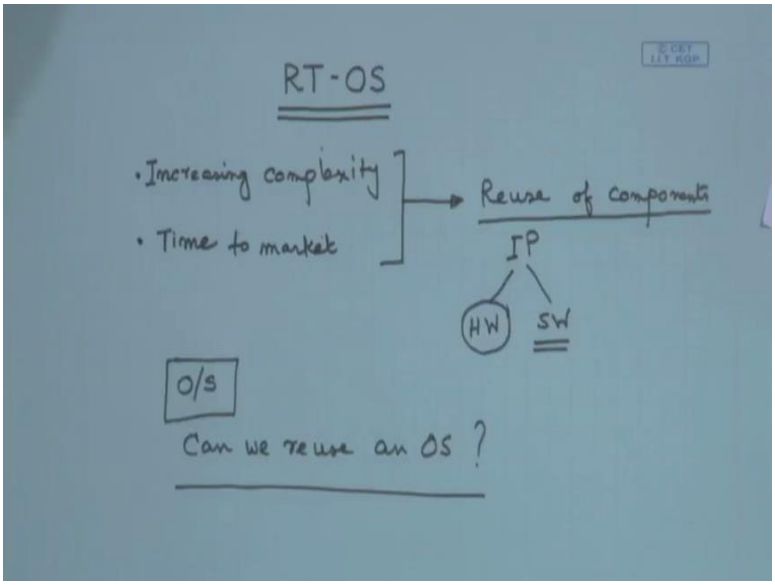
Embedded Systems Design
Prof. Anupam Basu
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 26
Real Time O.S – I

We have looked at how we can deal with the power issues in embedded systems.

(Refer Slide Time: 00:35)



Today we will start discussing about another very important part that is often involved in embedded systems design that is Real Time Operating Systems. As I had stated earlier that all the embedded systems need not be real time, but very frequently there are real time requirements posed to the embedded systems. So, the embedded systems need to be prepared for catering to real time constraints and for that often as we will see in today's lecture. Often the general operating system approach is not directly applicable.

Now, the points that really make the embedded systems design challenging is of course the complexity, there is becoming more and more. Nowadays we are not only remaining on one processor, but we are going to multiple processors also. And there is a mix of hardware, software; there mix of asics processor everything in that. So, while designing an increasingly complex system is more time consuming. At the same time we have got the stringent time to market constraint. We have to put it to market within time otherwise we will lose the business or whatever. So, these are two competing constraints;

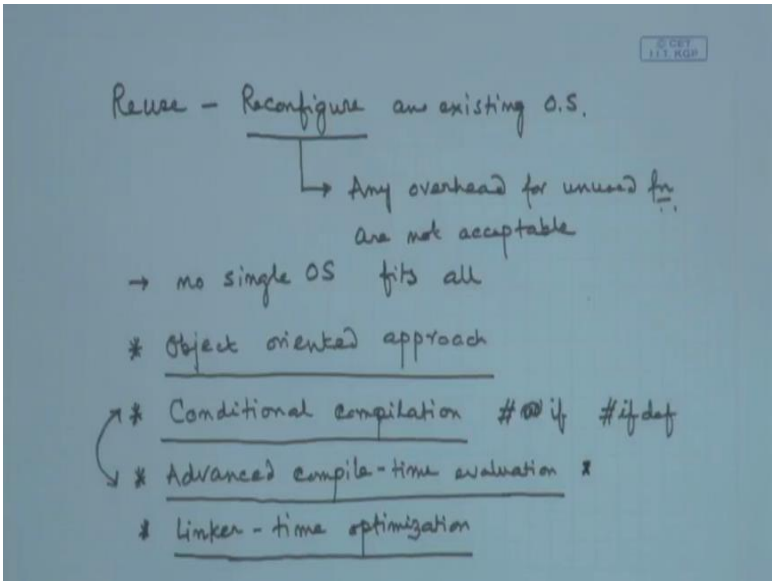
competing situations. And in order to overcome this hurdle the standard approach is for reuse of components.

Now, reuse of components we came across this idea in the name of IP, whereas IP stands for intellectual property. Intellectual property can be of hardware and also it can be of software. Usually in the common terminology and common usage we often say intellectual property is to be the hardware course the processor course, but it need not to be restricted to hardware course alone it can be extended to the software also. So, whenever we try to design a an operating system for an embedded system then we will also try to reuse an take a standard operating system and try to see whether we can reuse that.

Can we reuse the OS? Say I have got an operating system, but everything else of that operating system is not required. You see that whenever we are going to build an embedded system we have to keep into account mind not only the complexity and time to market, but also about the payload, it is a area, the size, the memory and all those things. So, we cannot make an embedded system to heavy.

So, an operating system that has got a lot of functions may not be required for an embedded system.

(Refer Slide Time: 04:41)



So, how can we approach that? So one way is reconfiguring or reuse means here or reconfigure an existing OS. So, there are some things which you should keep in mind when we are reconfiguring an existing OS for an embedded system we cannot tolerate any overhead for the functions which are not used; unused functions are not tolerated; not acceptable. In an operating system there are so many functions. Now for embedded applications I may not need all those functions, but I cannot also have the luxury of putting them in my memory and wasting memory space.

And also we will find we will see that no single operating system, there are so many different applications in which embedded system can be applied. So, no single OS fits all; that means no single operating system can satisfy all our needs. So, one of the very popular approaches is to take an object oriented approach. What do you mean by this? All of you are familiar with the term object oriented approach where we have got a class and we can build specializations of those class subclass and all those things.

For example, I have got a general class of schedulers which has got some features, but for a particular application I need a special scheduler which will have only a specific policy, specific time quantum, specific tricks embedded for that scheduler so that can be generated as a class subclass of the general scheduler class. So in that way different parts of the operating system functions we can create instantiated versions of the general class. So, in that way an object oriented approach will enable us to reconfigure our system to this.

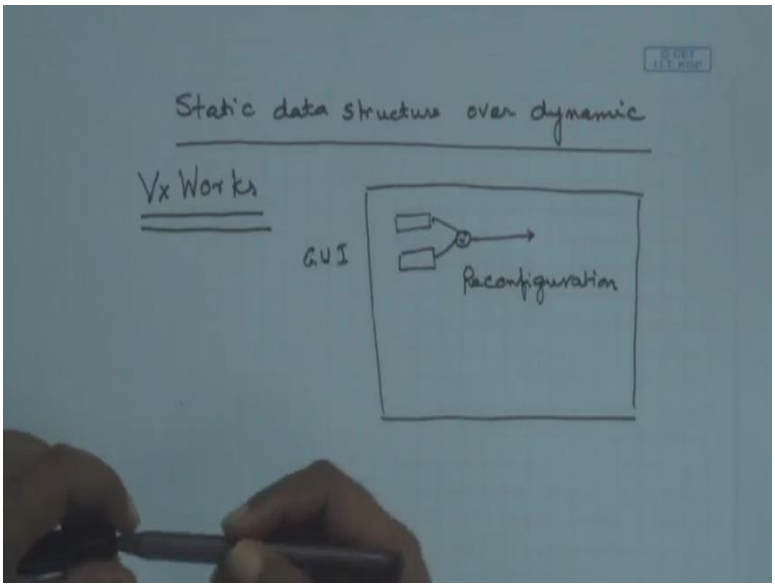
Another very popular approach is; but not very popular by the way it is a may be effective that is conditional compilation. I think I had mentioned about this earlier in some other context where we used some preprocessor commons like predefined if or if def type of statements, which means that the particular part of the code will be compiled under some conditions, but so that will be the compiled code can be therefore reconfigured depending on your particular application. But the major cons of this approach are that the code becomes highly unreadable with so many this things, so that is not very much used.

Sometimes some advance compiler level decisions which is similar to this, some compiler time at evaluations; compile time evaluations are used where we use; while we compile we decide whether this is very similar to this. So, these two are only same more

popular so I may not lay much stress on this, but let us talk about this. There is linker time optimization. Now this is quite obvious for as I said that for all applications all the modules are not required. Therefore, I can optimize there are different modules, but whenever I generate the OS, I compile with the link time; I link some of the modules and leave out the rest that is a very important way of reconfiguring the operating systems.

Now let us not lose the perspective. Why are you trying to do that? We are trying to generate a quick version of an operating system. But in order to that I want to generate a quick version of the operating system that will create to some specific need and specifications. I do not want to make it unnecessarily bulky; I also do not want to make it inefficient. So, in order to do that I just want to; it is a component base design sort of. I will just take the relevant component and plug them.

(Refer Slide Time: 10:56)



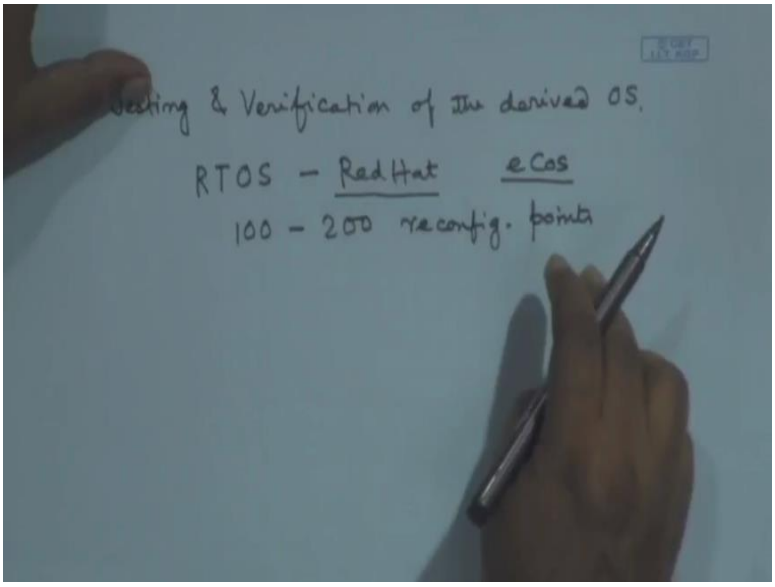
In order to make the operating system efficient also another thing you should keep in mind that often we prefer static data structure over the dynamic ones. The linked list and all those we would like to avoid and we would like to replace them with arrays; very well possible. So, that is another approach we take in order to increase the efficiency of the performance efficiency of real time operating systems.

Now, such reconfigurations are very much practical. And there are operating systems like Vx Works; it is an operating system quite popular and is used for embedded system design which from the GUI level it provides you with the GUI which allows you to

select the components that you want to tie up together, you tie them up and you can reconfigure your system. So, there is a reconfiguration panel using which you can create your own version of Vx Works. So, Vx Works you can check at your leisure time look into the details of Vx Works and its features. So, that is a very popular operating system.

Now, the other problem of base; now there is a downside of this also; downside of this is that whenever we are reconfiguring an operating system then that operating system must be tested thoroughly.

(Refer Slide Time: 13:11)

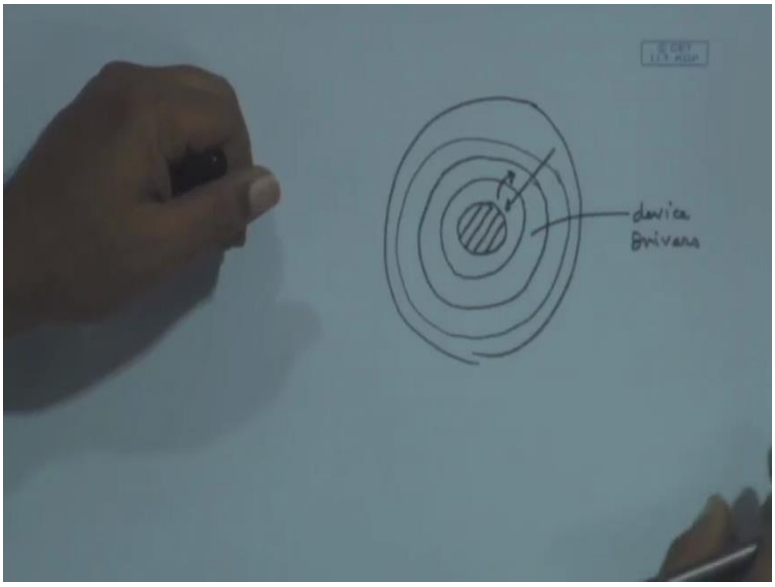


Testing and verification of the derived OS is very important, because when I created a reconfiguration I must test it thoroughly. Now on the other hand if I had developed only one dedicated operating system I would have tested that operating system over once. But since I am reconfiguring it time and again it every time whenever I am generating a reconfigured OS I need to verify it again. For example, typically in the open source real time operating system by Red Hat Linux has about 100 to 200 reconfiguration points.

So, if they has different varieties, so think of the multiplicity of the different versions that can be generated from this- say one typical subversion is the e Cos operating system which is a version real time operating system from Red Hat Linux. Now it has got so many configuration points so it becomes very difficult to verify each and every one of them. That is another downside of reconfiguration approach.

Now, we will look at particular diagram. Now let us quickly have a look at the tasks of a typical operating system. If I ask you what is an operating system: an operating system is a set of software packages which takes some help of hardware support of course, but it actually manages the resources efficiently so that we get a virtual environment. But that virtual environment may not be that important for real time or embedded systems, but in a way yes it gives us the compiler front end so you can talk in the; at through high languages etcetera.

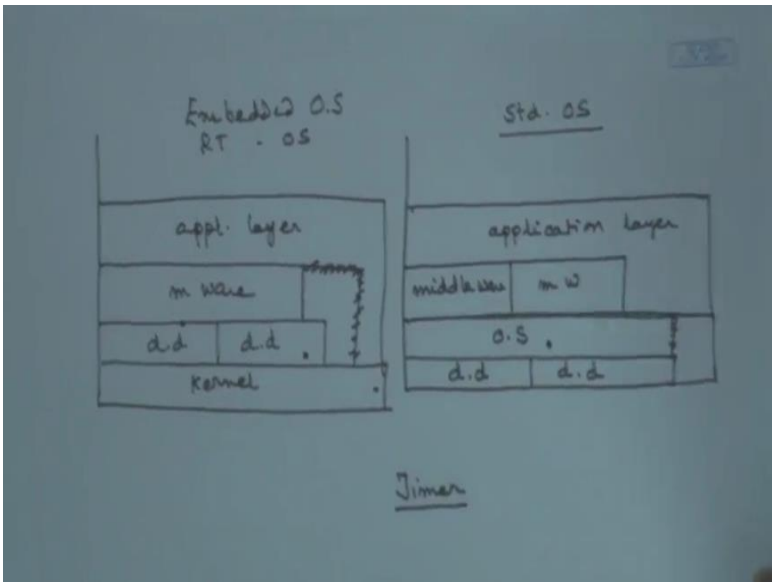
(Refer Slide Time: 16:03)



But one of the major; see if I think of an operating system. The operating system has got at the core there is hardware; there is hardware at the core. And just above that hardware are the device drivers or kernel is a there are kernels and then there are device drivers, and whenever we make a call we make call to the kernel and the kernel selects the device drivers. So, the device drivers are not usually directly called by the user functions.

The reason you know that if there will be a device which is being used by one particular process another process can throw it out or can invade in between and the results can be undesirable and everything that is. And above that there are compiler layers, database and all those things ultimately the application layer. They are different middle were here. So, that is a layered view of any operating system.

(Refer Slide Time: 17:21)



In the case of standard OS (Refer Time: 17:18) embedded operating system; I would like to draw this diagram. On this side is a standard OS where I can have d.d- means device drivers there are device drivers here and the operating system oversees that the OS is above that. And then on that the OS is actually what I draw is OS is a little goes beyond, because not only the device driver the OS takes care of some other thing. And then there is a there is a middle wax here I write m w. And above that there is a application layer clear. So, that is the standard OS variety.

So, what happens here is that whenever I write any application that application that if it wants to access the device drivers it cannot bypass the operating system, he has to come to the operating system. Some application may bypass the middleware, but not the operating system; the operating system is spreading all through.

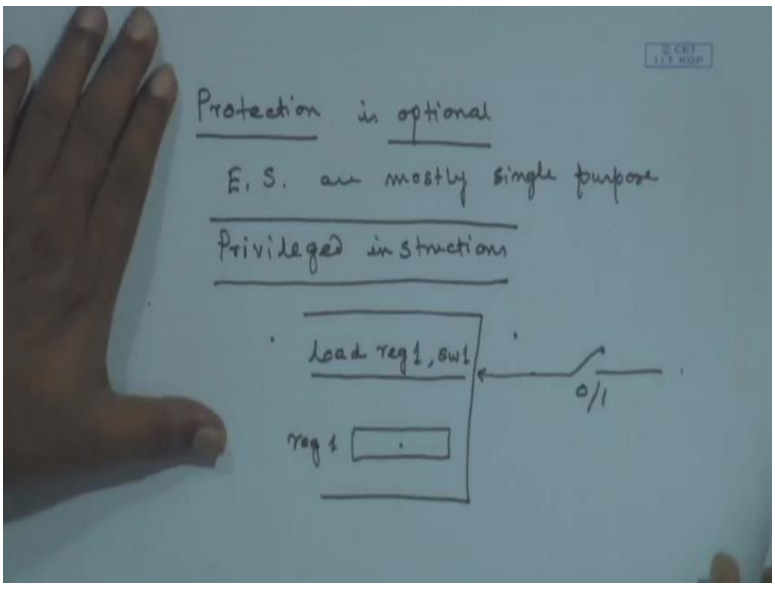
On the other hand if I draw the embedded OS then I will draw it like this that I will have the kernel. Of course there, so kernel is also here. Now the device drivers are here, then in the same way I have got the middleware and the application layer. So, the difference this is the look of the embedded OS or real time OS or RT OS. Now here what happens when an application layer can directly access the kernel, can also directly access the device drivers. I think the middleware need not come upto this it can middleware I can also bypass the middleware. And an application program can directly access the device driver.

So, there is not necessary that there has to be an intervention of the operating system all the time. Just to illustrate this point let us take the example of a PC based operating system windows. Say for example; there the keyboard, the mouse, the disk, the network, everything is under the control of the operating system. So, whenever we any process windows process for example, once to access the disk it has to go through the operating system layer as this shown here.

On the other hand, embedded system being often single user systems or mostly for single purpose applications; it is possible that it will directly access the particular device or if disk or if flash memory is attached to that it will directly access that, it is not necessary that will go to through the operating system: number 1. Number 2: typically when we design the windows OS we are thinking of all possible applications such a general purpose platform we are, but we are no but no particular application actually requires all the devices except probably for real time system that is very important.

One particular device is very very important that is timer. Except for timer there it will not be the case that all the processes are using the same set of devices. So, depending as the devices will vary it is not appropriate to all the time to go through the operating system to the device drivers. So, that is the major difference between operating system; I mean embedded operating system and other operating systems.

(Refer Slide Time: 22:40)



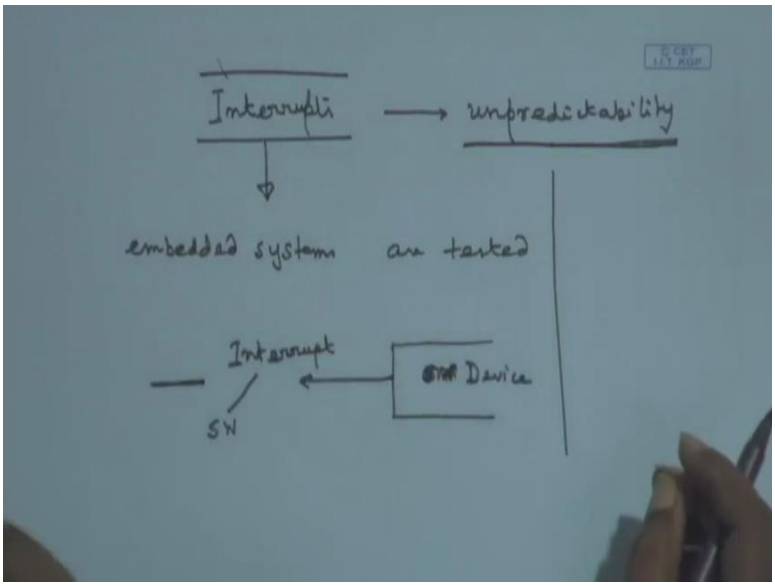
Another point is protection. Protection is a very important feature in any operating system; if we recall you will see that the program segments of the different processes are also protected by base limit registers. Similarly, their protection across the devices, protections against mutually I mean simultaneous access, critical sections all those things are there. But protection is optional in case of real time operating systems, because that is not because the embedded systems are mostly design single purpose; are mostly single purpose.

Another very interesting thing that we found in general operating system was the privilege instructions; I hope all of you recall that. What are privilege instructions? Privilege instructions are those which can be executed only through the operating system mode or the kernel mode. So, whenever there is a set of instructions which are unprivileged, there is a set of instructions which are privileged. Whenever the system finds that a privileged instruction is being executed a system call is generated and you go to the operating system kernel level and execute that from that layer. The reason is this privilege instructions were introduced to ensure the protection that a particular device cannot be simultaneously accessed by more than one and all those things. But here privilege instructions are not required, because again we are designing it for mostly for single purpose.

So see for example, there is a switch here and that switch is connected to the system. And directly the application and the system is running some application program where I can write load say reg 1 switch 1. So, a particular register reg 1 can be loaded with the value of this switch 0 or 1. In order to load this I need not take the help of the kernel or access that. That is in general true, but nowadays because of security reasons a lot of security issues are coming up with embedded systems specifically for the defense sector and very critical sectors, protections are also being thought off. How protections can be incorporated at the operating system level as well as the hardware level.

That is a very recent phenomenon that we are all worried about, but in general a major difference of embedded operating systems with that of standard operating system is that protection was not that considered to be not that critical at that level.

(Refer Slide Time: 26:40)



Now another very vital thing is interrupts. Now interrupts always turned out to be point of unpredictability, because the very nature of interrupts is that they are asynchronous, they can come at any particular point of time. And we want the performance of embedded systems and real time systems to be predictable. One point is of course meeting the time constraints, but also we need the predictability. And that is how the interrupts have to be very carefully handled, but here in general operating system that can lead to in unpredictability, but how do we handle interrupts in embedded operating systems.

So, we can allow any particular process to connect to the interrupt. Now when we come to an interrupt and we service it then that typically in normal operating systems what happens with the interrupts, whenever the interrupt comes we go to the kernel, but here we are not restricted to go through the kernel. Here we consider that embedded systems are tested, we have tested that I am assuming that so. If the interrupts come they are the interrupts of not to be restricted to the OS. See that the point that I want to make is, interrupts where it was mandatory that the interrupts will be processed through the kernel or the operating system in general systems.

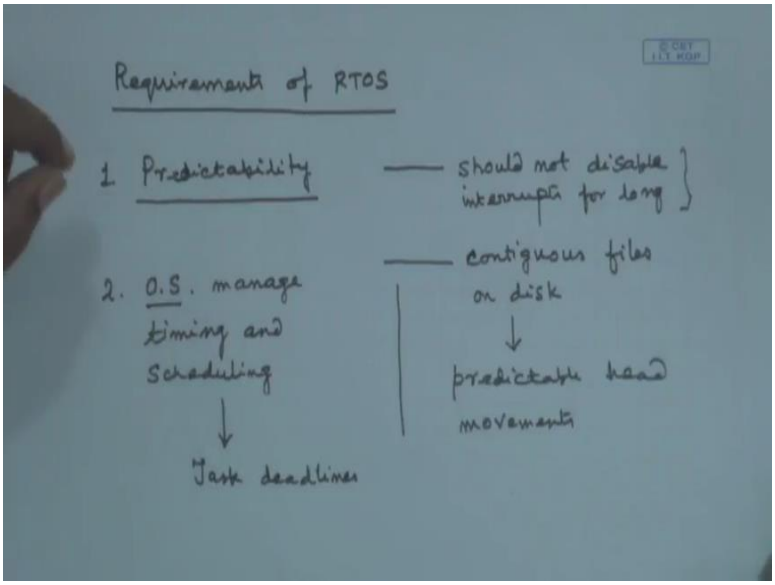
But in our case we need not restrict it to the operating system, a device can directly connect to the interrupt line of the processor because they are already tested and protections are not always required. And efficient control over the variety of devices is

required. So, it is possible that particular software say or say particular device connects to the interrupt and corresponding to the interrupt I have got a software. And it can be directly done without going through the operating system.

But, there is one problem here that if I want to have composition of actions against a particular software I want that one more than one software to be selected depending on different conditions that I cannot do here because I am directly utilizing the interrupting pin by the device here. So, the operating system is no longer required for this.

Next is real time capability which is very important now. What is the real time system if we define, what is the real time operating system? The simplest possibility fine definition will be- a real time operating system is an operating system that supports the construction of the real time systems. And real time operating system is an operating system that supports construction of real time systems and so that real time behaviors can be delivered; that does not communicate much.

(Refer Slide Time: 31:23)



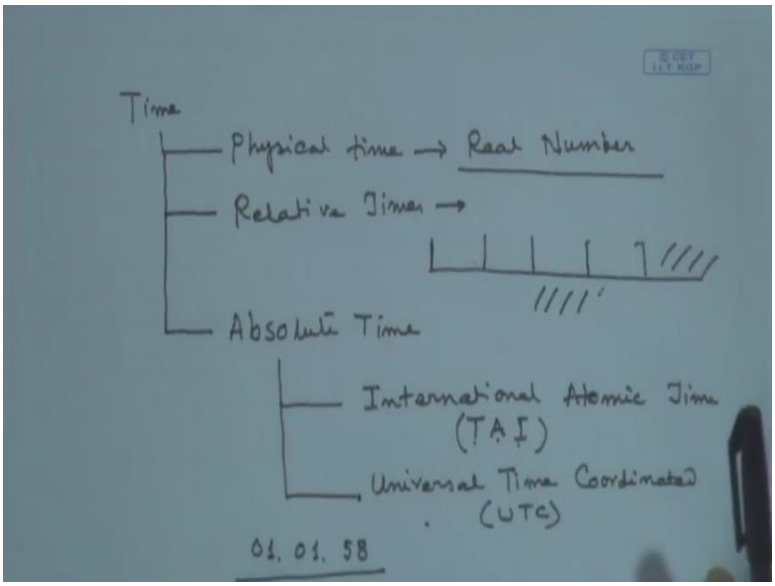
So what is required is number one: what are the requirements let me just put it as requirements of RT OS. One is predictability: the timing behavior must be predictable. For that I cannot disable interrupts for long; should not disable interrupts for long, why? If I disable I do not know when exactly the interrupt is coming. So, if I keep the span of interrupts disability long then I can miss the interrupts corresponding to my predictability of the behavior will be affected.

Again another thing whenever we need this, contiguous allocation of the file is important. Why? You can think of many such other requirements- contiguous files are required because (Refer Time: 32:58) on disk. Typically the files are not stored contiguously. So, I do not have any apriori knowledge of where the things are accordingly the access times can vary, I do not have any predictability over the access time; one sector can be at some other point etcetera, etcetera.

So, if I have contiguous files then I have got predictable head movements, that lead to predictable head movements- that is helping in predictability. The other constraint is the OS should manage the timing and scheduling of the works. Now this brings in to point the task dead line, if you just think of your typical usage in your PC's you are whenever you compile a job or you want to execute a job you typically do not allocate any deadline with respect to that. Now for real time operating systems the deadline is very important and the operating system must do the scheduling in such a way that the timing and the deadline must be honored otherwise it would not be a real time behavior at all. So, that is important and also very resolution timings may often be required. The resolution of the times it cannot be say- I mean it is a quite predictable it will come within 1 hour that will not certainly work for most of the applications that is not allowed.

So, these are the two major requirements that we see. Now we talk of time; now this is a thing that I often discuss why is it called real time. We always say it is a real time system, but have you ever thought why we call it a real time and what is a unreal time. So, what is time I mean how to represent time for that matter?

(Refer Slide Time: 35:52)



Time means one thing that can be the physical time which is a real number; which is a real number. But, there are relative times also. See for example, the clock ticks I say that this event must take place after 5 ticks; would take place here anywhere here and another job must take place after 3 ticks; so anywhere 1 2 3 here. Now this relative delay between these two, I have got no in claim, no idea about the absolute time that is there. That is typically whenever we; Now obviously whatever we do, we do in computers we do not in the real numbers is zone we do in the discrete time zone.

In discrete zone also this is a relative time. However, small (Refer Time: 37:22) are high rather however high my resolution be still it is a discrete. So, that is relative time. Another time; so in a real time scenario I cannot say the job must end after 5 ticks. Now the 5 ticks can be 1 hour right, so that is not acceptable. What we are worried about is the real time or the absolute time that is the clock time. And there are two varieties of absolute time that is used is international atomic time. It is known as TAI, because the French name I cannot pronounce it properly temps atomic international or something like that time is something like whatever; time atomic international.

The other thing is universal time coordinated, have you seen this timestamp anywhere? Where? You will see it in the emails also or UTC. You will see it in emails; you will see it in airline reservations everywhere. Now this international atomic time was purely based on some (Refer Time: 39:14) and it is not based on any artifacts, whereas this one

is based on the astronomical standards since this one is based on astronomical standards and there is always some variation of the speed of astronomical movements, therefore it requires some corrections at times.

So because of that there may be little variation in that and that has to be corrected, it is said that in January 1st 1958; these two clocks are synchronized, but after that this has changed a little bit. So, we will see how we some more requirements of real time systems in the next lecture. And after that we will move to some real time operating system scheduling problems.