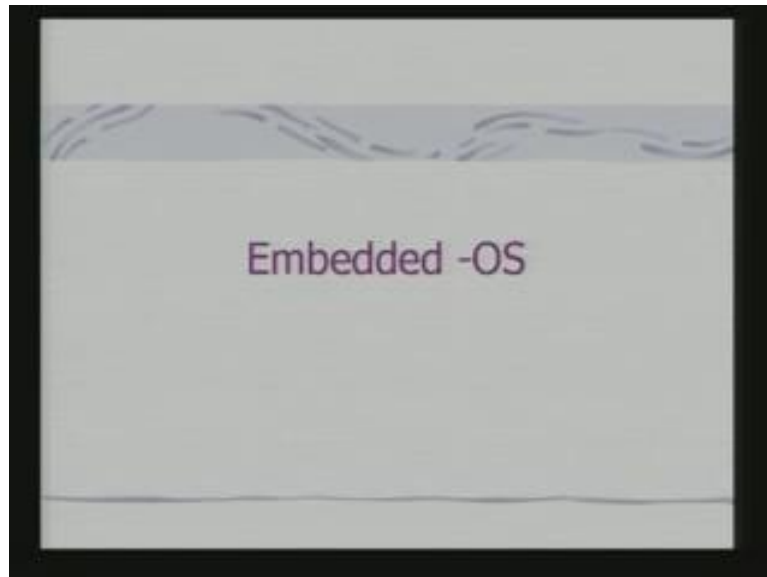


Embedded Systems
Dr. Santanu Chaudhury
Department of Electrical Engineering
Indian Institute of Technology, Delhi

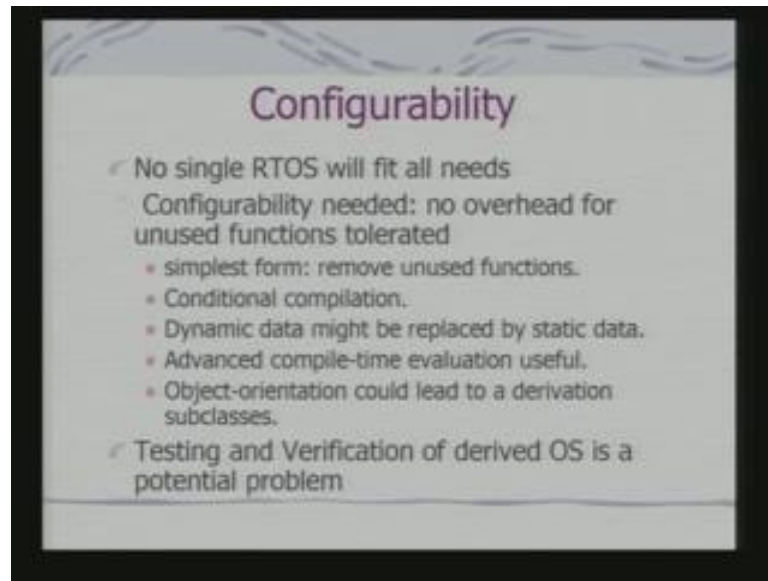
Lecture – 23
Embedded – OS

We have so far discussed the basic features of an operating system which are intended to be used in the context of embedded appliances. Today, we shall recap at these features in the context of the complete systems or complete OS packages which are available for deployment on embedded appliances.

(Refer Slide Time: 01:42)

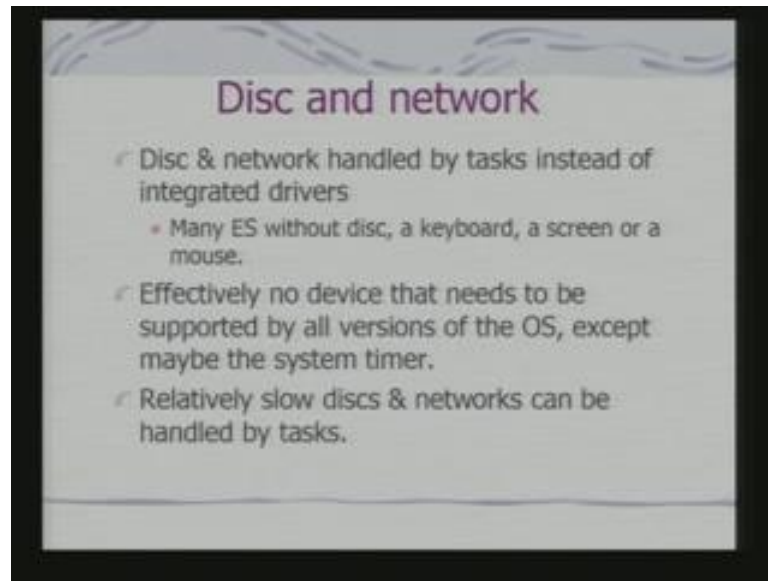


(Refer Slide Time: 01:44)



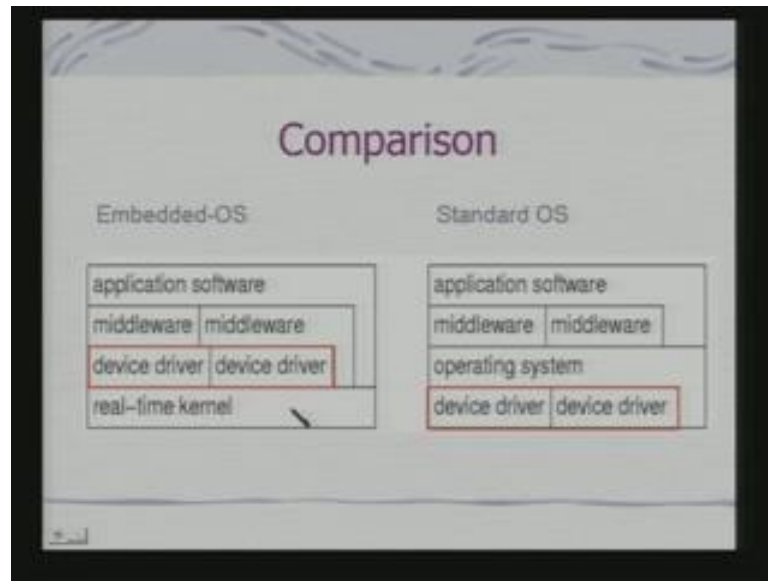
The basic issue is that of configurability. Given an appliance no available package will fit all the needs. And we need configurability, because the overhead for unused functions be tolerated, because you would not like memory to be used for such purposes. So, what is the simplest form remove unused functions and many cases you actually have the feature to compile your OS with applications together. So, we can use conditional compilation to illuminate those modules which are not needed. In a way what we have that such OS are organized in modular approach that means, OS consist of a set of modules and you make the selection of the modules depending on your requirement. And this advance compile time evaluation is useful for optimizing the whole system and object orientation can lead to derivation of sub classes specialization on the basic functions provided by the core OS. But obviously, testing and verification of the derived OS is a potential problem. So, the whole idea is what that you have an OS a generic version of an OS and you tailor it for your suitable applications depending on the functions that you need. As well as you have to specify if required the features of the hardware on which the OS is expected to run. So, configurability is a very important characteristic of such an OS package. The other issue is management disc and network.

(Refer Slide Time: 03:33)



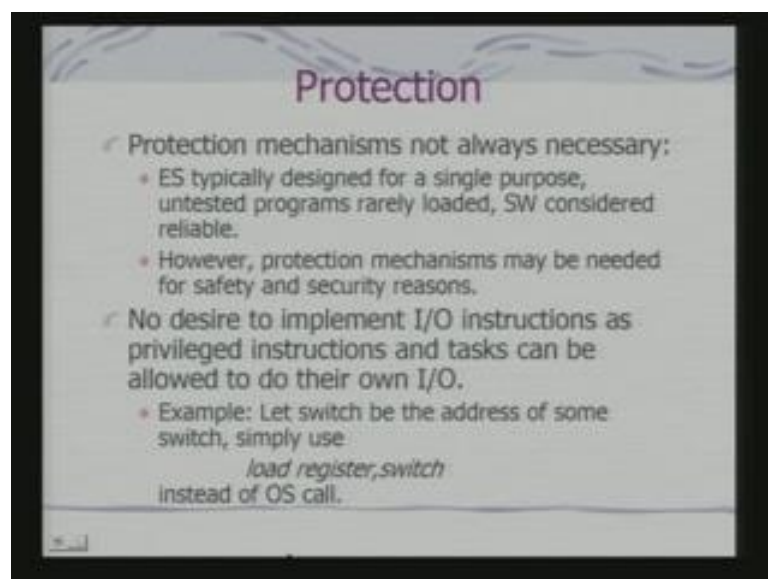
Not all embedded systems have disc and they are all not connected on the network. But general purpose compute in system of variably have OS and network connectivity. So, these 2 aspects are always needed as part of the general purpose operating system. But that is not necessarily true with an embedded system. So, what we say? The effectively no device that needs to be supported by the all versions of the OS except may be the system timer, because critically system timer has to be there to manage various aspects of operation of an embedded system. But other devices are actually depends on what kind of application you are looking at. So, in fact, that is again another reason for configurability and choice of the appropriate device drivers to be made part of the OS. In fact, if you really have discs and networks that can be handled by the tasks this does in form part of the OS kernel and the device driver than comes bundled with the OS.

(Refer Slide Time: 04:43)



So, if we compare the basic idea wise if I have the embedded OS and I have the standard OS see we have the operating system and along with you have got device drivers for a multiple applications. Here, you make choice of the device drivers depending on the devices that you would like to use. And it very likely it would be a real time kernel, because it needs to support real time tasks the kernel has to be a real time kernel which is not a necessity for a standard OS. And in many cases you will be using a standard OS to compile an OS along with applications bundled in to a package to be downloaded on to an embedded appliance.

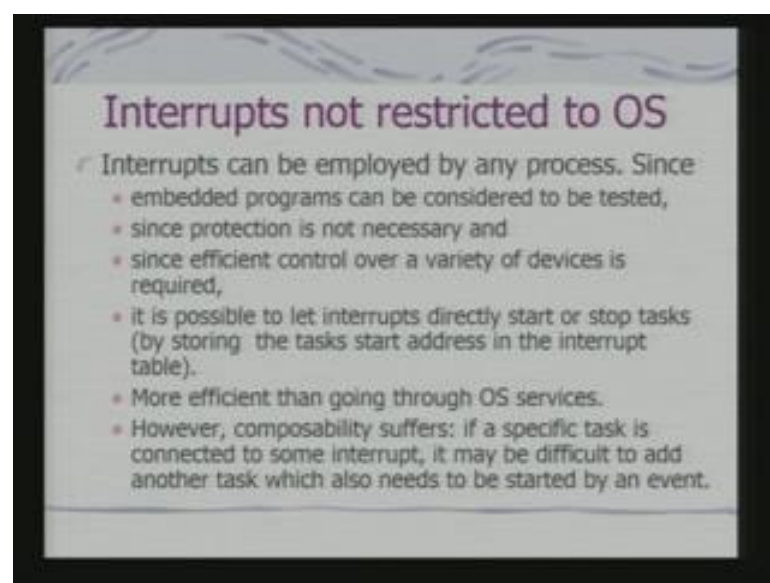
(Refer Slide Time: 05:32)



Next issue is protection. In the last class itself we have seen that you can use MMU to provide a variety of protection. Now, these protections in many cases I require, but also in many cases in an opt provide a limited amount of protection not extensive protection as you require for a general purpose system which is expected to support multiple users. Because yes that is an embedded systems typically design for a single purpose and untested programs really loaded. So, it will not be expected that a program would like to access a memory area of some other process. Software is considered reliable under those conditions hence the protection mechanisms may be needed for safety and security reasons where you want to preserve the option of enhancing the software on the flight.

So, related to this protection becomes the privilege mode operation. In fact, in a standard OS you always have the IO operations in privilege mode that is that when I am actually doing an IO approaches the users may not have a direct access to an IO. But in many possible for a process which is in an embedded OS to access directly IO. So, in IO instructions did not be always be executed in a privilege mode when it is to be done in a privilege mode in we have seen that 1 mechanism is kernel map. So, you map the pages corresponding to the kernel corresponding to the device interfaces in to the page table of the processes itself with the domain memory conditions domain security condition enabled. So, in that case you really do not have a context switch, but there is a switch in the status of the processor for the execution. Interrupts are also not restricted to OS.

(Refer Slide Time: 7:44)

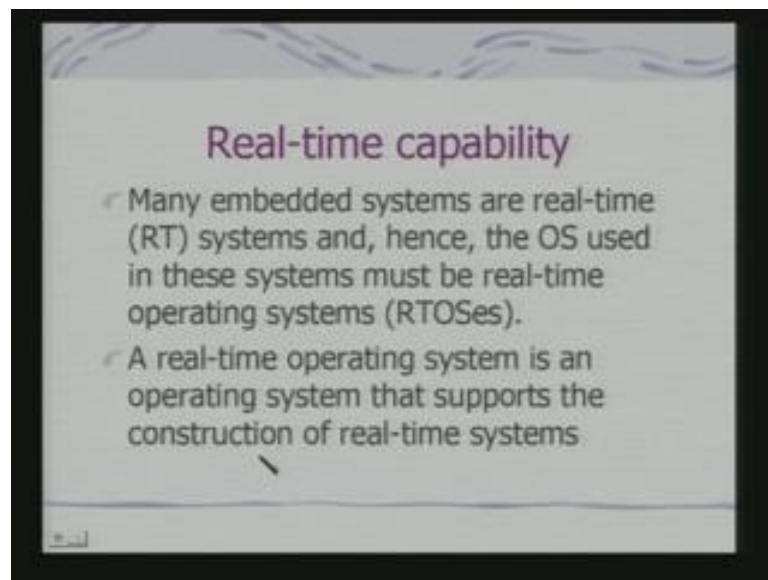


Interrupts not restricted to OS

- ✓ Interrupts can be employed by any process. Since
 - embedded programs can be considered to be tested,
 - since protection is not necessary and
 - since efficient control over a variety of devices is required,
 - it is possible to let interrupts directly start or stop tasks (by storing the tasks start address in the interrupt table).
 - More efficient than going through OS services.
 - However, composability suffers: if a specific task is connected to some interrupt, it may be difficult to add another task which also needs to be started by an event.

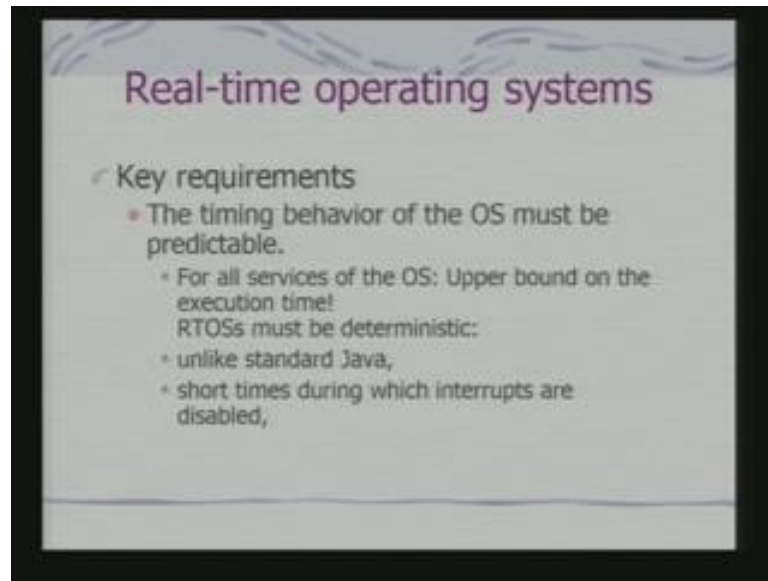
Interrupts can be employed by any process. Since, embedded programs can be considered to be tested since, protection is not necessary and since efficient control over a variety of devices is required. It is possible to let interrupts directly start or stop tasks by storing the task address in the interrupt table. So, it becomes directly math on to the task depending on the application. It can be more efficient than going through the OS services; obviously, the overhead of the OS service execution is getting eliminated and so, what happens effectively the software latency for the interrupt service effectively goes down. However, what you say the composability suffers if a specific task is connected to some interrupt it may be difficult to add another task which also needs to be started by an event In fact, by a similar event. So, it can be done if I use OS service to math an interrupt to multiple tasks depending on the requirement.

(Refer Slide Time: 8:50)



The other point which is fundamental concern for an embedded appliance and a system is a real time capability. So, many embedded systems are real time systems and hence the OS used in these systems must be real time operating system. And what is the real time operating system in a way? A real time operating system is operating system that supports the construction of real time systems. And we have already had the definitions of real time systems and we know the difference between soft real time and hard real time systems.

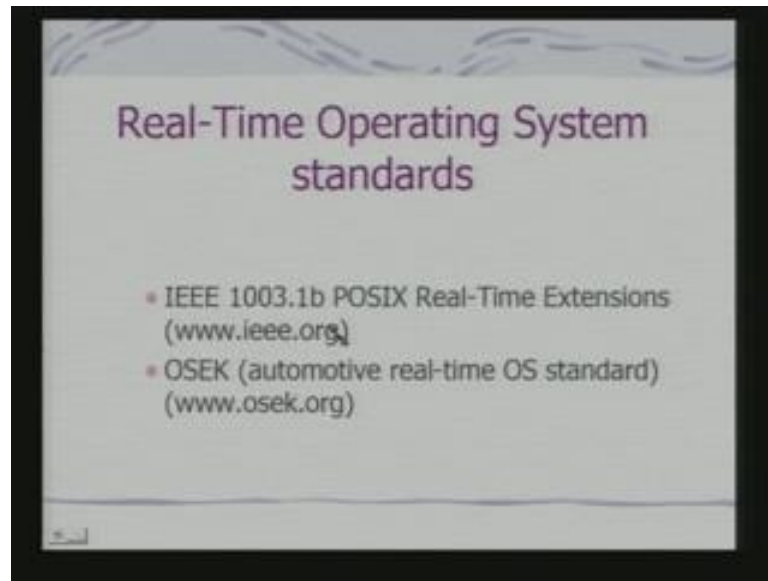
(Refer Slide Time: 9:25)



So, what are the key requirements? The basic key requirement is timing behavior of the OS must be predictable we have talked about the scheduling strategy. But we have to also consider the services that OS provides. Each service has got an overhead in fact, scheduling services has also, but an overhead and you have considered whole scheduling scheme without considering the overhead. But in actual real world deployment this overheads play a crucial role. And what we need to have? You need to have exact timing and predictable timing of the OS services. So, for all services of the OS we should know the upper bound on the execution time So, in that sense RTOSs must be deterministic then only I can actually talk about carsickness of the schedule why unlike standard java?

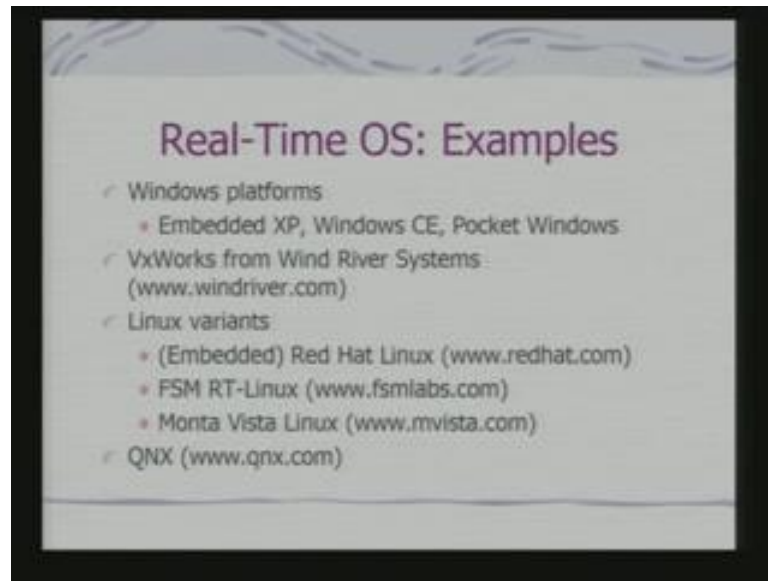
In a standard java we have already talked about that garbage collector can come in to play at an arbitrary instance of time holding up a thread which is doing a job or the task the user intense to do. So, that brings in non determinism and that is not really desirable. So, such kind of interruption definitely is not desirable in a real time OS. Other thing is there should be short times during which interrupts are disabled, because interrupts is disable for a long time then you cannot actually guarantee meeting of the service constrains of the devices are external senses. Because external senses when they are providing an interrupt it is required the devices at to be serviced with in a deadline. If you actually disable an interrupt for a long time period then interrupt will not be acknowledge and such services cannot be provided. In fact, there has been standards for this kind of real time operating systems.

(Refer Slide Time: 11:41)



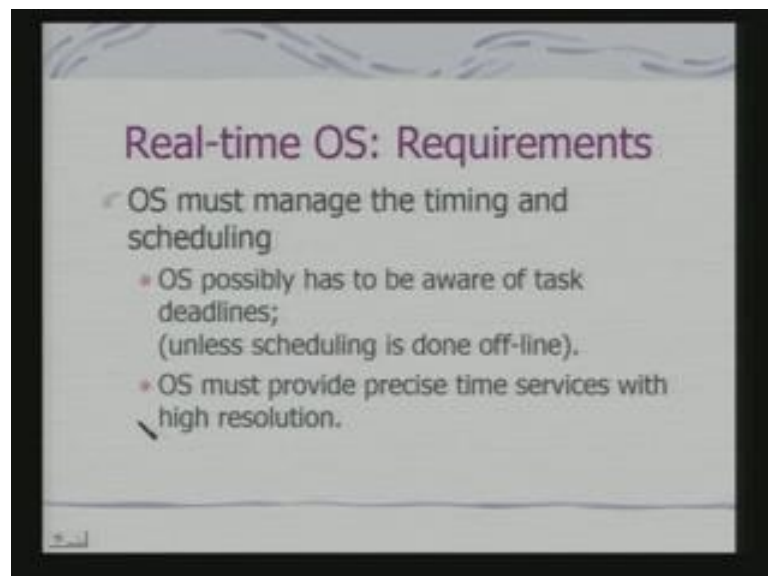
POSIX is an IEEE standard and in fact, if I have an OS my own OS I would like it to be POSIX compactable so, that my applications which are POSIX complaint can run on my OS. And in fact, that is the basic motivation for defining this kind of standards because actual implementation can vary from system to system OS to OS. But the basic features if they can be standardized then your application can run on a variety of operating systems. The other thing is OSEK this is automotive real time OS standard. This is particularly meant for automotive applications. So, there have been standards and you talk about OSs and when you are making a choice of the OS you would like it to be POSIX complaint. And when you develop an application you will be using the system calls which should be ideally POSIX complaint. So, that they can move from one OS to another OS; that means portability across the OS without much of rework required.

(Refer Slide Time: 12:54)



There are number of such examples which are available and today being used windows platform this are primarily source from Microsoft targeted for handle devices then VxWorks from Wind River system possible. The most well known commercial operating system used in embedded appliances. There are variety of Linux variants therefore, Linux is now is turning out to be the most popular in fact, on an upward swim for its application in embedded domain.

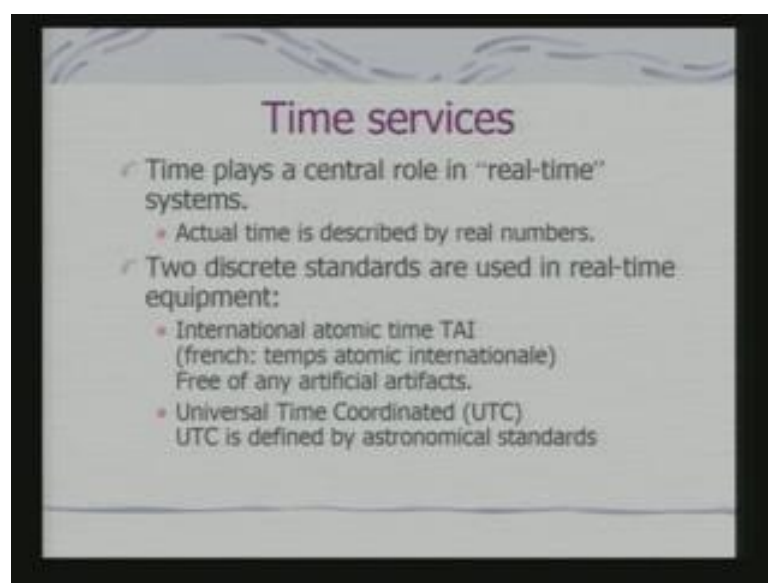
(Refer Slide Time: 13:30)



Now, what are the further requirements? The OS must manage the timing and scheduling what we have found first of all the OS services had the deadline then OS all the OSs should provide the services for timing and scheduling. And OS has to aware of task deadlines unless scheduling is done off line; that means, if you are using a table revert scheduling a design time scheduling then the OS need not bother about that deadlines. But otherwise it has to bother about deadlines to decide about the priority of the tasks. And OS must provide precise time services with high resolution in fact, this is the key issue, because depending on the time resolution that OS can handle.

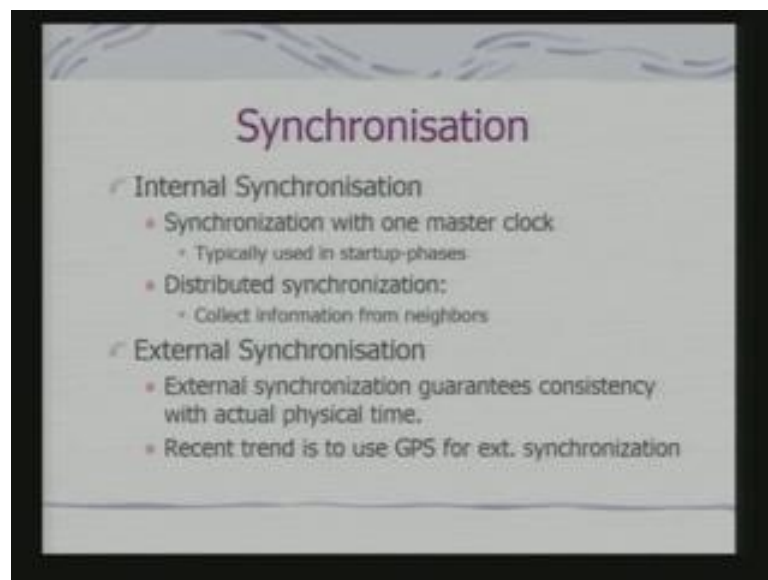
You can have specificity of the deadlines try to understand this because you have a timer and the system may be operating at Giga Hertz timer has a specific counts a you have OS has got the internal structure to maintain the timing information. In fact, you can understand the pending on size of the data structure you can actually talking terms of the time resolution with which you can deal with the deadlines. So, for example, I may have a gigahertz clock, but I have implemented an OS which uses same a simple four bit to store the timing information internally. In that case I cannot use time resolution of gigahertz level for dealing with the deadlines. So, time resolution in terms of your internal management becomes a key issue for managing the schedules. And this is related to your underline hardware as well. So, what we say the time places a central role in real time application.

(Refer Slide Time: 15:27)



And this is also related to your actual time. So, actual time is normally described by real numbers. And if I have if you see depending on what is the data structure allowed that are how many bytes being allowed for a storing this real numbers would actually determine the timing resolution. And there are two discrete standards are used in real time equipment. Now, these are with respect to the physical time making the references with respect to the physical time. One is international atomic time TAI and universal time coordinator that is UTC. Now, this is for an unversialization of the physical time when you are using in an embedded system which may be network and working with other system located at different time OS. Just consider that when you making a mobile a phone call the time zone if there is a tariff depending on the time period the time zone can actually vary from country to country. So, there has to be timing information associated with the call. So, that way also the physical times becomes an important parameter. So, how the synchronization does is done with respect to the physical time? In fact, many of the embedded systems and the OS do have features to synchronize with physical time.

(Refer Slide Time: 17:04)

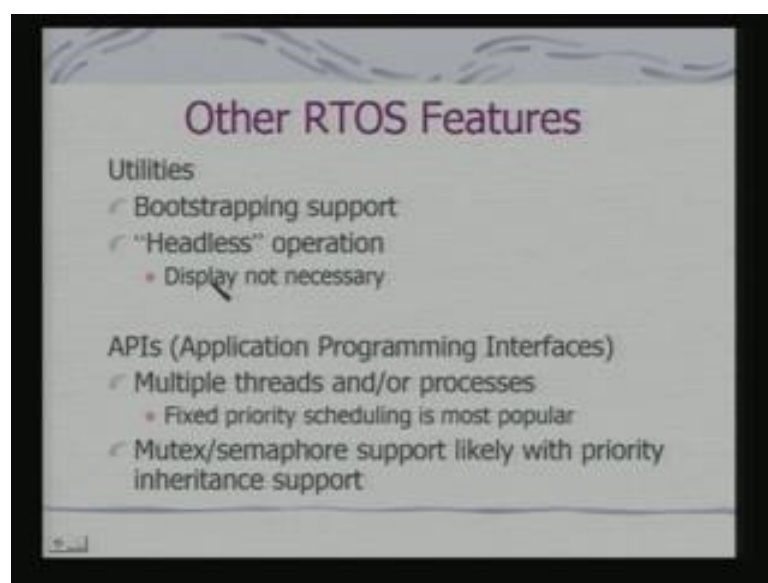


One is internal synchronization in fact; this may or may not correspond to synchronization of the physical time. But this would definitely correspond to synchronization with respect to a time stamp which is consistent across systems which are dealing with the time stamped data fine. So, you can understand today, I am time stamping may be right now, time stamping on my mobile some time say a 10 30. I need

riches a system which is network and it reaches at time on that mobile may be written 30 what is the meaning of this 2 times? It has reached may be after say 10 seconds how this time becomes pair. If they are clocks and not internal synchronize in fact, this synchronization then very important problem when you consider a distributed OS and a distributed coordination in a distributed OS. So, similar issues do come in the context of embedded systems as while. So, there can be synchronization with respect to one master clock and this is typically used in startup phases.

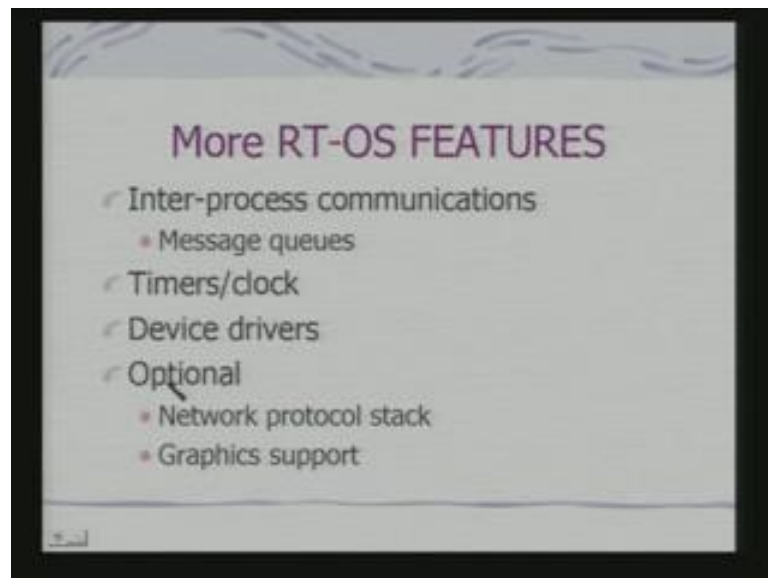
And there can be a distributed synchronization by collecting information from other neighbors and trying to offset the errors with respect to the timing information available from the neighbors. So, it something like that I look at your watch and as well as you friends watch and try to find out the best time. And all of us all few of us can synchronize our watch with respect to the joint agreement of a time. The other issue is that of external synchronization. External synchronization guarantees consistency with actual physical time in fact, across multiple time zones as well and what is done? In now a days to use GPS for external synchronization global positioning system do communicate with the timing information also and it can be of the order of 100 nano seconds the time resolution. So, using that the physical time synchronization system. In fact, physical synchronization of time is a very important issue with these kind of embedded communication appliances which needs to manage a variety of tariff structure.

(Refer Slide Time: 19:40)



Other RTOS features is bootstrapping support, headless operation this they really not have and a display to show the status. In many cases you have some kind of if you want to flag an error. Some kind of external hardware interfaces to be provided not really a display and you have API's. Now, when you develop an application with reference to an OS you will be using this API's. So, there will be multiple threads or processes in fact, API provides you with the facility to create threads as well as processes. And they will provide support for semaphores with priority inheritance. And we have discussed and we have seen why priority inheritance is so, very important when you where scheduling real time tasks. So, typically they come with priority inheritance and the semaphore support and ability to create multiple threads with fix priority scheduling.

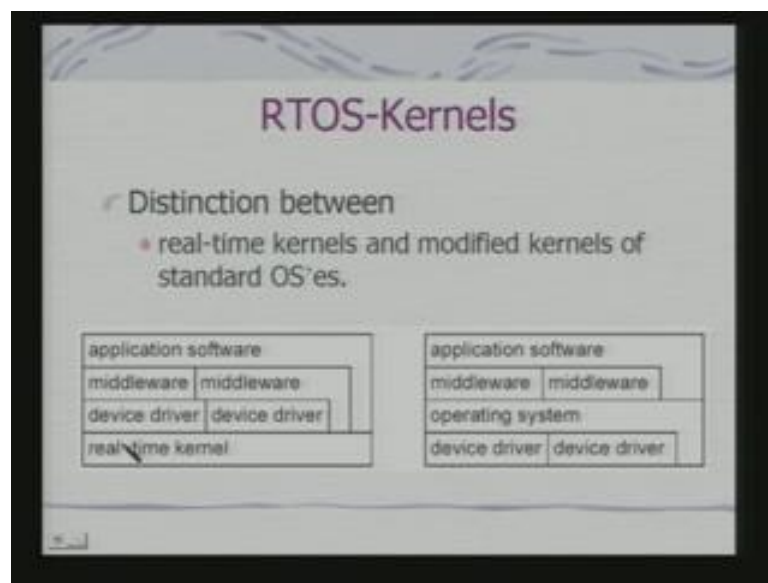
(Refer Slide Time: 20:45)



Also there are features for inter process communication. The most common thing is message queues. If there are two processes next to communicate how do they do it? So, you can think in terms of a message queue a standard a producer consumer kind of a scenario. So, the producer puts in message from 1 end to the message queue and the consumer removes the message from the other end of the message queue. It is a kind of a buffer it can consider a simplest implantation of these in terms of a buffer and so, it is a bounded buffer. So, in the buffer is empty; obviously, consumer it stopped from accessing it and when the buffer is full producer is stopped from writing on to it.

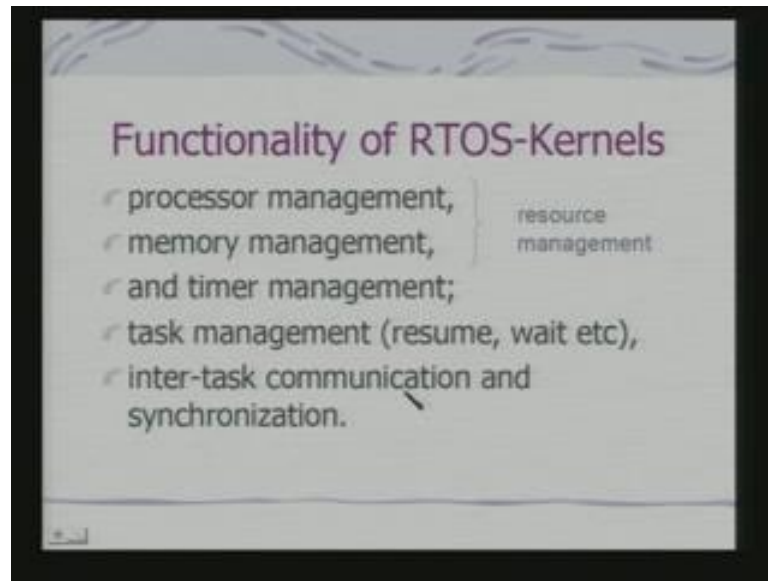
But there could be various other ways also to implement message queues. In fact, it can be a complete asynchronous implementation where the messages are posted message are received the process continues its own execution. And may come back at an appropriate point in time to check the message queue and take appropriate action. Timers and clock management In fact, this is a very key features it has to be definitely there. Device drivers there may be a set of device drivers available and your network protocols stack and graphic support are optional depending on what kind of service you are really trying to provide.

(Refer Slide Time: 22:16)



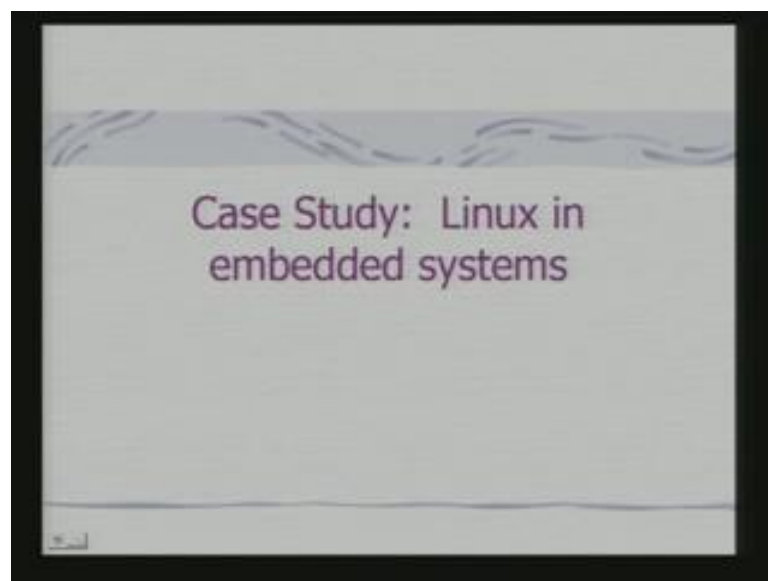
So, we look at RTOS kernel see all this features has be provided through real time OS kernel. And we are seeing the same diagram In fact, real time kernel can be this pure newly return real time kernels are it can be a modified kernels of standard OS. So, if you look in to it this is going back to the standard OS diagram. The only difference here is the kernel area would have could be modified with a different scheduling scheme and scheduling strategy and here you have a real time kernel.

(Refer Slide Time: 22:58)



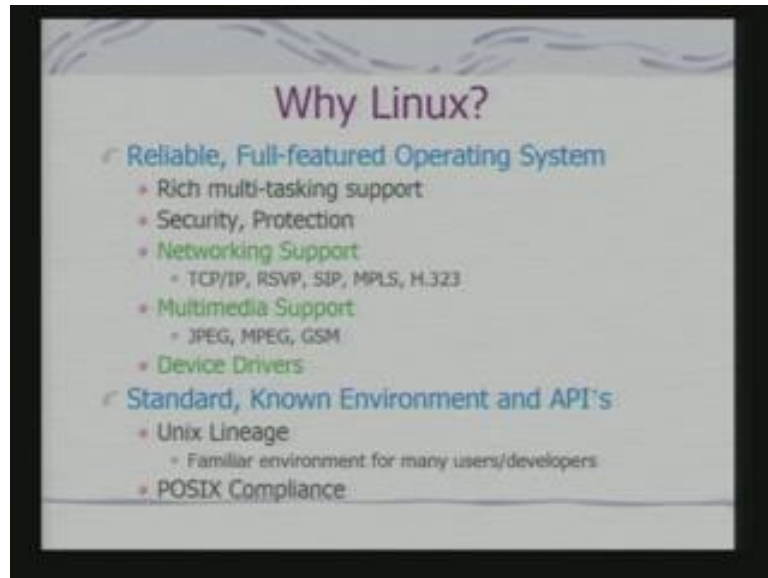
So, what is the functionality of the real time kernel? Obviously, the processor management this key memory management and timer management among the devices timer is a most important device and it has to be universally managed. Other devices are actually picked up depending on when there are available or not. The other thing is task management that is resume way it start extra and inter task communication and synchronization for sheared resources. So, what we shall see is that with reference to Linux how does this address kernel features are provided.

(Refer Slide Time: 23:39)



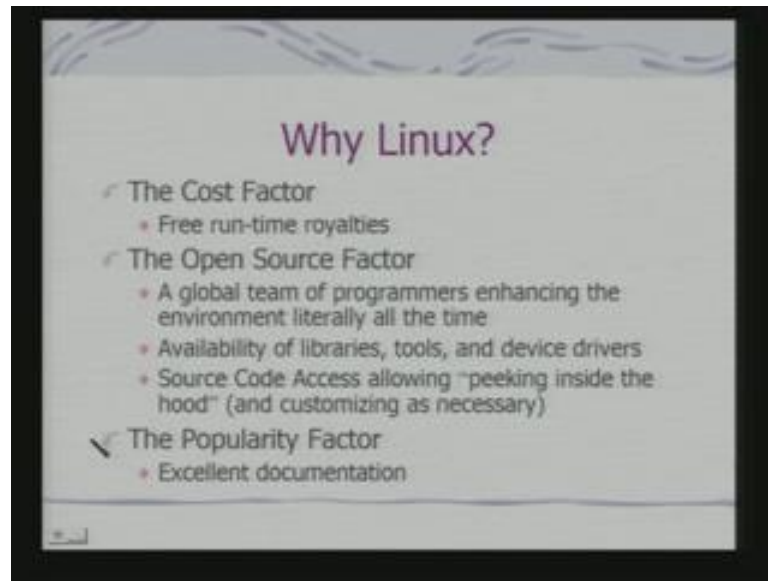
There are various strategies which have been followed to enhance Linux kernels to provide this feature. In fact, that gives you what I already said that two basic approaches writing a real time kernel are modifying a existing kernel for enabling this services.

(Refer Slide Time: 24:00)



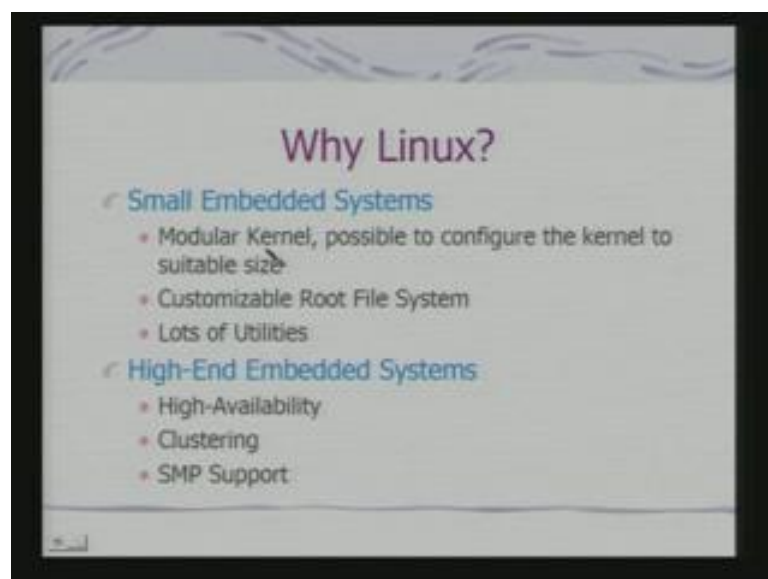
So, why Linux? The basic idea is why Linux is for off-screen for the use in embedded systems. It has got a rich multi tasking support security and protection it was also got a networking support although I say networking may not be a key issue. But many ways networking today many of the embedded systems are coming with networking features, multimedia support and variety of device drivers which are available on Linux and standard known environment an APIs. So, the familiar environment for many users and developers as well as POSIX compliances so, all this features make Linux a popular choice.

(Refer Slide Time: 24:37)



In fact, the other features as well the reasons as well. So, it is a cost factor, the open source factors we have got the source code which is allowing what you say the peeking inside the hood. And customizing as necessary which is not true if you are really working with a commercial OS and popularity factor based excellent documentation and excellent third party documentation. Although it is not always true for each and every aspect of learners, but the whole idea is that if there is a problem you get variety of support that is a basic commercial consideration for making a choice about Linux. So, the other so, what happens is with the Linux if you have a modular kernel.

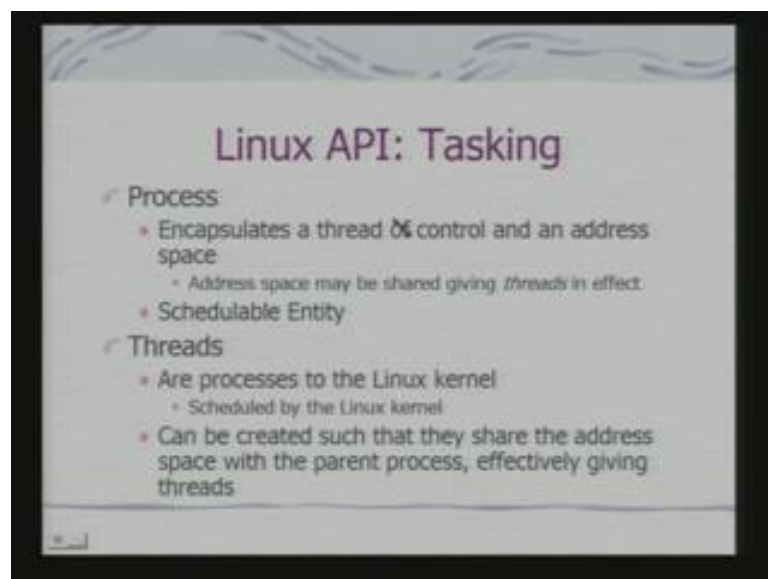
(Refer Slide Time: 25:21)



You are possible to configure the kernel to suitable size. So, Linux is organizing embedded Linux is what we say that if a Linux is organize in terms of a collection of modular components in the kernel you have got a customizable root file system and lots of utilities. If you look at the high an embedded system point of view it is a high availability is in the sense what is availability of the software system? Availability means that if I distinguish between reliability and availability reliability is what? The time for which it would work without failure that the probability is that for certain time period the OS would work without failure.

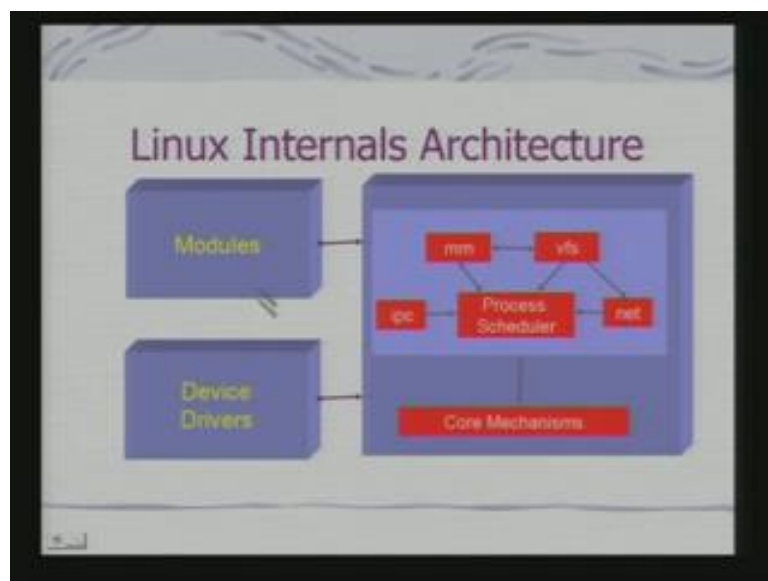
Availability is what if there is a failure there is an ability to catch that exception and come back soon enough so, that the services can be rest out. So, that makes an OS more available. So, the problem is that we talk about availability of the software system we accept the there may be failures, there may be exceptions and modular such failure OS has the ability to come back and rest of the service. So, this is how availability distinguished from that of reliability. Also there is clustering ability to cluster multiple system and symmetric multi processing support. So, if you have a multiple processors you can use them through this OS. This is a primarily high end application if they are really looking at a more complex computation has to be done then only this issues become important.

(Refer Slide Time: 27:08)



So, if you got the Linux API and the most important aspect of the API is the tasking and here what happens? A process the Linux API you work in terms of the processes. A processes encapsulates a thread of control and an address space. Address space may be shared giving threads in effect that is there are multiple threads within the processes boundary and processes schedulable entity. In fact, threads in a way or processes to the Linux kernel and they are scheduled by the Linux kernel. And can be created such that they share the address space with the parent process effectively giving threads. So, what we are talking about is two ways of leading to threads. So, I can talk about processes the schedulable entity is and process can be consisting of a set of threads which share the address space. I can look at threads we should Raquel and to processes for Linux kernel. So, threads now, become a schedulable entities and a set of threads group together as a process so, that the shear a common address space. So, this can be done for your application using standard Linux API.

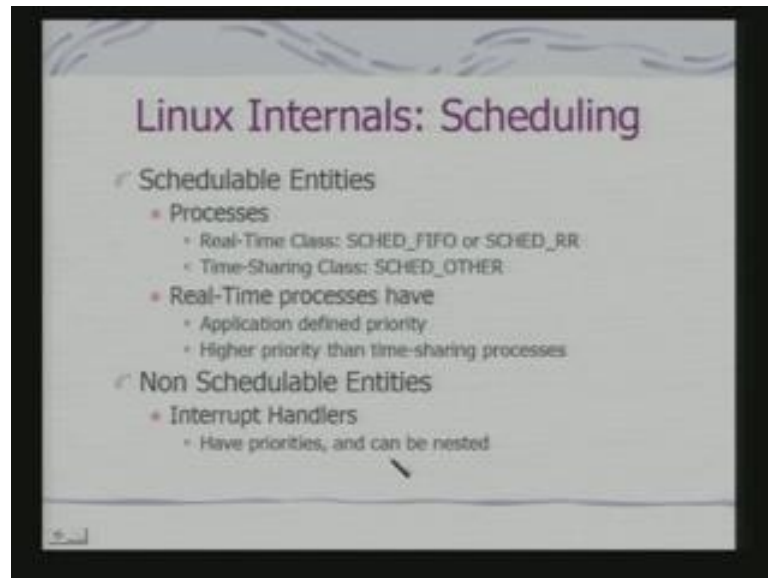
(Refer Slide Time: 28:28)



So, if you look at typically a Linux internal architecture is here we are looking at a key issue as process scheduler. You can have multimedia support you can have a inter process communication networking support and this is link to the core mechanisms. Now, what you do? You would select the modules which are available in terms of the modular architecture device drivers which are available together and construct this architecture. Because process scheduler would manage all this processes and, because they are of the different nature and that, why they are link to process scheduler.

Depending on what kind of tasks do you have you will have the corresponding module configured for your particular version.

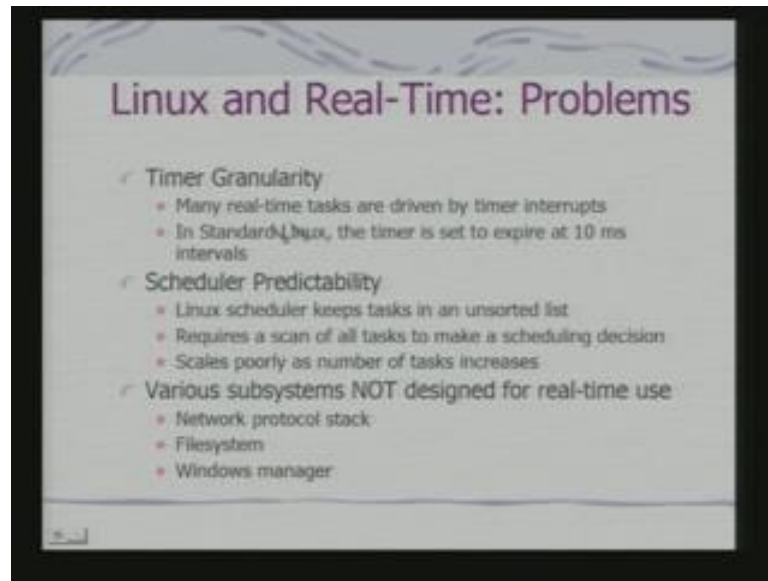
(Refer Slide Time: 29:20)



So, what are schedulable entities? Schedulable entities I already talked about processes are threads are also processes. If you go wire that path and so, you can have what is called two distinct classes' real time class or time sharing class when you talk about a time sharing class obviously, what we are implying is that for this task there are no deadlines to be meat. Application processor can have application defined priority as well as higher priority than time sharing processes. And what are non schedulable entities? It would have interrupts are interrupt handlers can have priority and can be nested. But they are not schedulable why they are not schedulable?

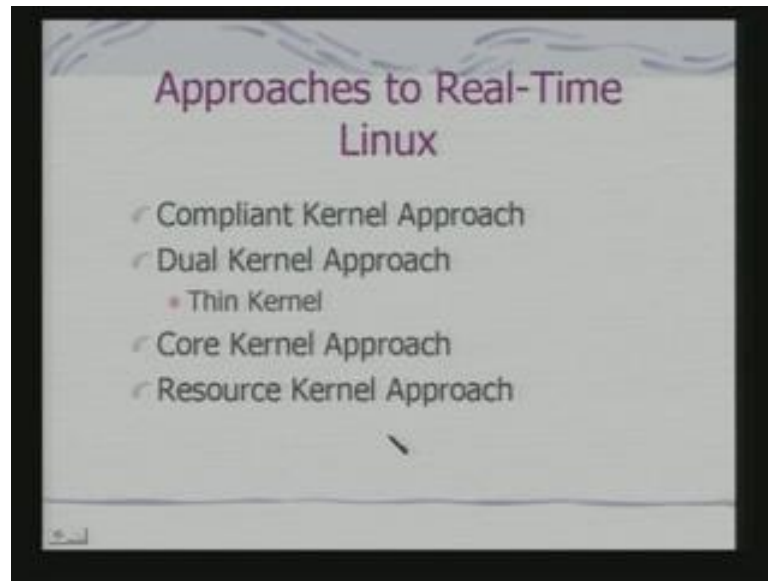
In the sense when such interrupt arrives the interrupt has to be serviced and priority between the interrupts means either how they are to be resolve when simultaneously more than interrupt is spending are when a particular interrupt is serviced whether you would accommodate interrupt another interrupt where that is being serviced . So, that organization can be done through the OS as well, because OS has to disable interrupts. If there are mask able interrupts OS will disable those interrupts to while a particular interrupt is being serviced other interrupt will not be deducting. So, that is again a configurable aspects, but there are some problems; obviously, which is there what is the timer granularity?

(Refer Slide Time: 30:57)



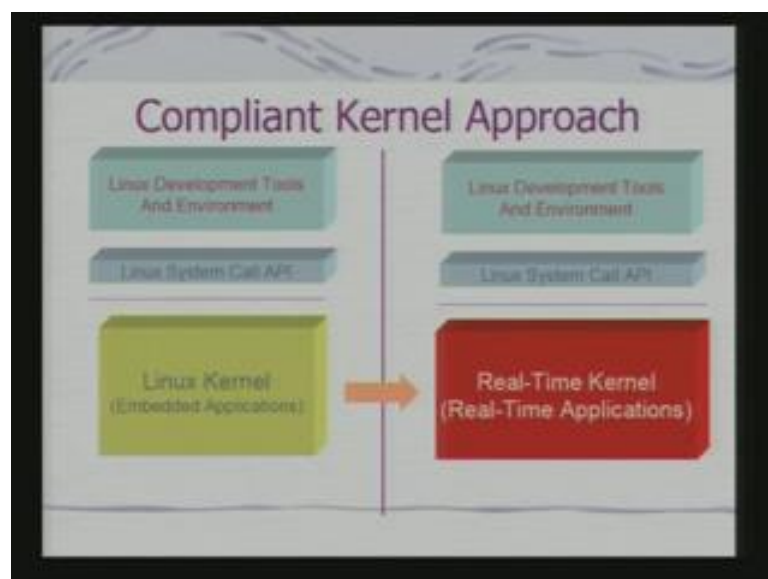
In standard Linux the timer is set to expire at 10 millisecond intervals, because this is the smallest interval that you can really have depending on the way the ways has been configure. So, the granularity can become problematic; that means, you cannot have the deadlines specified at a granularity lower than 10 millisecond level. Scheduler predictability scheduler keeps tasks in an unsorted list. So, requires a scan of all tasks to make a scheduling decision. So, that has an overhead and so, it will scale purely as a number of tasks synchronize. If you are have a more real time tasks can; obviously, have a real problem and various subsystems on design for a real timings. Obviously, network protocol stack file system windows manager required. You can say that they are needed also in various embedded subsystems when they are not needed the really do not call as a problem. But when they are needed they do cause a problem.

(Refer Slide Time: 31:55)



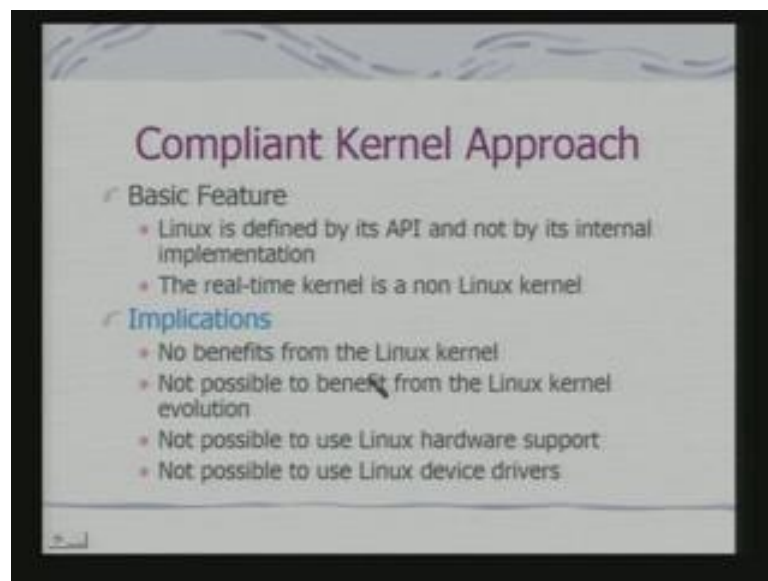
So, how do you modify are the basic issue is that how do you deal with the Linux kernel and bring in this kind of real time features. Currently on the, if you look at the different flavors is a Linux that is available the approaches that have been followed at the following; compliant kernel approach, dual kernel approach, core kernel approach and resource kernel approach. So, these in fact, in a way are examples that how you architect an operating system. We have looked at features of an operating system we have not really looked at how you architect an operating system. So, what we are looking at with reference to Linux how you architects such an operating system.

(Refer Slide Time: 32:44)



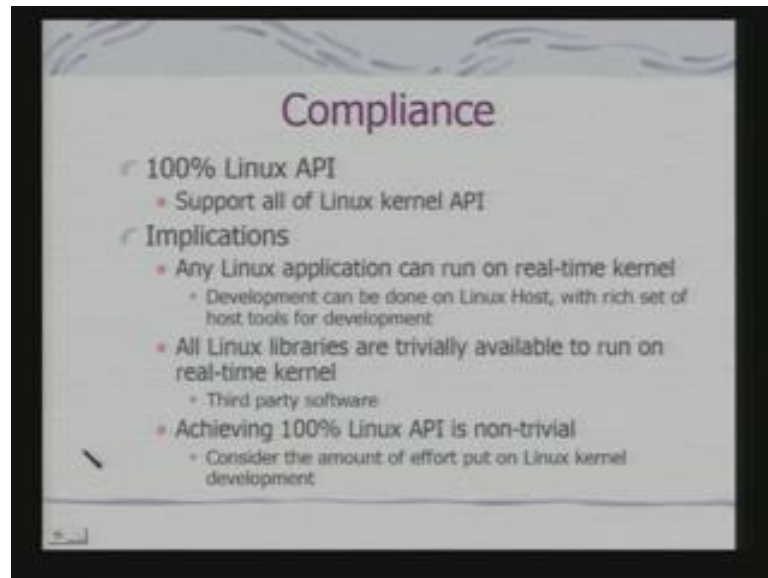
So, a compliant kernel approach is based on this basic philosophy. You got a Linux kernel which all of us know any of got this Linux development tools and environment which uses standard Linux system call API to develop an application, but you do not have a real time feature on this Linux kernel. So, what you do? You write a kernel which is a real time kernel, but it provides a same API as that of the original Linux API. So, that gives your compliant kernel approach. So, what you say?

(Refer Slide Time: 33:25)



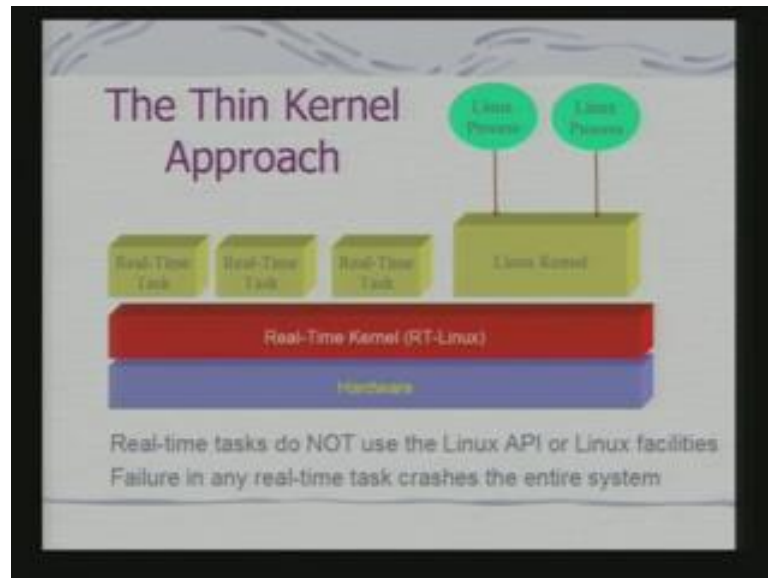
The basic philosophy the Linux is defined by its API and not by its internal implementation. For an application developer Linux is defined by an API and note by its internal implementation. So, the real time kernel is strictly unknown Linux kernel you can provide your own kernel as such with an API same as that of the Linux kernel. So, in this case what are the implications? You really have no benefits from the Linux kernel you cannot you not possible to benefit from Linux kernel evaluation Linux kernel changes over time, because that is again what are the issues software it is evolving in a continuous basis? Not possible to use Linux hardware support not possible to use Linux device drivers. So, these are things which have to be developing as part of your compliant kernel approach.

(Refer Slide Time: 34:20)



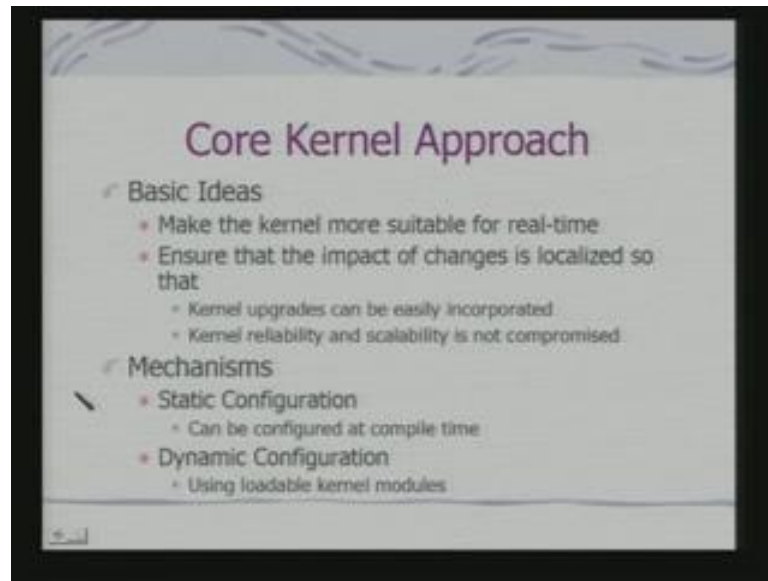
So, compliance is if you have a 100 percent Linux API they needed to support all of Linux kernel API and all of Linux kernel applications which are built up on the Linux kernel system calls. So, any Linux application therefore, can run on real time kernel development can be done on Linux host which rich set of host tools for development. This is the basic advantage all Linux libraries are trivially available to run on real time kernel. So, all third parties comes in, but achieving 100 percent Linux API is not trivial. So, consider the amount of effort put on Linux kernel development which has taken place in a generic fashion So, this what we have look trying to look at as a trade of complaint kernel approach what could be the alternative.

(Refer Slide Time: 35:11)



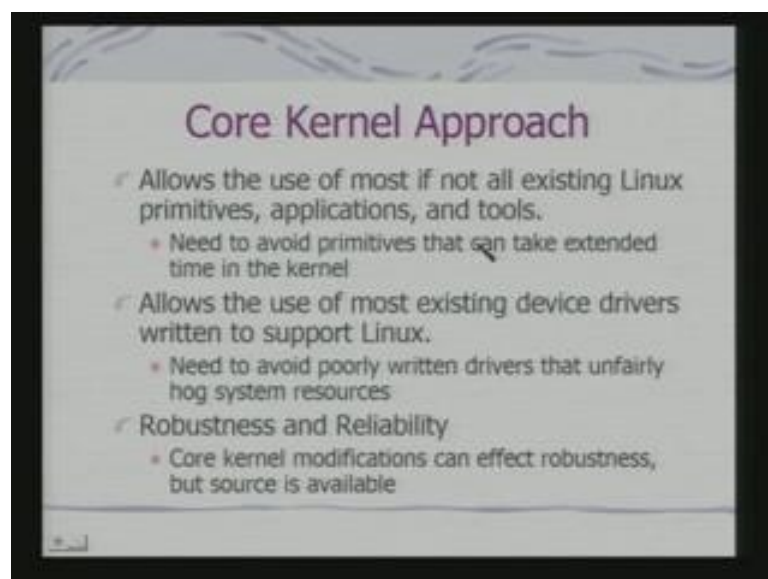
Alternative is a dual kernel approach and here what we say one of the dual kernel approach is the thin kernel approach. So, we have got at this layer as a real time kernel at Linux and your Linux kernel classical Linux kernel becomes a task under real time kernel. So, your user processes this Linux processors they can be run on the Linux kernel and you real time tasks gets scheduled by the real time kernel. And In fact, in a way this you can consider a set of. So, called periodic tasks if you go back to a original model which does not have a real time. So, that could schedule by the Linux kernel. So, in fact, real time kernel schedules Linux kernel and Linux kernel in terms schedules you use a processes which can not necessarily real time processes. So, real time tasks do not use the Linux API or Linux facilities failure in any real time tasks may crashes the system, because it is going through this path. So, the whole development is the real time task may not be based on Linux API.

(Refer Slide Time: 36:32)



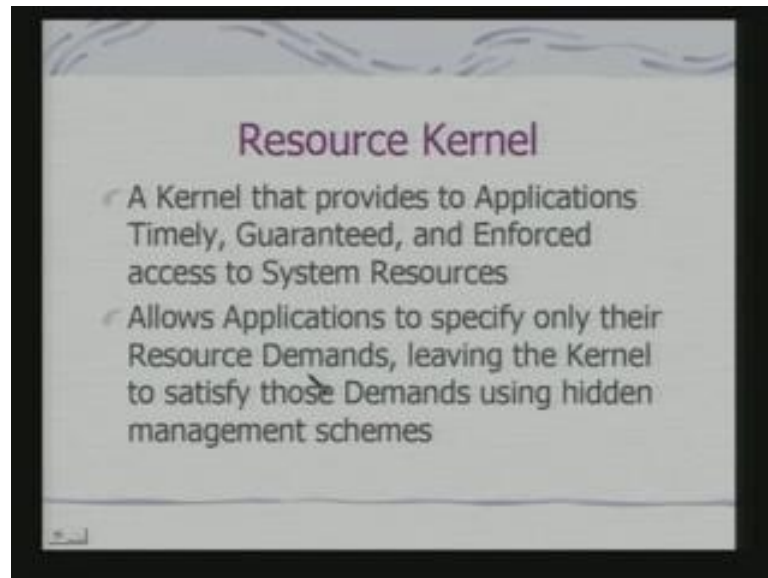
The other approach is core kernel approach where you make the kernel more suitable for real time ensure that the impact of changes is localized. So, that kernel upgrades can be easily incorporated and kernel reliability and scalability is not compromised. So, there are two ways to do it one is the static configuration can be configured at compile time, other is dynamic configuration using loadable kernel modules that is in the run time you make some of the modules enabled. Now, in this case; obviously, what you assume is that you have a secondary storage where you have the storage information by which from which you can actually add on the modules on demand.

(Refer Slide Time: 37:21)



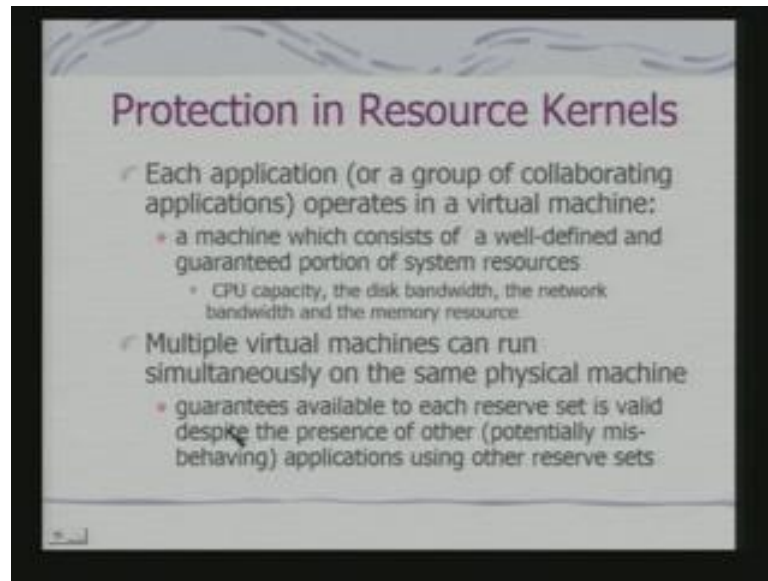
So, the core kernel approach allows the use of most of it. If not all existing Linux primitive applications and tools and allows use of most existing device drivers written to support Linux. But if you are trying to play with the core kernel then it can effect robustness, but in a way source is available this still a possible way to architect the Linux system for a real time application.

(Refer Slide Time: 37:47)



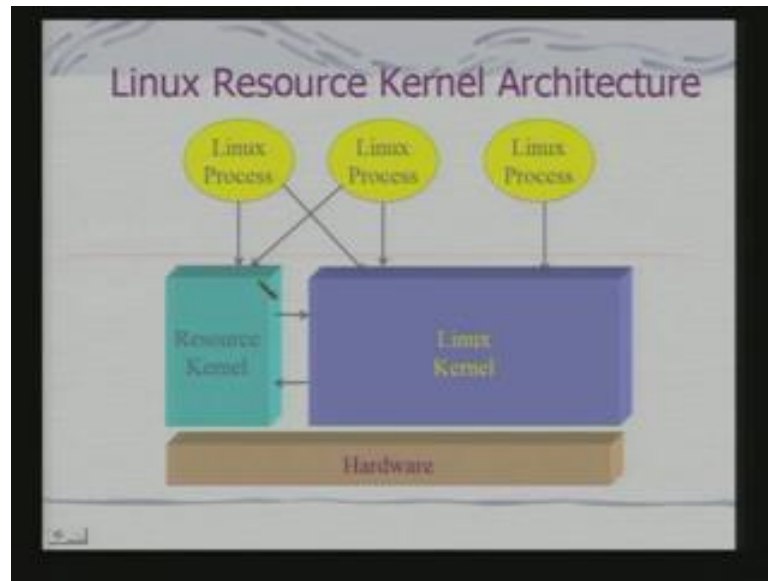
The other approach is that of the resource kernel. Here conceptually it is the distinct approach from that of what you had considered so far. But philosophically it was closer to that of the dual kernel approach. So, what you a say a resource kernel is a kernel that provides to applications timely guaranteed and enforced access to system resource allows applications to specify only their resource demand, leaving the kernel to satisfy those demands using hidden management schemes. In fact, this is architecturally much advanced version in a flexible version compare to others.

(Refer Slide Time: 38:28)



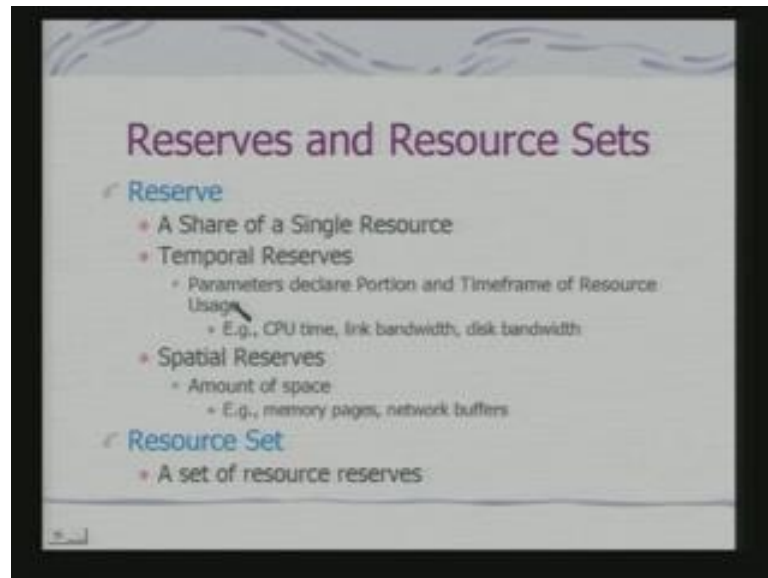
So, what is the production in resource kernels? Each application are a group of collaborating application operates in a conceptual virtual machine. A machine which consists of a well defined and guaranteed portion of system resources that is CPU capacity, disk bandwidth, network bandwidth and memory resource multiple virtual machines can run simultaneously on a same physical machine. So, guarantees available to each reserve set is valid despite the presence of other potentially misbehaving applications using other reserve sets. So, what does that mean? Conceptually I am reserving the resources giving to a set of tasks so, effectively for that of tasks with given set of resources allocated at in a static fashion becomes the definition of the virtual machine. For that set of processes other processes really do not in consist, because it has got that set of processes has got of view of the system which consists of only those resources allocated to it. So, if you have this resource kernel.

(Refer Slide Time: 39:37)



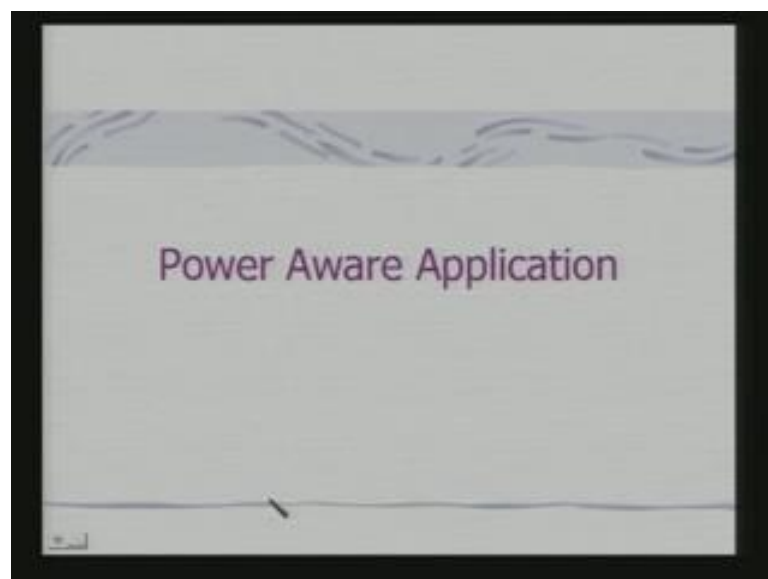
This is the Linux kernel this are the Linux processes you can have the Linux process running on the Linux kernel. But this other Linux process can communicate for a resource kernel. So, resource kernel makes what? Makes reservation of the resources for this Linux process with respect to the Linux kernel and these resources are made available to the Linux processes. So, the Linux process this process runs with reference to a virtual machine. Each process has got a resource demand. So, resource kernel handles the resource demand. So, effectively it makes reservations of those resources with reference to Linux kernel. So, Linux kernel knows that this much resource has been already allocated. So, that gets allocated to the Linux processes. So, the Linux process which refers to resource to kernel operates effectively in a virtual machine.

(Refer Slide Time: 40:34)



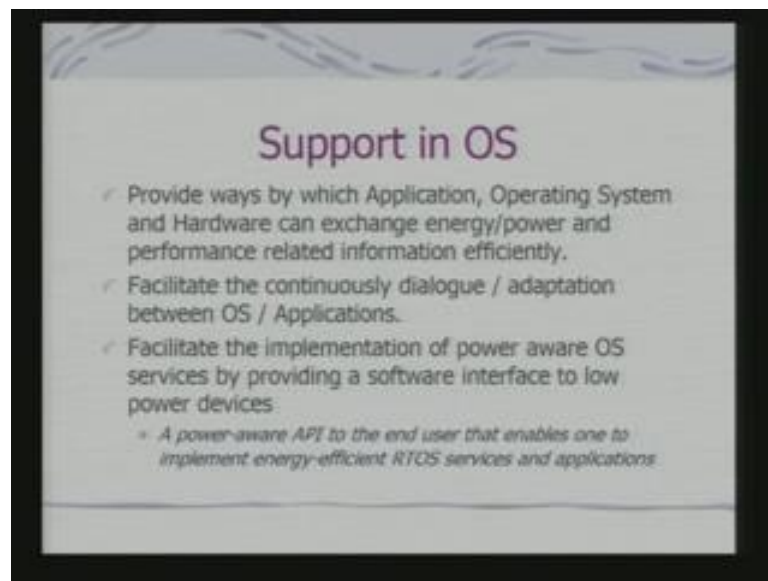
So, these are related to two basic concepts which we talk about reserve. Reserve is a share of a single resource there can be temporal reserves there can be spatial reserves. Temporal reserves are parameters which declare portion and time frame of resource usage CPU time, link bandwidth, disk bandwidth is that effectively a temporal resources. Spatial reserves amount of space it can be memory pages network buffers. So, everything gets allocated to a process by resource kernel and resource set is a set of resource reserves.

(Refer Slide Time: 41:14)



So, what we do with this resource reserves? Effectively this isolates one process from another process. So, there you do not disturb each other and this processor are guaranteed operating conditions and the timing parameter, because they have got guaranteed resources. Next, what we shall look at is that problem of dealing with power, because so far we have not considered power as a resource. But fundamentally in an embedded system power is a critical resource. Now, we have to see how this OSs can be adopted in order to manage power?

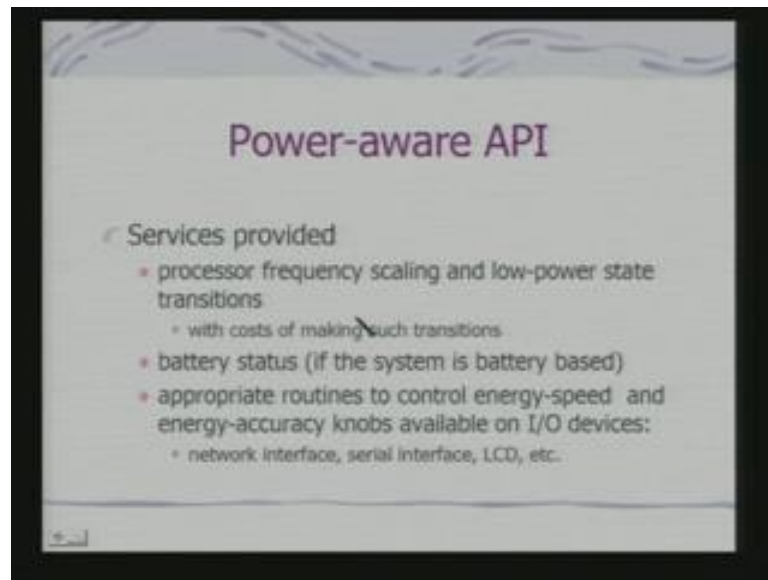
(Refer Slide Time: 41:59)



So, what kind of support we expect in operating systems for dealing with power? Operating system; it should have the ability for the application operating system and hardware to exchange energy power and performance related information efficiently, because then only the OS can thing about power driven scheduling policies. So, it should facilitate a continuous dialogue and adaptation between OS and applications. It should facilitate implementation of power aware OS services because, so, for services and what we talked about for a real time we wanted to have a deadline or an upper bound of the time regard to services. Now, you like to have services which would provide the management services. So, what we say the power our OS services the providing the software interface to low power devices. In fact, in a way what we are looking at? A power aware API to the end user that enables one to implement energy efficient RTOS services and applications. So, just like we were talking about Linux API in terms of the OS using the API preprocesses and threads and they become the schedulable entities,

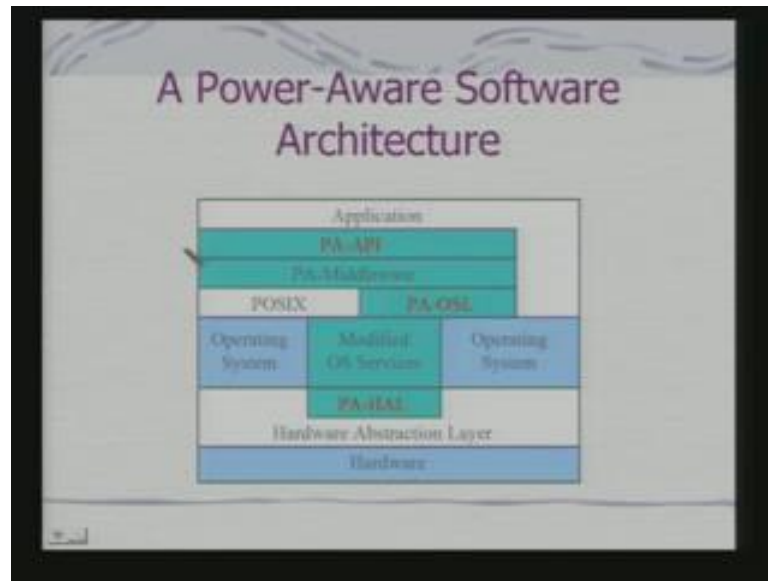
meeting the deadlines. In that same way we would like to have API to which we can create power aware services and this API implemented in a junction with the other way services can enable me to manage power.

(Refer Slide Time: 43:36)



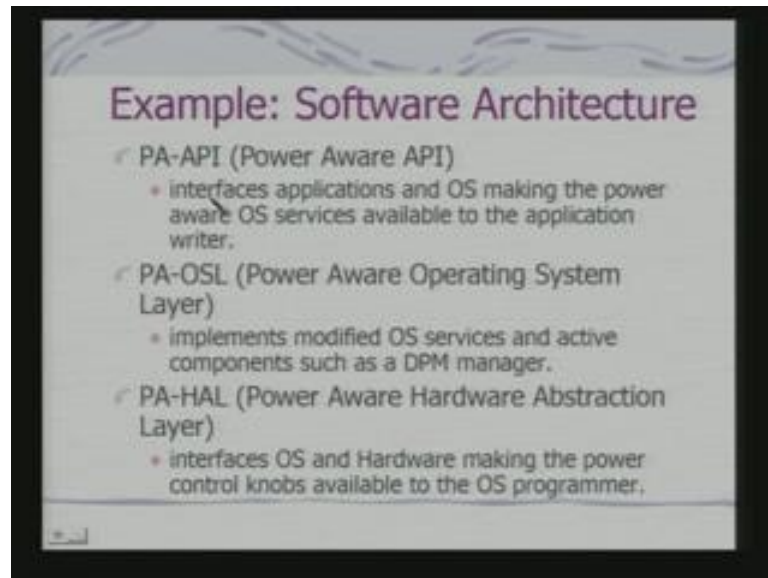
So, power aware API should provide services like processor frequencies scaling and low power state transitions. And we have looked at this in the context of power aware architecture when we said that each processor can have different stage corresponding to different amount of energy consumptions. As well as we have seen that I can select dynamically the supply voltages and have the frequency scaling which dictates my energy consumption depending on the deadline that example we had gone through. So, these kind of services is what API should provide for. The other issue is ability to monitor battery status battery becomes a device. So far we have not really considered battery as a device the power supply basically becomes a device I need an interface to the device. So, that I can monitor its health and I would need also appropriate routines are this are again system calls to control energy speed and energy accuracy knobs available on IO devices. That means, if you remember that we talked about flexible applications we discuss flexible applications with reference to timing consideration. Now, you talk you should have flexibility and ability to deal with flexible application in relation to its energy consumption.

(Refer Slide Time: 45:01)



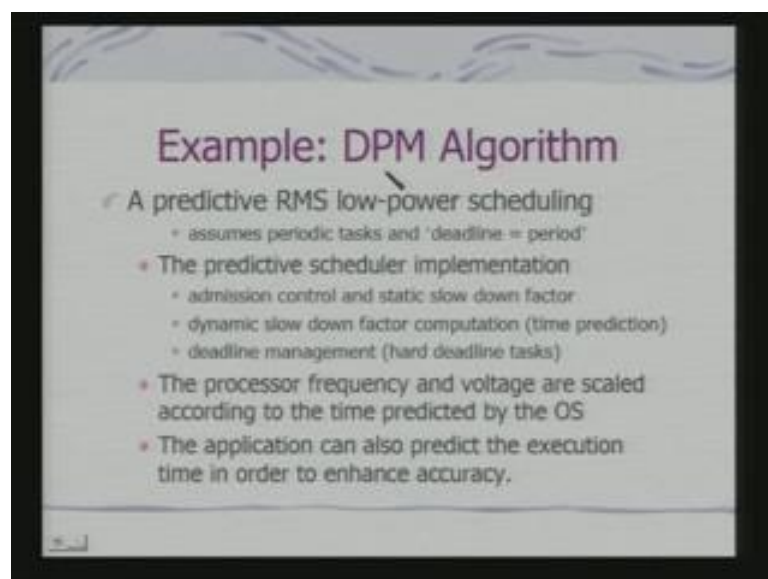
So, power aware API would something like this that you have got an application setting on top you have got a power aware API this is the power aware middleware. This is POSIX compliant OS that we are talking about with relation to the real time requirements. And this is power aware OS services that should be provided and that is services modified with an add-on. And that would work on a kind of an abstraction of the underline hardware because, if I am talking about a OS. Now, there would be a variety of processes would have different power saving stage we have seen that power pc has got some set of states which is different from other processor. So, there we need to have some kind of an abstraction so that when you talking about a generic service you can have an abstraction over this hardware's.

(Refer Slide Time: 45:54)



So, conceptually a power aware API should interface applications and OS making the power aware OS services available to application writer. Power aware operating system layer implements the modified OS services and active components such as a dynamic power manager. And power aware hardware abstraction layer would interface OS and hardware making the power control knobs available to OS programmer. What is a power control knob? Just take a simple example if I have the ability to modify the supply voltage ability to control the frequency of the operation. If it is a flexible application have the ability to modify them.

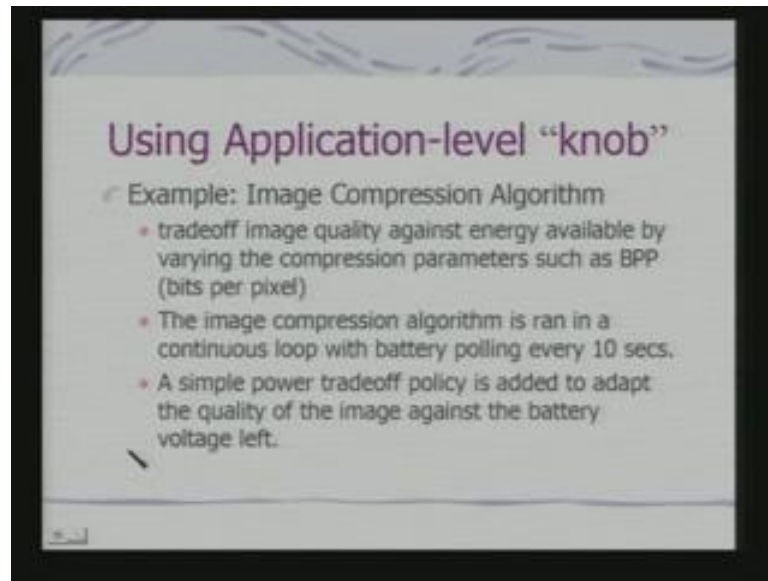
(Refer Slide Time: 46:41)



So, an example dynamic power manager algorithm could be a predictive RMS rate monotonic schedule, but it is a low power schedule. So, just like a rate monotonic can assume that periodic tasks and deadlines coincides in that of the period. So, it is a predictive scheduler implementation. So, there would be an admission control. So, I can have an admission control that whether the power budget permits that to be executed that as well as meeting the deadline and would have the static slow down factor, because if you have a longish deadline I can slow down the processor. I can have a dynamic slow down factor computation depending on how much time required to finish the execution and that has to be combined that your deadline management. So, what we say the processor frequency and voltages the scale and according to the time predicted by the OS.

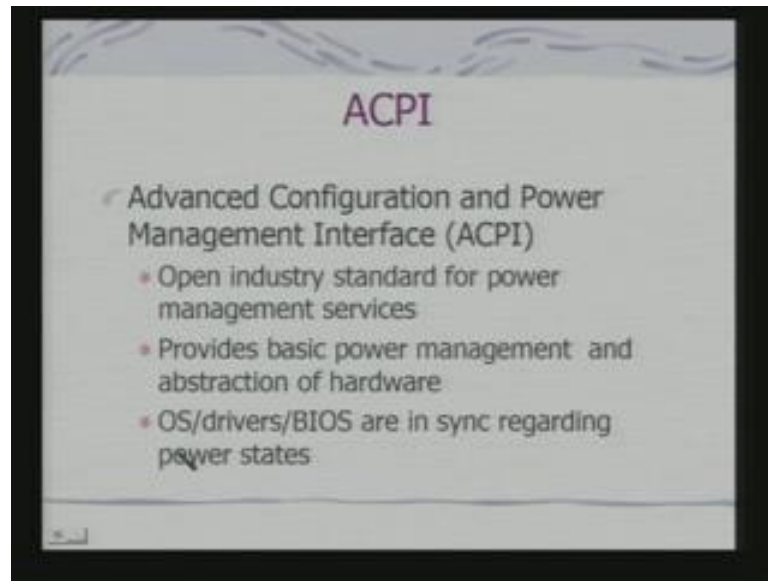
The OS can provide an prediction on the basis of the computation time required by the tasks. The application can also predict the execution time in order to enhance the accuracy. If you have this prediction then only you can do what? Alter the processor speed to deal with the energy then only you would know the define now slow down the processor whether I can meet the deadline. In fact, we have seen that the slow down the processor I can have optimal energy consumption as well as ability to meet the deadline. But for all this things we need to have a prediction of the execution time. So, if I have an RPM if I have a this RMS right monotonic scheduling. So, I know the periodicity along with if I know the execution time prediction correctly then I can decide over each period what may be the frequency of operation. So, that becomes a key show of dynamic power management algorithm are scheduler with the power aware application.

(Refer Slide Time: 48:43)



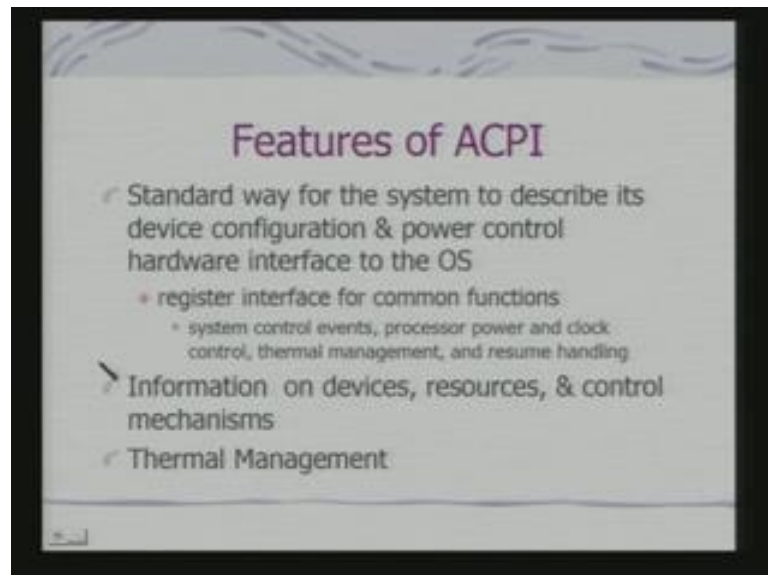
In fact, there also ability you may have a ability to use the application level knob which is actually m something very similar to your flexible application. So, it can be an image compression algorithm you trade off your image quality against energy available by varying the compression parameters such as bits per pixel So, the image compression algorithm is ran in a continuous loop with the battery may be polling every 10 seconds, because you are try to find out the battery life. A simple power tradeoff policy is added to adapt the quality of the image against the battery voltage left. So, again tradeoff is coming from where if your compression is less it will consume more memory when you are storing in the flash, but your processor is spending less time in compression. So, you are trading off. So, you have the in such cases particularly I told you in earlier also this multimedia application is got this flexibility. So, you can use this flexibility to deal with your deadline requirements as well as your even energy requirements. Actually there is again as standard just like we had a POSIX standard for real time you have a ACPI standard.

(Refer Slide Time: 50:01)



Advanced configuration and power management interface standard; this is an open industry standard for power management services the basic philosophy is built on the software architecture we had discussed. It provides basic power management and abstraction of the hardware.

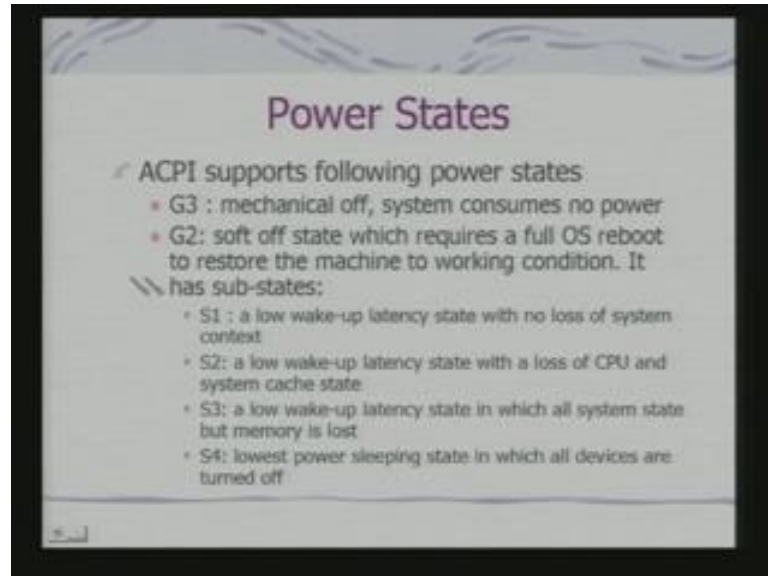
(Refer Slide Time: 50:19)



So, basically what are the features? It would provide OS for the system to describe its device configuration and power control hardware interface to the operating system, because OS means to know this information so, provides a standard way. It also provides

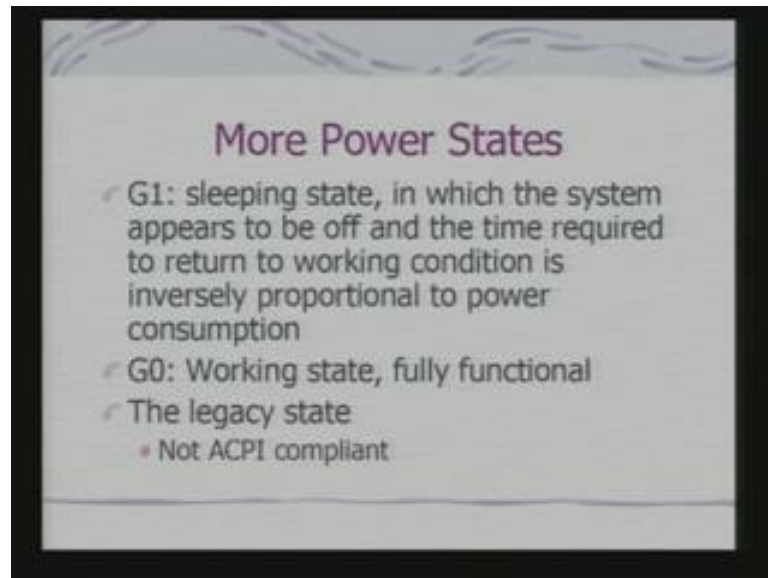
mechanism for thermal management and also an mechanism to specify devices resources and its controls mechanisms.

(Refer Slide Time: 50:44)



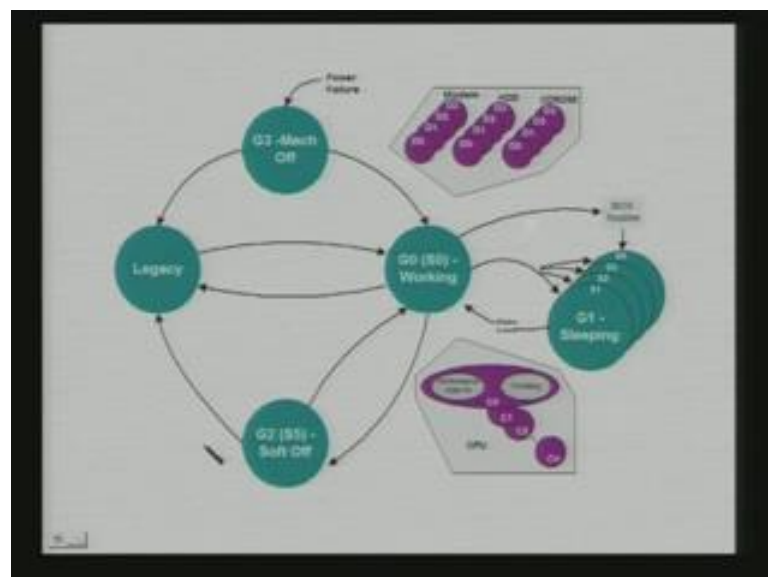
In fact, if you are talking about the hardware abstraction ACPI supports kind of a general powers states. A set of power state which are the abstraction of the power state of the processor may support. So, the power states are G 3 one is mechanical off system consumes no power. G 2 is a soft off state off state which requires full OS reboot to restore the machine to working condition. It can have several sub states S 1 S 2 S 3 S 4 and these sub states are depending on how much information is actually retained. Say for example, S 2 it is a low wake up latency state without loss of CPU and system cache state. So; that means, if the system has to reboot are initialize it knows what kind of system context information is available. So, that is how the states have been classified. So, it is not a processor dependent, but it is kind of a processor independent characterization of the power state.

(Refer Slide Time: 51:52)



Then you have got a G 1 a sleeping state appears to be off the time required to return to working condition is inversely proportional to the power consumption. But in this case what is interesting? I am not talking about rebooting of the system and G 0 is the working fully functional state and there is a legacy state which may not be strictly ACPI compliant.

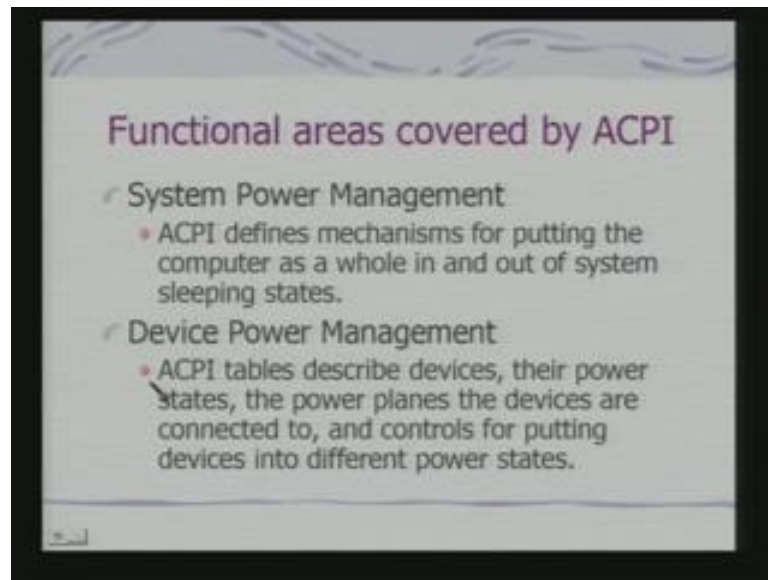
(Refer Slide Time: 52:15)



So, there should be a power transition diagram. So, if you had a mechanically off from mechanically go to a legacy state. And there you can go to a G 0 which is a working state

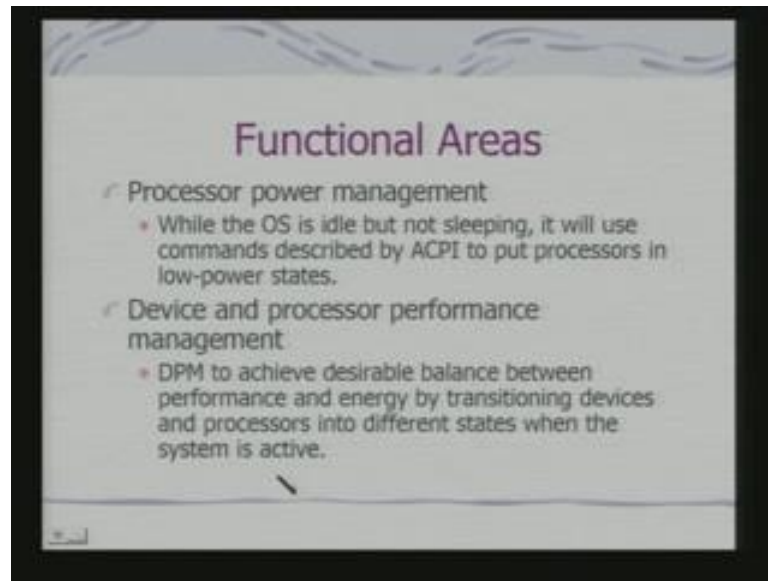
which is ACPI compliant. That means, you will can make ACPI calls complaint calls in this state you can go to sleeping mode, you can go to soft off mode which is your G 2 and where every variety of this different modes ion which you can go and your bios suit in basic would come in to play corresponding to the different modes in which system is.

(Refer Slide Time: 52:52)



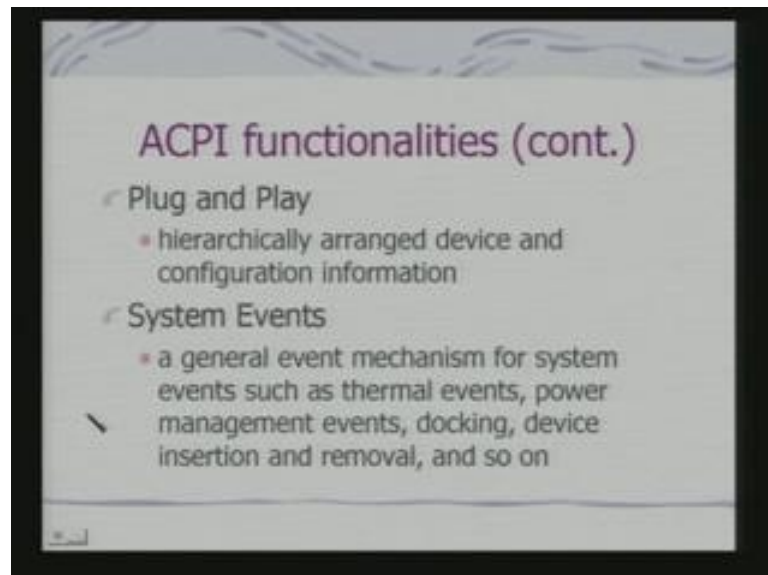
So, what are the functional areas? Functional areas; it addresses a system; obviously, power management device, power management, because devices have got their own power consumptions which is different from that of your processor. And describe devices the power states power plains the devices are connected to controls for putting devices in to different power states. So, it is a low level control knobs depending on the devices.

(Refer Slide Time: 53:19)



So, you have got processor performance management. So, you have the DPM dynamic power management software decide up on the scheduling scheme. And well the OS is ideal, but not sleeping it will use commands describe by ACPI to put processors in low power states if that is the need p.

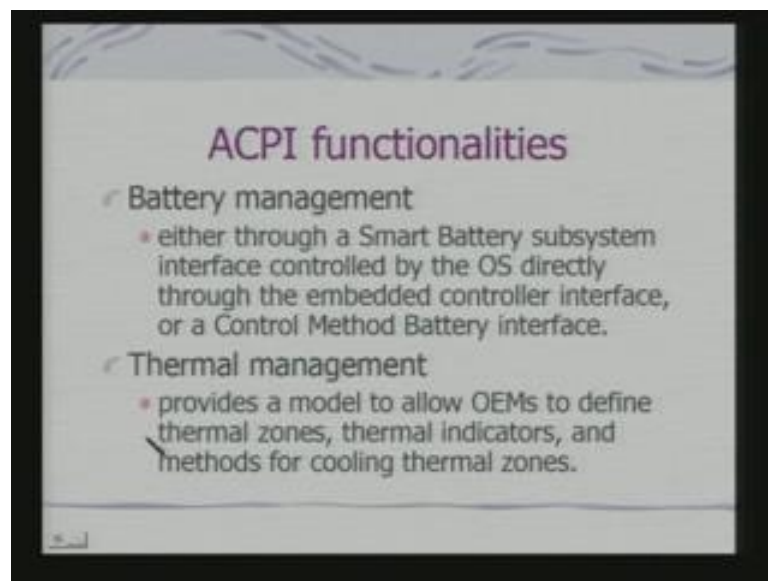
(Refer Slide Time: 53:42)



Then plug and play which is concerned to the standard applications. And that it of as is another interest is a general even mechanism for system events such as thermal events. Power management is like a docking device insertions and removal. So, this thermal

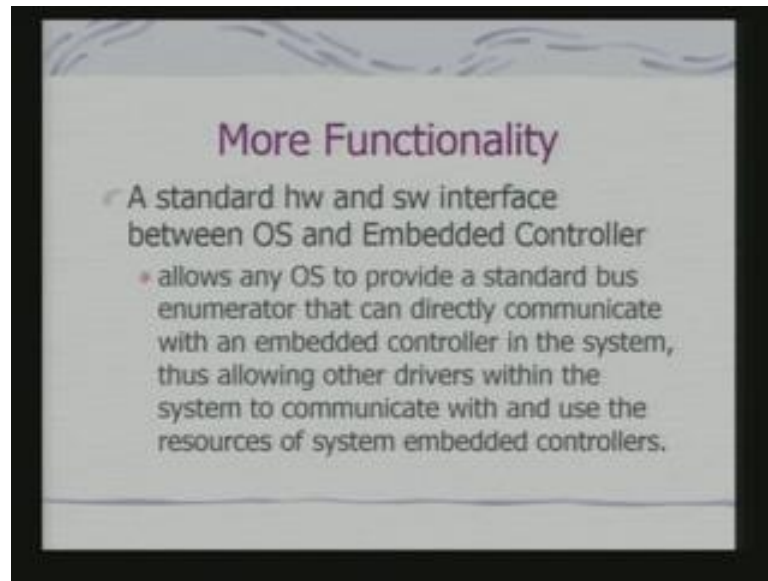
events thermal implication the temperature goes beyond temperature limit on the processor when its working in a working vigorously the temperature goes beyond the limit what kind of action to be done thermal shutdown. So, this is the thermal event who shuts round it? The OS should shutdown it and ACPI provides the complaint call and the interface detect such an event it also provides this battery management sub system; that means, it provides the standards for that interfacing.

(Refer Slide Time: 54:32)



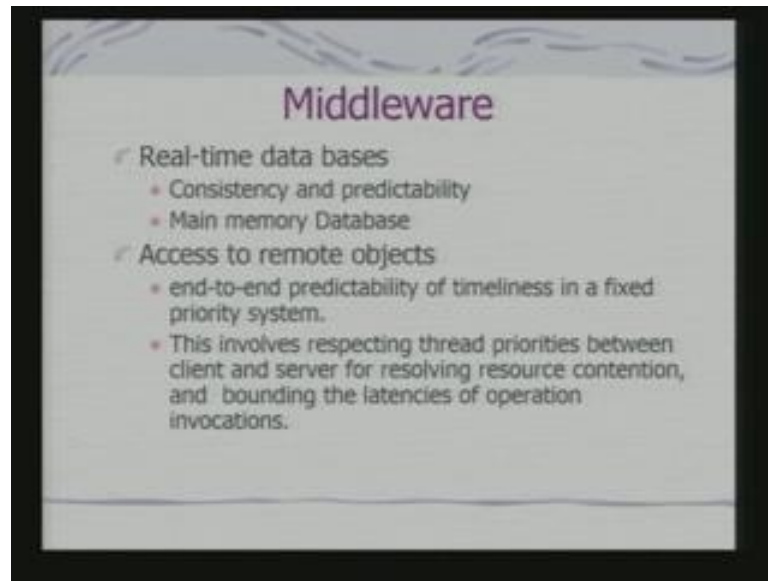
And thermal management provides a model to allow OEMs to define thermal zones, thermal indicators and methods for cooling thermal zones. So, this has to be controlled by some other device and those devices can be related to the OS itself. So, for example, I would like to put in a control signal should increase speed of the fan to cool those kind of interfaces should be provided for OS to manage the devices. So, this are basic ACPI functionality.

(Refer Slide Time: 55:01)



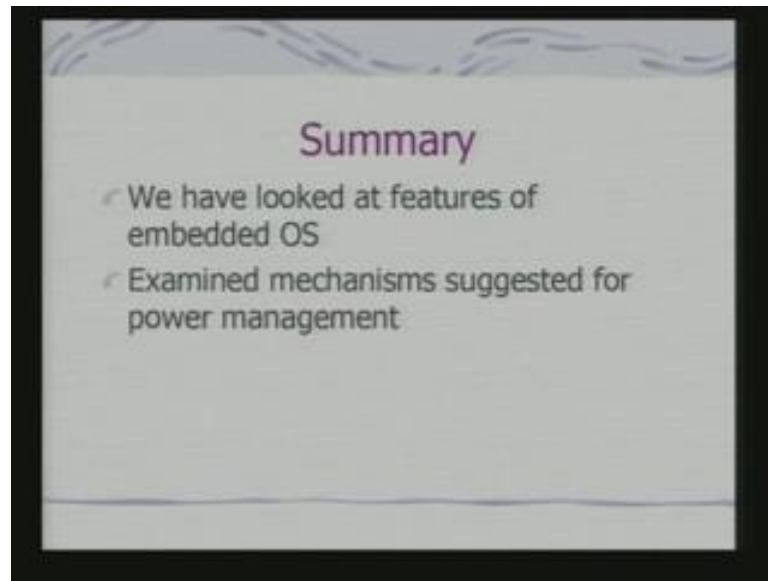
And the lowest level functionality allows this is the standard hardware software interface between OS and embedded controller allows any operating system to provide a standards bus phase interface. Allowing the drivers to communicate with and use the resources of embedded controller's standard bus these controllers. So, you have the basic interface provided through a ACPI through this devices for doing this kind of energy oriented management. So, that is a low level hardware software interface, because there would be hardware signals required also to provide this kind of functionalities and the service. The other thing which we have not touched up on related to the OS is the basically the middleware.

(Refer Slide Time: 55:54)



When they are really connected via the network system you have the issue related to the real time database and real time remote objects that is functionalities of the services in a client server mechanism. The basic issue in both this cases when it is a real time database there has to be a consistency, because when an update of the database has to take place that update has to take place the timing deadlines. So, this update this has to be implemented in many cases the databases are managed in the memory itself not with respect to the disk when you are talking about access to remote object what you try to provide? You try to provide the n to n communication with the timing deadline and In fact, this real time database others are becoming very important in the context of today's if you are gathering information from an external environment and trying to store the data since a network kind of an application. In such a scenario you need to maintain this databases and those updates have to be real to be in real time and the shoot satisfy the deadlines. So, this takes us the, to the end of the features that we had refer to for OS in an embedded system.

(Refer Slide Time: 57:18)



We have examined the basic features of the OS which is targeted for this kind of the application how to architect such a system. And also examine mechanisms suggested for power management. So, if you have any questions if you can refer to them. See the basic question is that time resolution. So, see you have the timers can be set to generate interrupt at fixed intervals. Now, these intervals are real times and the real time has to be internally represented by the OS. Now, OS allocate the fixed word size to represent the time then the resolution is fixed. So, if you the granularity of the time depends on how the OS manages that time data structure. It also is related to the underline hardware, because if you timer cannot support a variety of resolution are a very fine resolution depending on the clock at which is operating; obviously, when is OS supports lower granularity of time it has got no many.