide Time: 41:20)



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.

50

I will just tell you a small incident after we look at this. So, it is clear that open systems are not copyrighted, anybody can manufacture components with a specific interface, and saves by users and vendors from expensive IPR law suits, because if the interfaces were copyrighted, what would happen is that, if the user brought a component manufactured

by some other vendor or some vendor manufactured according to that interface, there will be a problem.

The open systems help interoperability of components manufactured by different vendors, and also portability, you can use different components in different manufactured, in different situations, for another new application that is, what is portability. And we will see that POSIX is a offshoot of this open system movement, I just wanting to tell you one incident, how the open system is a very powerful concept.

Before the PCS, and at the time the PCS came 1980 or so, the most popular desktop computer was a Macintosh computer Mac, it was called as Mac from apple, and you would find at least not that time, in India computers were not. So, popular, but if you go to western side, you will see that every secretary, everybody has a Mac on his desktop.

So, starting from secretarial job to everything they used to do on the Mac, but the Mac is a closed architect, they do not know what is there inside, and the hardware, the software etcetera nothing was known, but that time, the PC which came in 1982 or 81, it appeared like a ugly box in front of Mac, Mac had a nice user interface, graphical good sound multimedia, but the PC appeared like a ugly box, large one huge Mac was a small one, huge box and used to make very ugly sounds, but in four or five years, now you will think that time let us see these boxes are of no comparison with Mac, these will just die out, but in four or five years time, the Mac became absolute, what is the reason? The PC was a opens hardware right, you can just plug in a card.

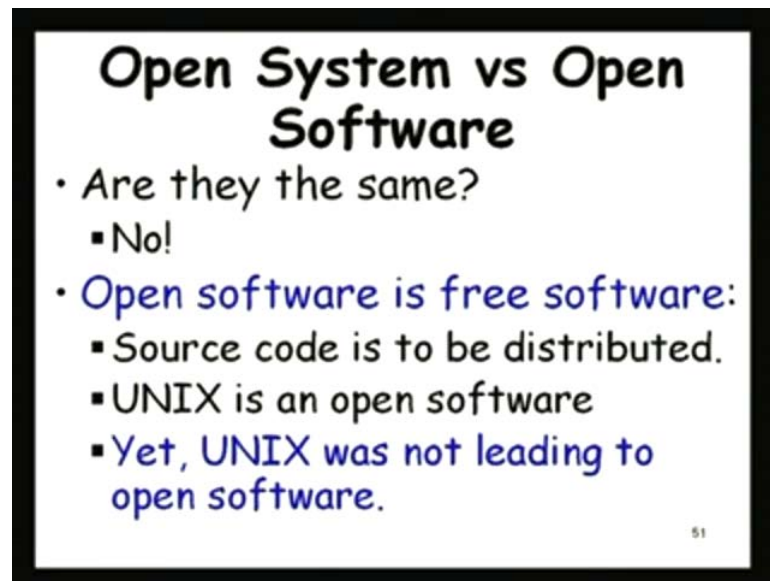
So, different manufacturers different vendors, they develop different cards and on the motherboard, you could attach those cards for example, video processing or example for a specific embedded application, or let say even to have a sound recording, very different types of applications, the vendors could manufacture many kinds of applications, they became available on the PC, because it was a open architecture, everybody knew what are the components, what are the interfaces not copyrighted, anybody can manufacture, and just attach cards, and the board, and it works.

Whereas in Macintosh, they did not, they could not do that, it was not opened you have to use those, and if you really wanted something, you will have to request apple. And as

a result, the number of applications that were available on the PC IBM, PC kept on growing phenomenally and the Mac could not compete.

Soon in four five years, the PC had changed its sip, it had become very powerful many applications available cost had dropped. So, open system is a very powerful concept.

(Refer Slide Time: 43:25)

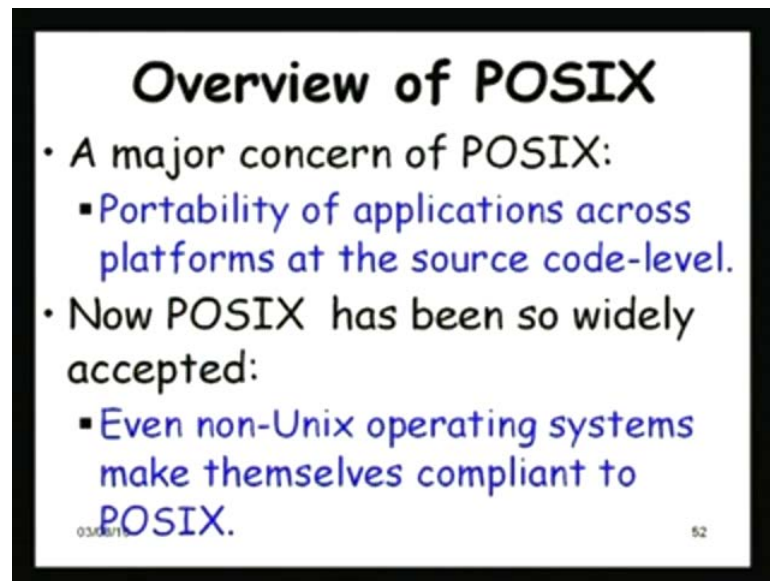


But what is the difference between an open system and open software or they are the same? No, open system and open software are not the same.

Open software is basically a free software, like you are saying open source; open software is open source software, where the source code is distributed UNIX is a open software, but just see that, even though it was a open software, it did not lead to, it was not leading to a open system, I am sorry, even though it was a open software, it was not leading to a open system, yesterday we written here system.

UNIX was not an open system, because different vendors had they extended it in different ways, anyway.

(Refer Slide Time: 44:25)



So, let us proceed with our discussion, the major concern for POSIX was portability of applications across platforms at the source code level, because that time the different versions of UNIX manufactured by different vendors application written by a programmer will not work on another UNIX, considerable rework is necessary.

So, the major concern is that, an application written on one platform should work on another platform, at least, at the source code level, because object code level is difficult to provide portability is not it, yes or no.

Why it difficult, why it objects code level portability is difficult?

(()).

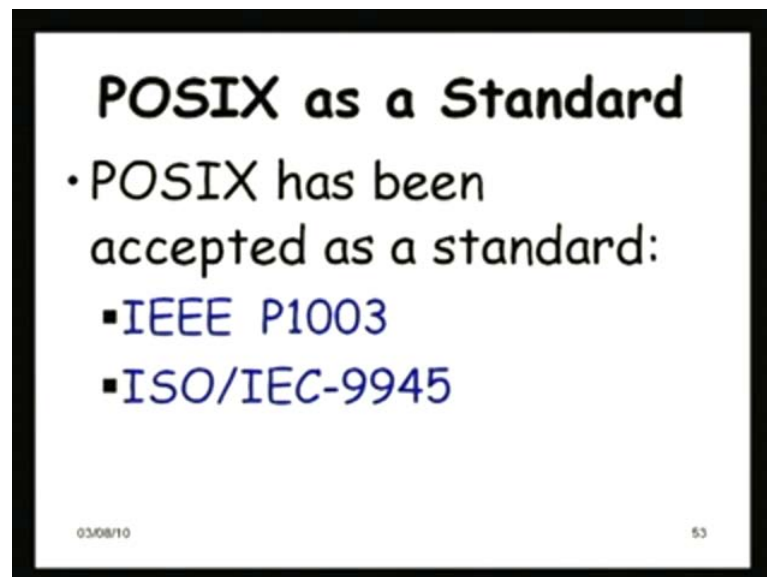
Exactly.

So, at the object code level, it depends on the hardware specific hardware on which the object code is targeted, and if it is meant for an Intel processor, it would not work on some processor, because their instruction set is different.

So, at least for at the source code level, it should be a portable, and it was a huge success, now POSIX is a, so, widely accepted, that even though it was initially developed for UNIX operating systems, different variants of UNIX operating system.

Now, even non UNIX operating systems make themselves compliant to POSIX, because otherwise the users will ignore them. They have their applications running across different platforms. So, they would try to use those platforms and ignore those which do not use, do not are not POSIX compliant.

(Refer Slide Time: 46:28)



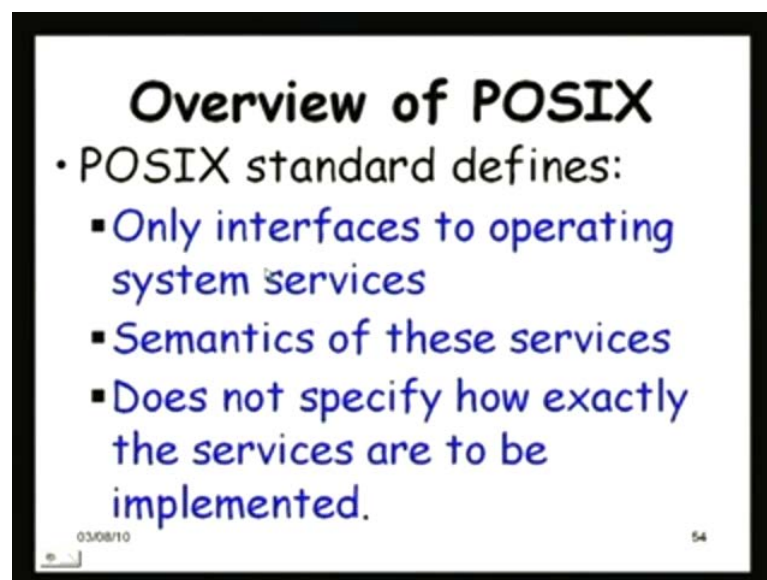
POSIX as a Standard

- POSIX has been accepted as a standard:
 - IEEE P1003
 - ISO/IEC-9945

03/08/10 53

So, POSIX is now an IEEE and ISO standard. So, these are the standard numbers, we are given the triple E P1003 and ISO IEC 9945 standard.

(Refer Slide Time: 46:42)



Overview of POSIX

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

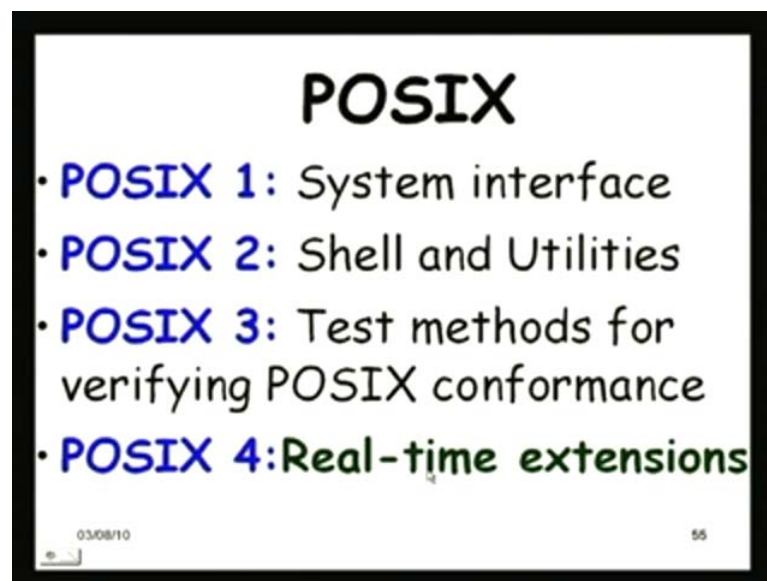
03/08/10 54

Let us look at the overview of POSIX I think I must be clear is that, it is not a operating system by itself it only defines the interfaces of the system services, that is what should be the name of the service, and what should be the arguments for this service, and what is the semantics of this service; these named services have to be supported and they would take this data and provide this result.

It is not an operating system by itself POSIX, but it just is a definition of the interface of the operating system, interface of the operating system services. So, not only the data or the arguments, that is an interface of the services, but also the semantics what exactly they should do and their result they would return.

But it does not specify how exactly a vendor would implement the service, for example, can we use threads to implement service multithreaded services, can it be just one thread nothing is specified, the implementation is up to the vendor, as long as the interface and the semantics are the same.

(Refer Slide Time: 48:08)



The POSIX actually came up with several documents; POSIX point 1 is about the system interfaces which we are discussing that the services and their semantics and so on. POSIX point 2 is about the shells and utilities, because after all, it was a UNIX initiative, then a POSIX point 3, this is another volume a document, which says test methods for verifying POSIX conformance.

If you look at operating system, how do you determine, that it is POSIX conformance what tests you need to do. So, this defines that, and the POSIX point 4 this is the real-time extensions, this is our main concern in our discussion this volume the POSIX 4.

(Refer Slide Time: 49:07)

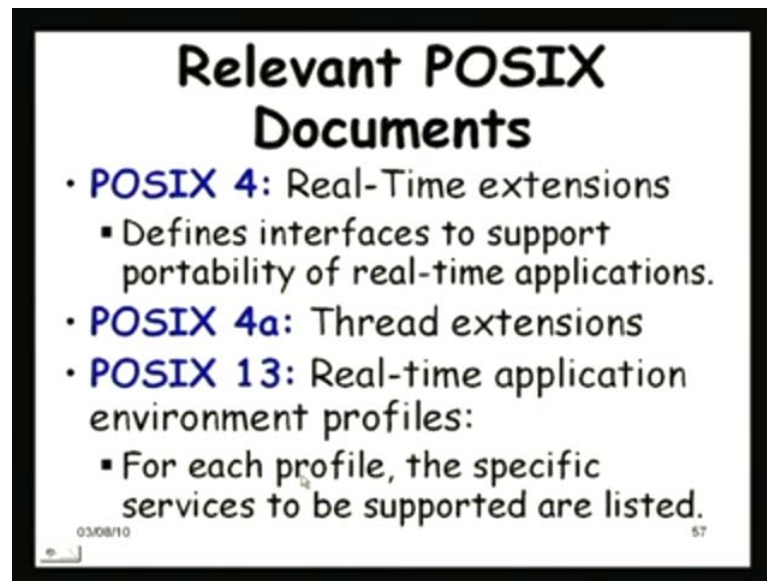
Real-Time POSIX

- 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.

03/08/10 56

But one thing we should keep in mind, as we proceed with our discussion that the embedded systems, they operate under various constraints power, memory etcetera, even the processors speed. So, many of these embedded operating systems, they actually support only a subset of the POSIX functionalities, and considering that the POSIX, the real-time POSIX had become slightly irrelevant to the embedded real-time operating systems, later the real-time working group identified a four application environment profiles, and standards were defined for these different profiles.

(Refer Slide Time: 49:56)



So, let us see the profiles, so that we will look at the next slide. Now, let us look at the POSIX document, that are relevant to real-time applications; one is of course, the POSIX point 4 volume which deals with the real-time extensions to the POSIX operating system interface. The POSIX 4 defines interfaces to support portability of real-time application.

This is the most important document as far as we are concerned. The POSIX 4a document, it deals with thread extensions, because the embedded and the real-time applications are multi threaded, rather than multi process multi thread has a special advantage here for real-time applications. What is the advantage of a multi thread compare to a multi process.

(()):

So, threads have faster response time creation is faster and so on. POSIX document 13 that is real-time application environment profiles. So, this is what we are saying that were embedded application the POSIX 4 is not very relevant, because this operate under constraints and depending on the type of the application, four profiles have been defined. So, these are specific services to be supported under the specific environment profile.

(Refer Slide Time: 51:37)

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.

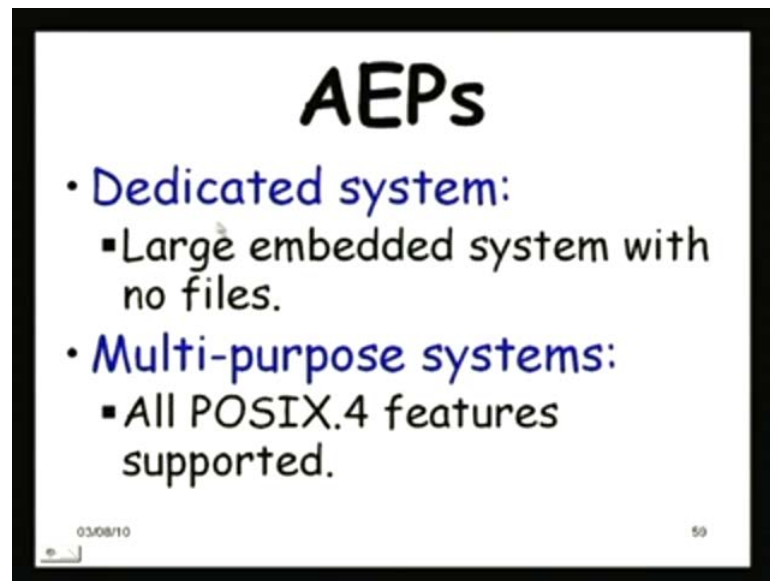
03/08/10 58

Now, let us look at the application environment profiles for embedded real-time application, one is the minimum system, the one is real-time controller; I think two more are there, we will see in the next slide, the minimum system is for the small embedded systems, which are no memory management, no disk, no display terminal keyboard etcetera.

So, these are the simplest embedded application, this is the operating system targeted for this have to be POSIX compliant, have to be compliant with the minimum system environment profile.

Next is the real-time controller profile which is a something similar to the minimum system, excepting that there is a support for file system and input output display, and input, and here the real-time controller has only one process, but there can be a multiple threads, only single process is allowed, but multiple threads.

(Refer Slide Time: 52:57)



Then we have the dedicated systems, these are larger embedded systems with no files, and then, another environment profile is the multipurpose system, which is actually the POSIX 4, the entire POSIX 4 is for the multipurpose profile, but the others are for smaller applications, specific applications.

So, we look at the POSIX 4 extension the real-time services, that have to be proposed, and that will also give us an idea about, what are the standard features required for a real-time operating system, we had seen so far, that what is expected of a real-time operating system, this will actually reinforce, that we will see. What the real-time POSIX standard specifies and that will also give an idea of kind of services that a real-time operating system per specific application need to support.

So, we will just stop here we will continue in the next hour.