Real-Time Systems

Prof. Dr. Rajib Mall

Department of Computer Science and Engineering

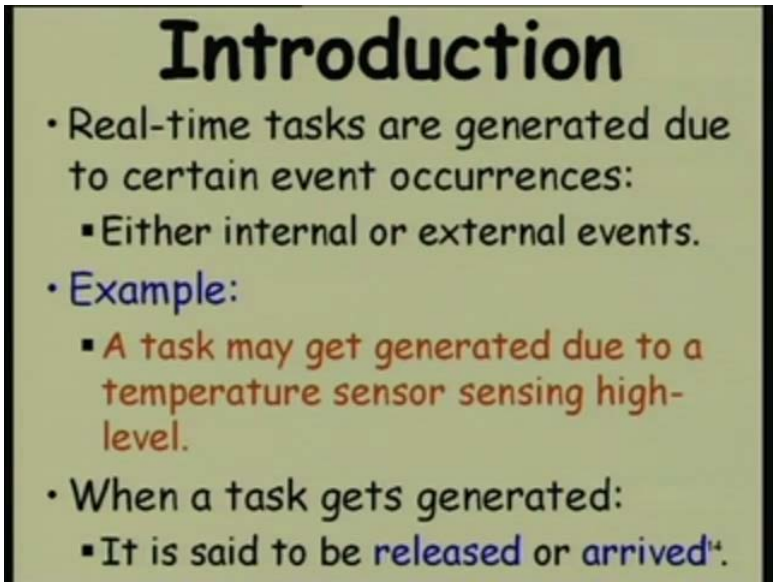
Indian Institute of Technology, Kharagpur

Lecture No. # 06

Basics of Real-Time Task Scheduling

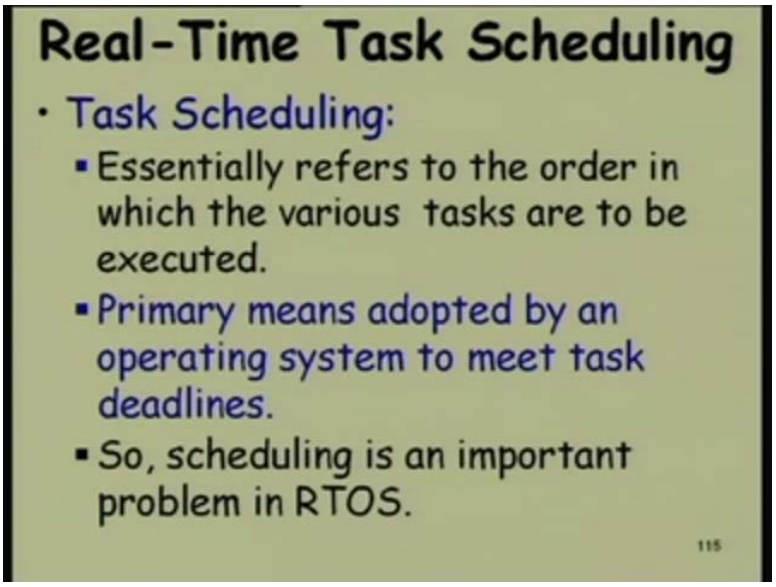
Let us get started. So, we had looked at the basics of modeling time can be referred to different types of timing constants and modeling, and we had seen some little bit of scheduling terminologies. Now, let us get started from that point. So, the basics of real time task scheduling, and as we proceed, we will look at sophisticated tasks scheduler starting from very simple tasks scheduler to more and more sophisticated schedulers. It is not that the simple ones are not used; the simple ones are the one that is used maximum, because many small devices exist.

(Refer Slide Time: 01:06)



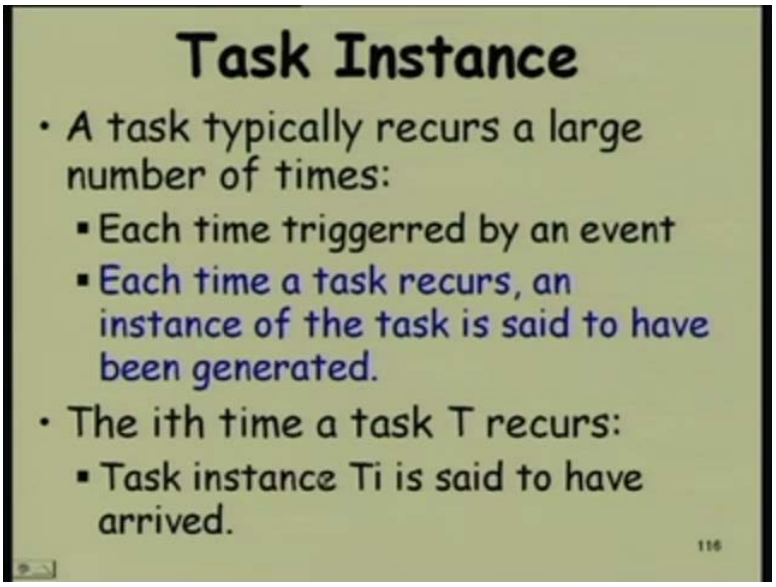
But the complex one is also used. So, you need to look at those various types of task schedulers. So, these are some of the terminologies, we had looked at like a task is released or arrived when the task is generated due to a event.

(Refer Slide Time: 01:19)

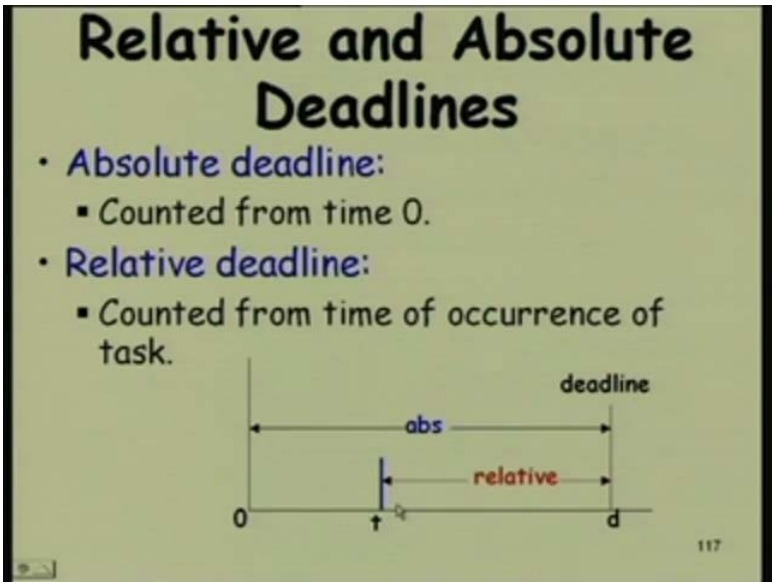


And then we had said that scheduling is a important problem; possibly one of the most important component - possibly the most important component of any real time operating system. And we are looking at this terminologies of task instants, a task instants arises again and again, new instances occur as events keep on happening, and we said we denote the i eth instance; i eth time the task triggers by T i.

(Refer Slide Time: 01:35)

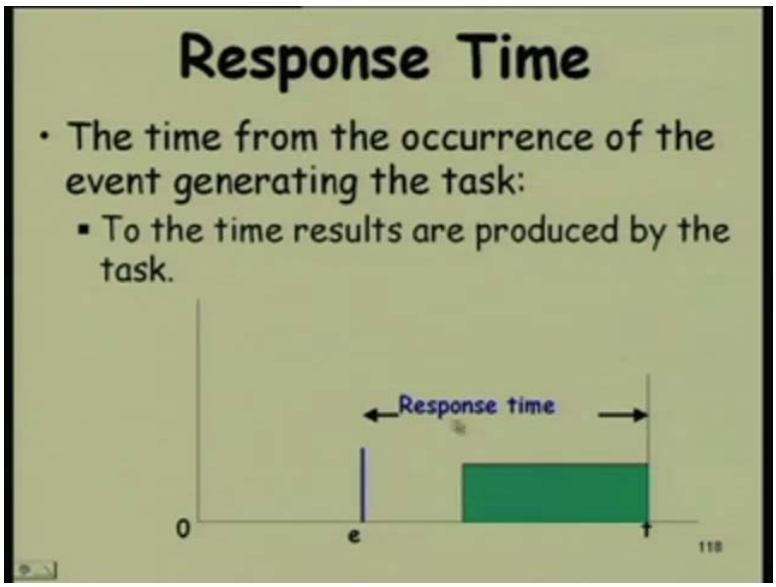


(Refer Slide Time: 01:59)



And then, we were discussing about the absolute and relative deadlines, this is a term, we will use quite often, absolute deadline whenever we will tell we will mean that it is with respect to time 0 0, deadline is measured with respect to time 0; whereas, the relative deadline is with respect to the task released time.

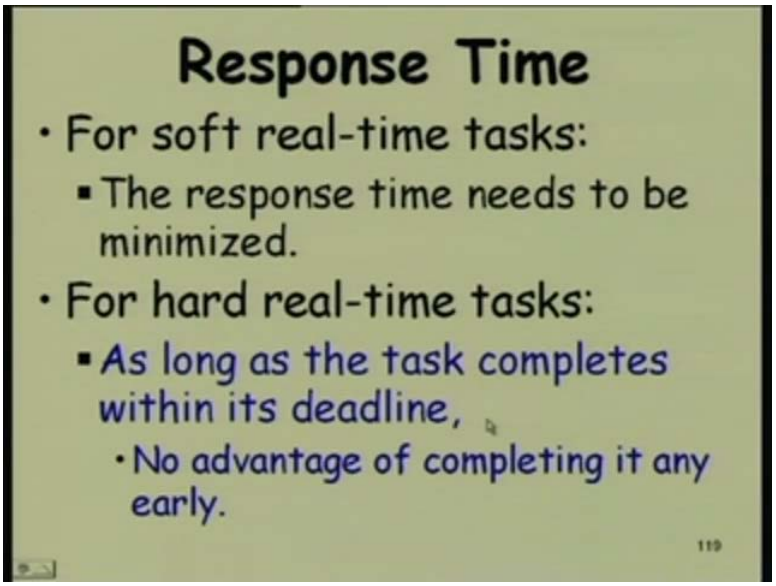
(Refer Slide Time: 02:25)



And response time, I had discussed this also a terminology in traditional operating systems, the time a task starts not starts actually released, this is the time and the task

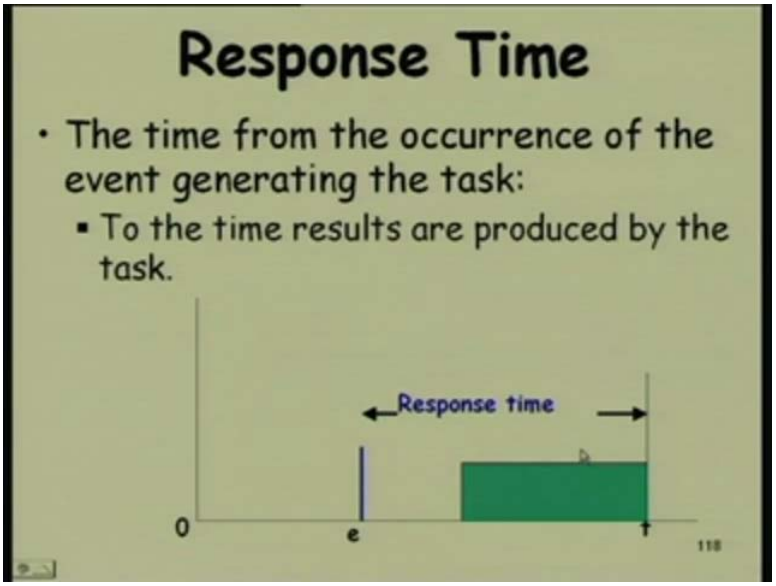
actually starts executing. So, the time the task is released to the time, and the time the task completes and produces result that is the response time.

(Refer Slide Time: 02:53)



So, one thing that possibly we can say here just to clarify the questions that you were asking, that response time is actually not the characteristic of a task response time – response time is the characteristic of a rest event.

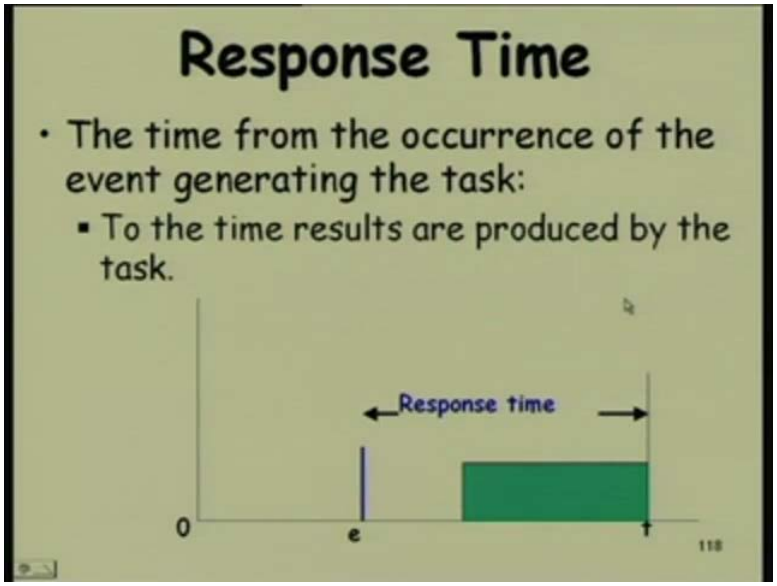
(Refer Slide Time: 02:58)



So, the characteristic of the event that is sort of the heater is the event right. So, this event is generated by this task. So, the response time of an event or an action taken by the system; that is the response time. It is not proper to say that response time with respect to a task, not a very proper way to tell that. The proper way is that the event generated by the system in response to this, what is the response time for this events heater sort of the response time is this. If the task generates some other events say executes possibly the response time of those events, I hope that the answers the questions that you were asking that it would not be proper to I mean for the simple case, where the task completion is coincident with the response event.

We can say that the response time is this one the task completion time, but the event can also occur as the task executes. So, those the response time of those specific events. We will have to talk of it is not except for the simple tasks. We will not associate response time with more complicated tasks were events keep on occurring right

(Refer Slide Time: 04:51)



Sir.

Yeah please yeah please.

The response time is till the completion of an action by the system sir or just start of the recognition of, what action has completed? No, see we were not really taking of actions here, we are talking of events generated by the environment and events produced by the

system right. So, once the event is produced, but the environment or may be with respect this event, that is a stimulus response event or a response response event. We will talk of the response time.

The response event, it is an event not action actually for example heater sort of signal is generated. It is not that when exactly the heater becomes zero degree or something the sort of signal was produced that is an instant of time. So, we will talk between two instants of time two events instantaneous events.

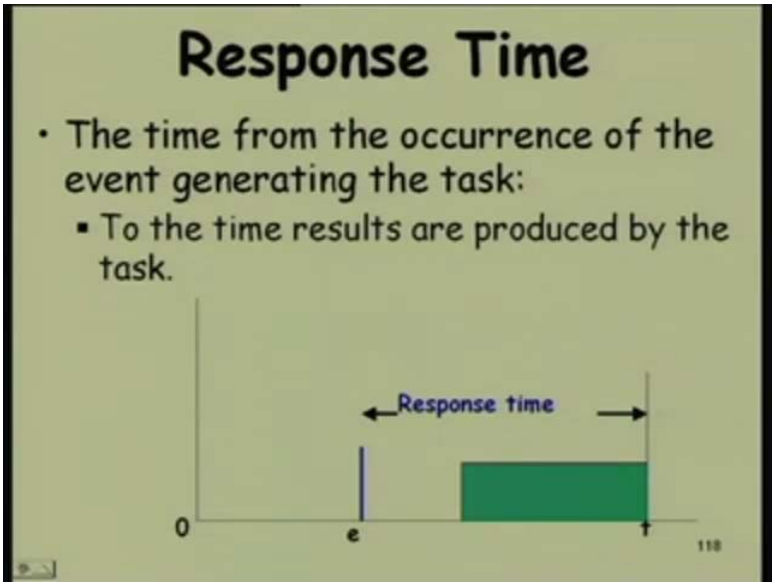
So, in this the response time will be from temperature becoming greater than sensed to high yeah. And the heater should on.

Exactly

From…

Exactly, but for this simple example.

(Refer Slide Time: 05:57)



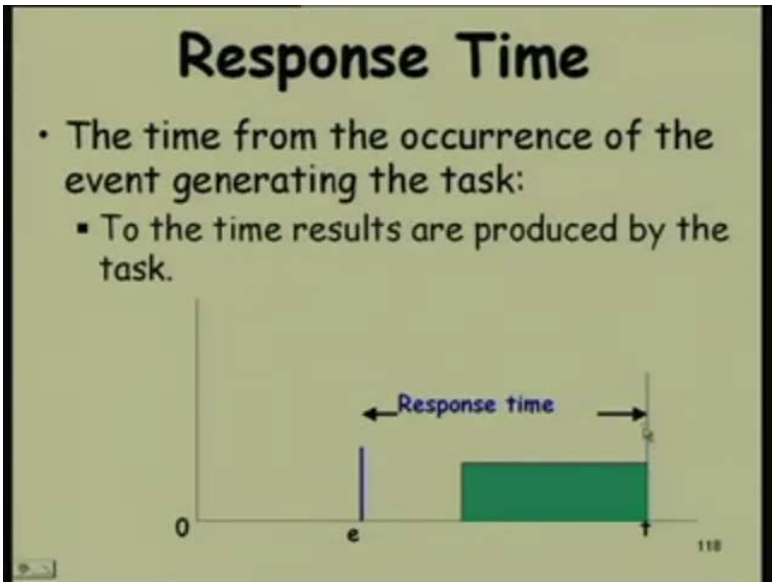
Sir, if the if the case is like that then event generated by the system in response to the event, if that is a durational event like if it is not instantaneous. It has got some duration.

So, then we will have to associate that when the duration event see his question is that, what will be the response time? If the system produces a durational event, so in that case,

we will have to again like we were doing earlier associate. When the duration event started and when the duration event ended. So, we have to consider those things the points of time the point at which the duration event started, and the point of time at which the duration ended. So, we will have to the response time will not be a very simple answer there for some cases, you will say that the response time is when the duration event started in some cases, we will say that it also depends on when the response event ended right the durational event ended.

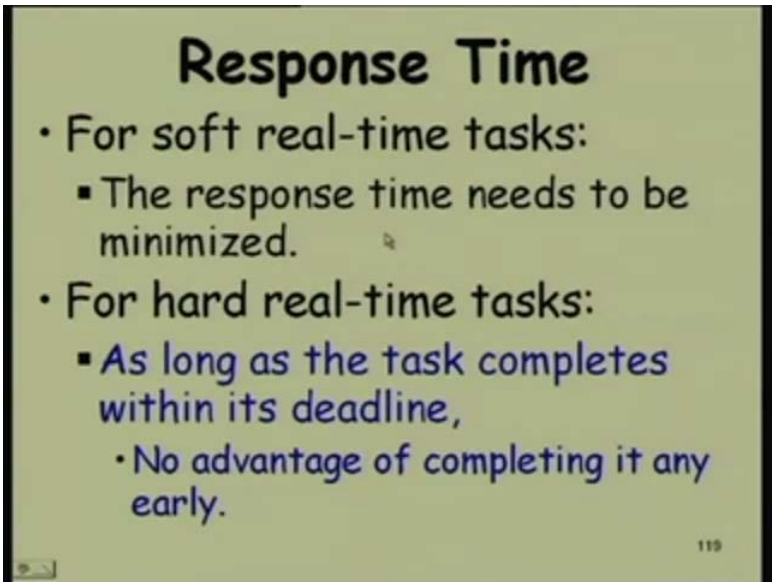
Right for the the simple case, that we are taking of where a instantaneous event occurs by the system in response to some event either a stimulus event or a response event. We are saying that that is the response time.

(Refer Slide Time: 07:21)



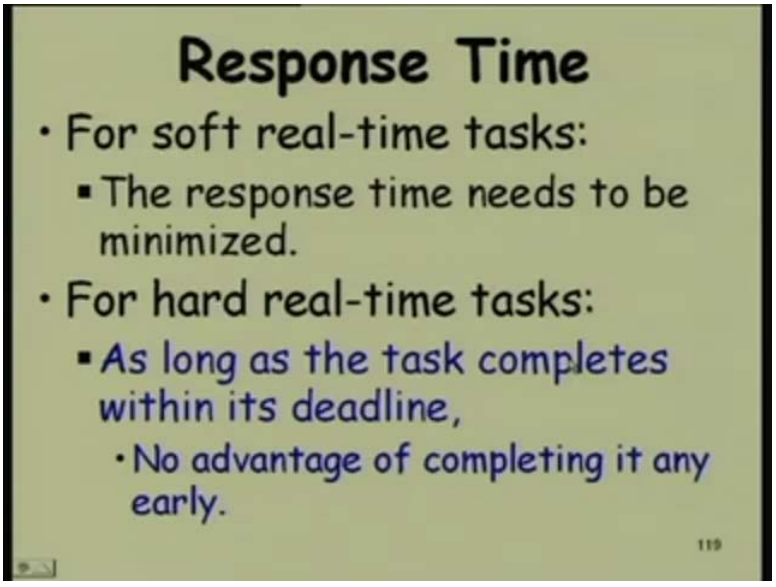
So, response time is associated with the event not with the task, but for the simple case, where the event coincides with task completion written this as the response time at the simplest case is that for a soft real time task. The objective is to minimize the response time that is, let us say if you are press the keyboard on a computer. What happens is the operating system recognizes it? And then shows you back the character and the terminal, is it not?

(Refer Slide Time: 07:27)



So, if you press the key the character does not appear in the terminal. You said, it is not very responsive, but if it is able to schedule the tasks, such that you can immediately see your response response to the you press the key immediately appears on the terminal say that the system is very responsive right. So, that is the traditional terminology.

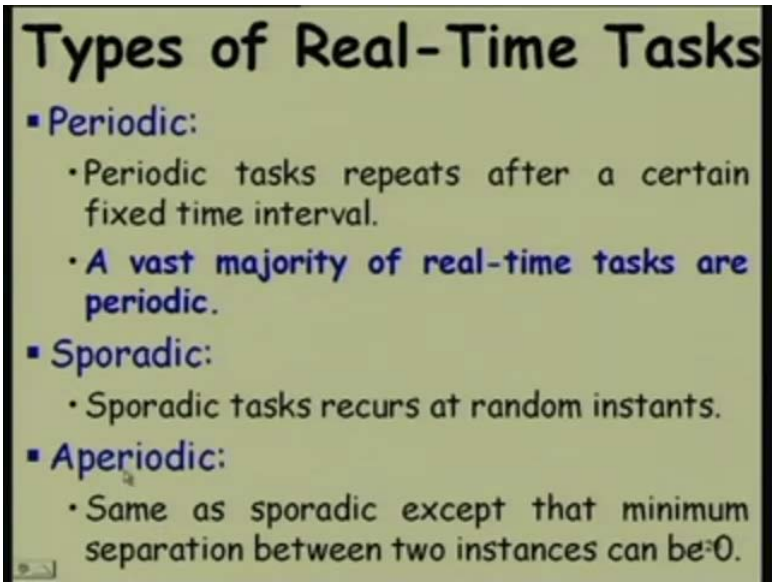
(Refer Slide Time: 08:19)



But here in the traditional operating system was that we minimize the response time, but here what we are saying is that minimizing response time holds no advantage, we will we will not minimize the response time in all the schedulers that we will consider. We

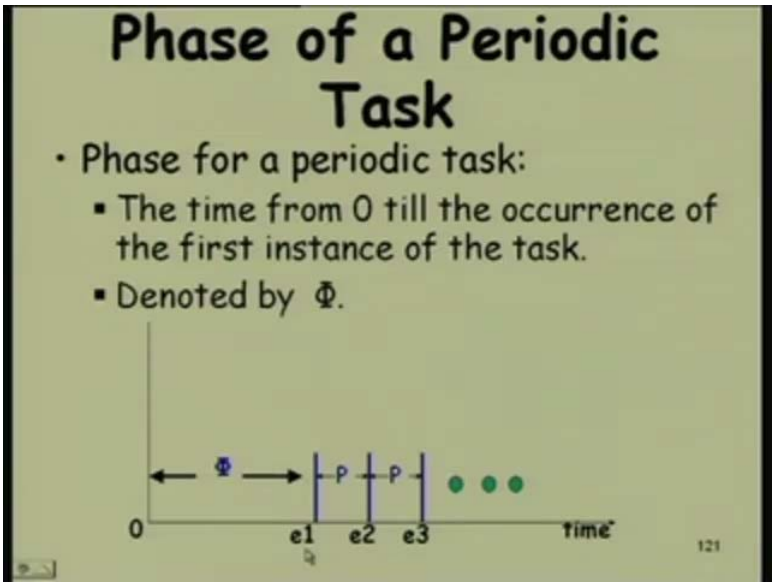
will just try on completing all the tasks within the dead line; that is the main objective here.

(Refer Slide Time: 08:39)



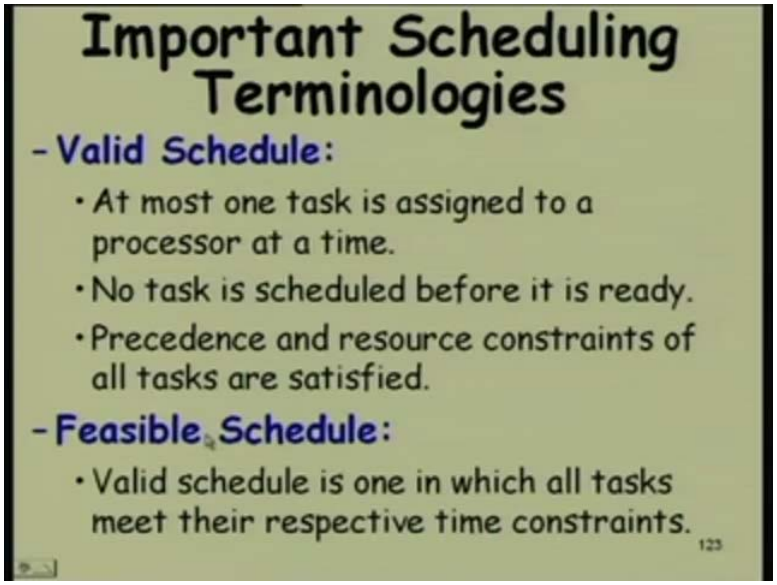
And then we are looking at the three categories of tasks that are common in a real time situation the periodic ones, and we are saying that in some systems all tasks are periodic, and in any system majority 99 percent or something tasks will be periodic. Sporadic tasks this occur at random instants, and periodic similar to sporadic, but the minimum separation between task instances is 0.

(Refer Slide Time: 09:13)



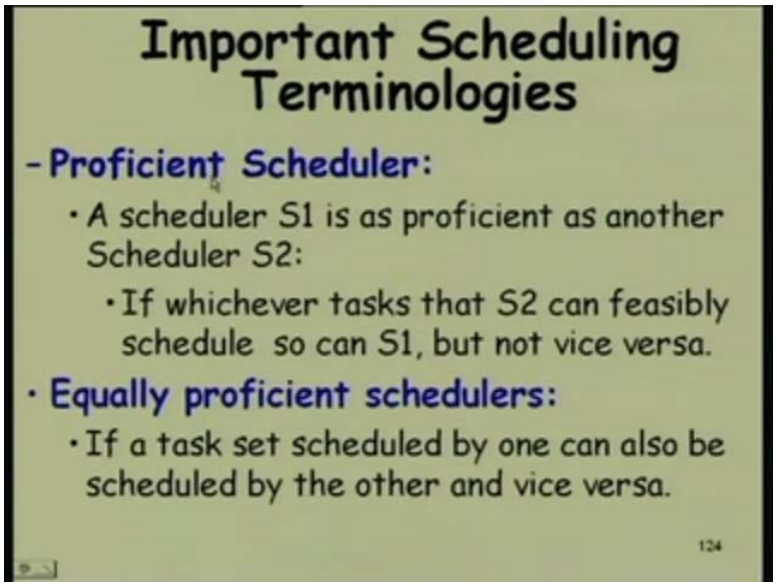
And we are also discussing about the phase is the time that elapses, before the first instance of a periodic task starts, and then it keeps recurring with some period. So, we will use the term phi denote the phase of a periodic task. So far a task T i, we will use phi i is it is phase.

(Refer Slide Time: 09:42)



And we are also looking at the terminologies: valid schedule and feasible schedule. So, some of you just remarked that a feasible schedule is also a valid schedule. Yes, a feasible schedule by definition is a valid schedule.

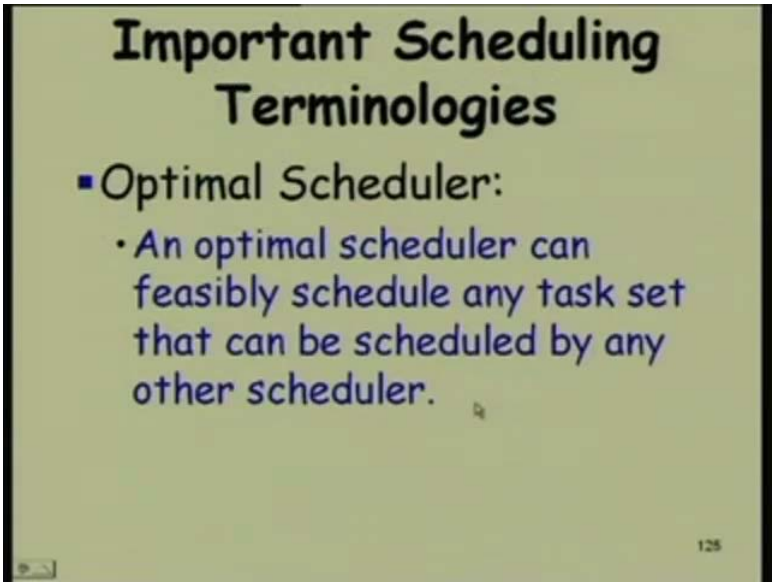
(Refer Slide Time: 10:01)



And then proficient scheduler, a scheduler S 1 is more proficient then another scheduler S 1 S 2, if S 2 can feasibly schedule sorry; if S whatever feasible schedule produced by S 2 S 1 also can produce some feasible schedule for that given a set of tasks.

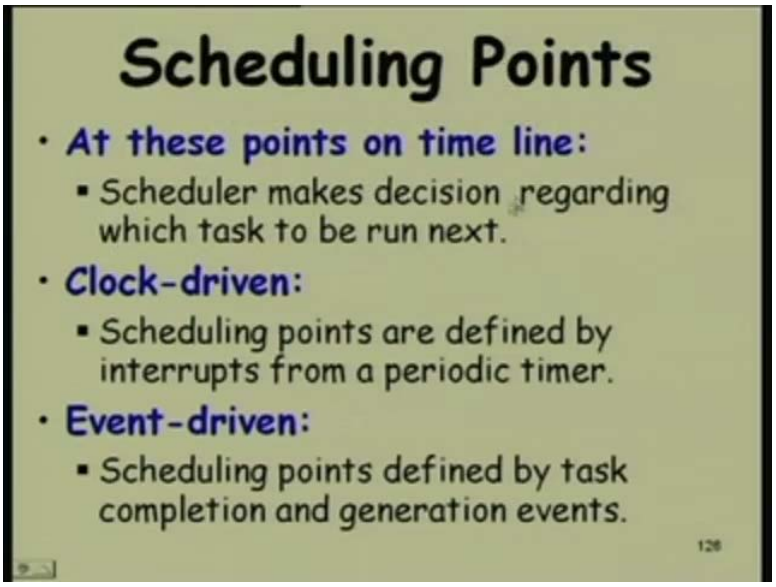
If S 2 produces a feasible schedule, so also can S 1, but there will be some schedule some set of tasks for each S 1 can feasibility schedule, but S 2 cannot. So, that is why we are saying that S 1 is more proficient than S 2 equally proficient will also use this term that for whatever, task situation S 1 can produces feasible schedule. So, can S 1 and also vice versa.

(Refer Slide time: 10:58)



And we also discuss about an optimal scheduler, where if there is any scheduler that can produce a feasible tasks, a feasible schedule for a set of tasks. So, can the optimal scheduler. So, not saying that the optimal scheduler will try to schedule any set of tasks no, if a some other scheduler can do. So, also can optimal scheduler that is the terminology, let us proceed now.

(Refer Slide Time: 11:26)



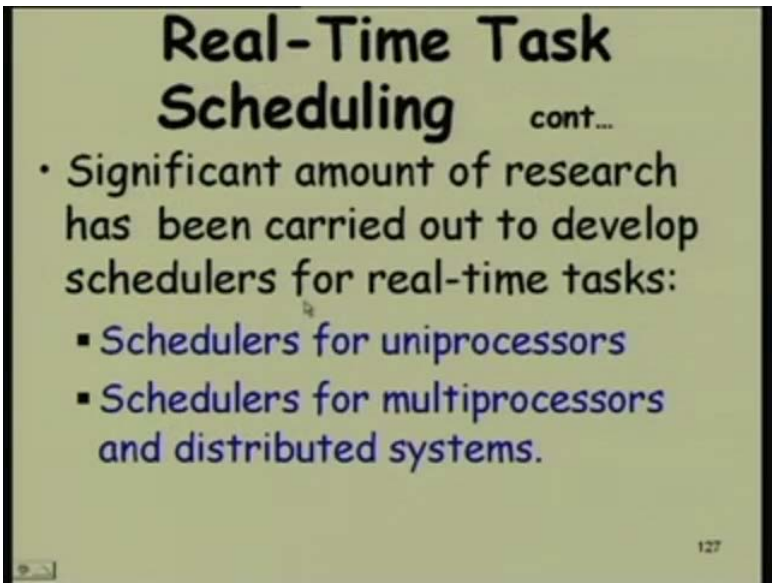
Another very important term, we will use almost every time, we talk of a scheduler is the scheduling point actually; these are points on the time at which the scheduler will try to find out which task to take up next. So, the scheduler does not run all the time. It runs at specific points of time as the computation proceeds, and at those points of time it wakes up and tries to decide which one to run next.

So, that is a scheduling point. A specific points on the time at which the scheduler becomes active and tries to decide which one to run next in a clock driven scheduler, this is the terminology, we will use in a for those scheduler, where the scheduling points are define by interrupts for a periodic timer. So, every time a timer interrupt comes. So, the timer is a periodic interrupt periodic interrupts come from some timer, and each time a interrupt comes, let us say 10 millisecond or may be 100 millisecond the scheduler wakes up and tries to find out what to do next.

So, that is a clock driven scheduler on the other hand. We will have another category of schedulers, where the scheduling points are defined by some events which are not really clock events here. Let us say a high temperature is sensed or let us say fire condition is sensed. So, the events need not occur according to some clock. And what once the event occur tasks needs to be released at that time the scheduler wakes up and tries to do find out. What to do at that time which task to take up and also when a task completes the scheduler also wakes. So, the scheduler does not wake up really periodically it wakes up

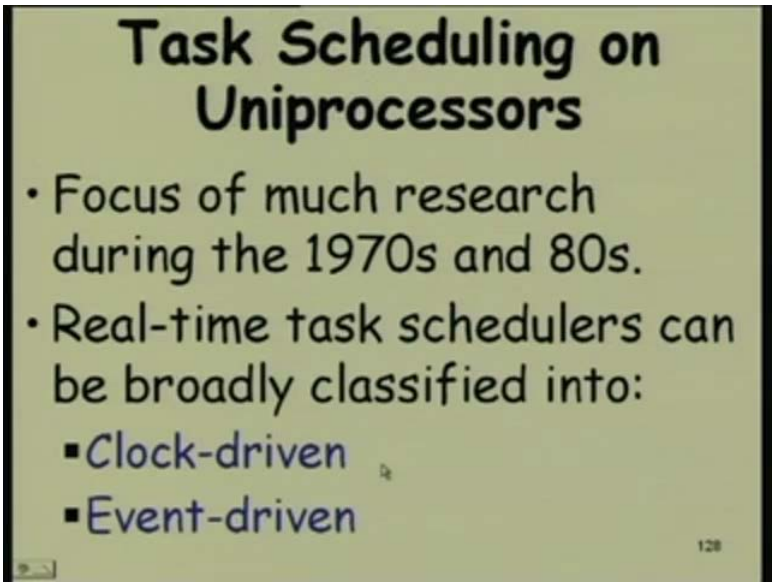
as and when required when a new task arrives or some task that is executing completes. So, that is the basic difference between a clock driven scheduler, and an event driven scheduler. These are two main categories of schedulers that we will be discussing.

(Refer Slide Time: 13:48)



As we are saying that lot of research has happened has been reported in real time task scheduling, and if we look at all that research reports that have been published mainly there are two categories of those scheduler. One is schedulers for unit processors, and the other is for multiprocessors, and distributed systems you can classify them into two categories.

(Refer Slide Time: 14:25)

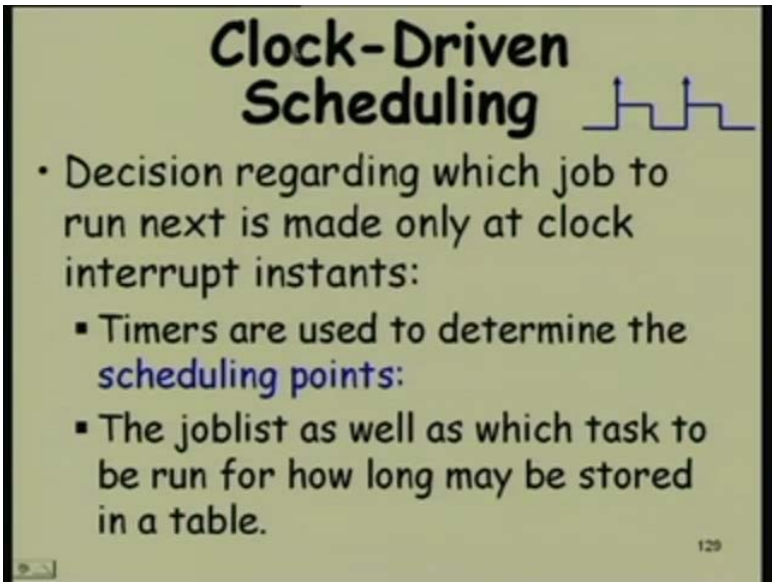


Now, let us try to look at task scheduling in unit processor, because this are much simpler problem .We will see that multiprocessors and distributed task scheduling is much more important; many use was there actually, here this is the simplest one. And if you look at the research results; most of the results occurred here during 1960s, 1970s and 1980s. And after that the intensity of research for unit processor scheduling has really reduced excepting for finding unit processor scheduler for special type of task situations.

But in general, you will find that new schedulers are not appearing for unit processor. Why is that, because optimal schedulers have been found in these cases, whereas multiprocessor and distributed we do not have an optimal scheduler right.

Now, let us see this. So, the schedulers for unit processors can be broadly classified into clock driven, and event driven as you were saying that two main categories of schedulers; the clock driven ones and the event driven ones. And in each of these 1000 of papers have been published many variants specially in the event driven 1000 of p h d degrees have been given 1000 or 10 of 1000 proposing new types of event driven schedulers with small difference in characteristics and so on. So, we will not really go into all that result, but we will go into the results which are enough to understand that this area well, unit processor real time task scheduling.

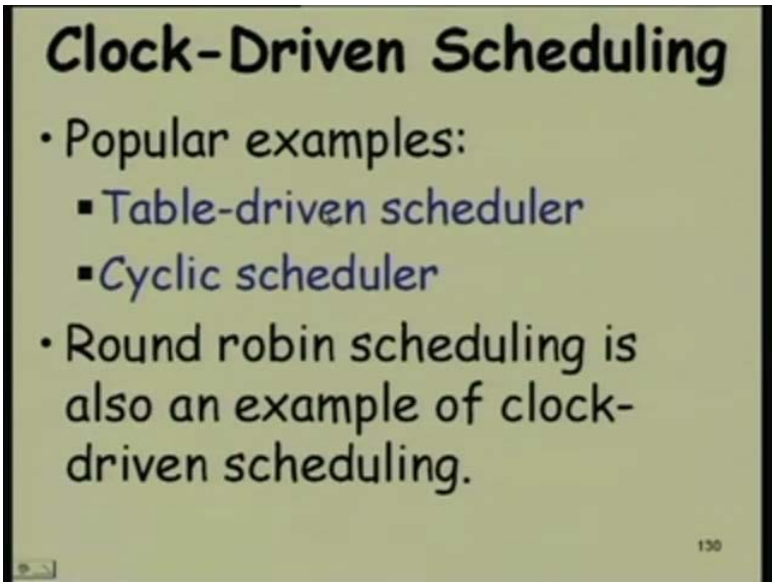
(Refer Slide Time: 16:24)



First, let us look at the clock driven schedulers, because these are the simplest ones, the event driven ones are more complex. So, here as you are saying that there is a timer which produces interrupts at regular intervals, and each time a interrupt occurs the scheduler wakes up and that is the scheduling point. And here normally, we will see that there are variations of this actually, but typically the schedule will be stored in a table right in a clock driven scheduler.

So far the scheduler nothing too much to do actually, as soon as the clock interrupt comes, it just checks which which has what is the status of the task that is executing it it does not even have to do that actually. It just looks at the table and finds out whether the current task needs to be stopped and the next next task needs to be started do that, it is a very simple thing. So, here they will have a table where we will store the schedule. So, the schedule is pre computed in a clock driven schedule. We will store it in a we will pre compute and store it in a table, and as the scheduler runs the clock driven scheduler regularly it will check the table and see what to do?

(Refer Slide Time: 17:53)



And there are two popular examples of clock driven schedulers. We had said that there are many types, but two which are sufficient for us to know I mean very popularly used most of the applications, you will see that these are the two were used one is a table driven scheduler, other we will call it as a cyclic scheduler. The one we discuss in the operating system course round robin scheduling time. Sliced round robin scheduling is also an example of this at your time slices, and at the clock interrupt according to clock interrupt the time slices occurs and after time slice. One task is next task is taken up, and that occurs in a round robin. So, that is an example of a clock driven scheduler.

So, we we will discuss these two the table driven scheduler, and the cyclic scheduler and these are also called as offline schedulers, because we would have to somehow develop the schedule the tasks, and store that in a table these are offline, because they are not decided the schedule is not dynamically decided as the system proceeds that is why this is also called as static scheduler. Offline scheduler and static scheduler 2 terminology will used to refer to these clock driven schedulers the concept is very simple, but used extensively in embedded applications, actually I visited some of the industries which manufacture embedded applications, and I just talk to them what kind of schedulers? You do. You use what kind of operating system and scheduler they said that see we use only clock driven scheduler, because our applications are simple do not need anything beyond that most efficient simple. You can manufacture devices cheaply which produces the desired result.

So, very important category, even though simple very important, it is likely that if you work in an embedded industry or where real time programming goes, and you you will mostly be using this. So, let us look at this the main advantage is that these are very efficient take very little space the scheduler component is nothing, actually we just consult something, and we just finds out whether to stop the current one and start the next one. But the table will take the instruction.

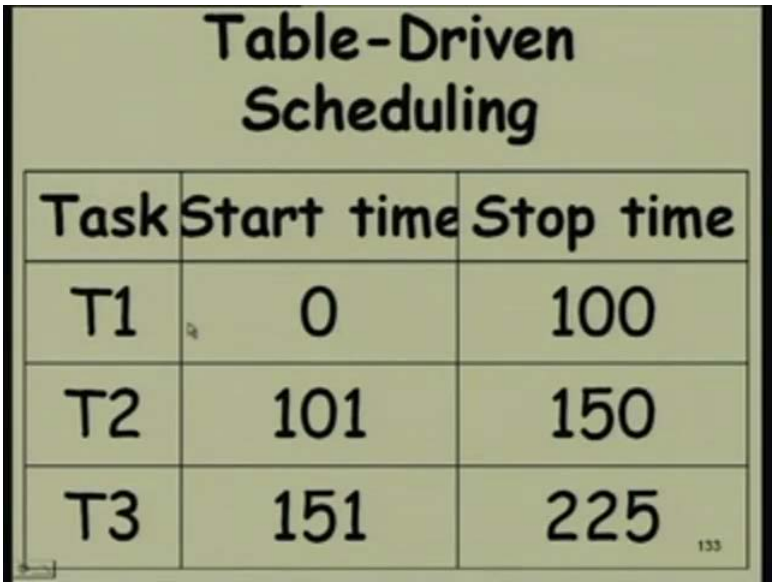
Ok

So, yes space is a problem here, because we need to store the schedule, whereas in the other schedulers the event driven ones we do not store the schedule, but we will use space there for other purposes. We do not store the schedule there, but we will any way use space to store the tasks that are pending all those right what have not completed. So, far, but here there is no such concept just keep the schedule keep on taking one out of that. So, very computationally efficient nothing much do for the scheduler checks up the schedule table, and just does as is there in the table, but the work is that we have to somehow reduce that table right.

Let us see and of course, minimizing the table is a important problem, because for embedded applications you cannot really store a table which is you know several megabytes take several megabytes, another I mean one big problem here. Why clock driven schedulers are not used in unless the application is really small is that cannot handle sporadic or a periodic tasks very, very difficult here.

Only the periodic tasks can be handled even for a cell phone. We were discussing the example of a cell phone there are some tasks which can be periodic, but most of the tasks there many tasks; for example, the user pressed a this thing send a sent a an SMS set up a call those all are a periodic, is it not?

(Refer Slide Time: 22:20)



So, table a table driven scheduling stores a table as we said, the table contains the schedule, so if you have three tasks. The task one will start at 0 time, and it will be stopped at 100; T 2 will start at 101continue at 150, T 3 starts at 150 one and stop at 225. Typically the scheduler what it will do is? Start the task 0 set a timer, and as the timer interrupt comes it will take up the next one right.

Even, if it is not completed in the time.

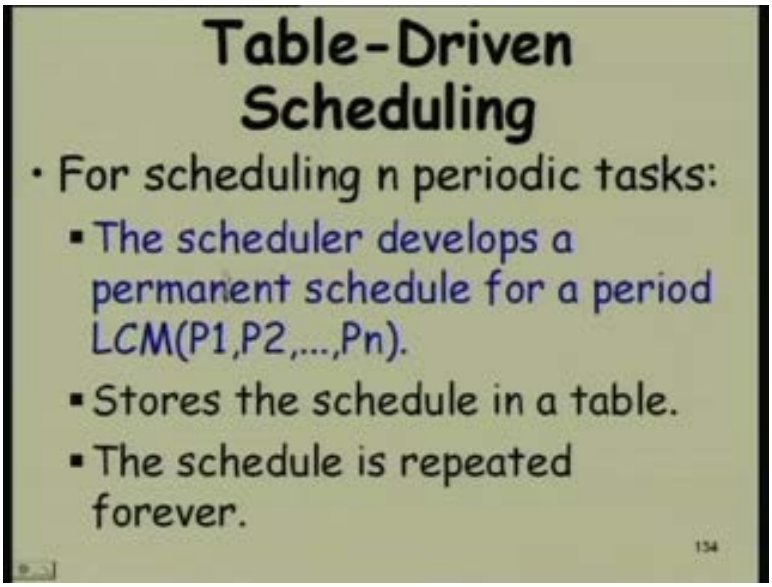
Yeah even if it is not completed yes that is the difficulty with this.

So, it some for some reason, it cannot complete by this it will be just stopped started the next one; that is all the scheduler just consults the table whatever is there in the table it just does that.

Let then for task T 1 then the so…

That in the offline we will have to do that. So, it is on to the designer it is nothing to do with the scheduler a scheduler will do whatever is stored in you have stored in the table. So, it is the programmer the designer who who is the one who will produce this schedule. Yeah we do not have to do that. Yeah we will see how to do that. So far each task for each time we will have to set up a timer.

(Refer Slide Time: 23:53)



And now, let us say if we have n tasks, then the scheduler develops a permanent schedule for P1, P2, P n LCM. Why is that see each task n periodic tasks, and let us say T1 T 2 T n. So, the tasks are T1, T2, T n; and their corresponding periods are P1, P2, P n or in other words task T1 starts at 0, and keeps occurring at P1 interval.

So, first one is at P 1 2 P 13 P 1, and sometime during this period it just starts executing and completes sometime, but before the end of the period. Right now there are other tasks like that P2 2 p 2 3 p 2 and p n.

Now, what we were saying is that what is the size of the table for which the schedule needs to be stored should we store up to p n will that be enough, what do you think? We think again here.

No see can we store the entire schedule starting from 0 up to P n. Let us say let us say this is P n, the n th task is 0 to p n. Let us say p n. Will this be enough, if we store 0 to p n, will this be enough.

No sir.

No why is that?

Sir, this is one of the we will get (( )) be completed.

Yeah yeah. So, let me just draw another diagram. Let us say let us consider two tasks p 1 and P 2. So, p 1 let us say occurs like this here. Let me draw like this this is P 1 2 P 1 3 P 2. Now, let us say sorry, 3 P 3 P 1. I am sorry P 1 2P 1 3 P 1 express from time 0. And let us say this is p 2 then 2 p 2. So, is it if we store the schedule up to p 2? So, when p 1 runs first instance p 1 second instance of p 1, and then when p 2 runs is that enough to store that.

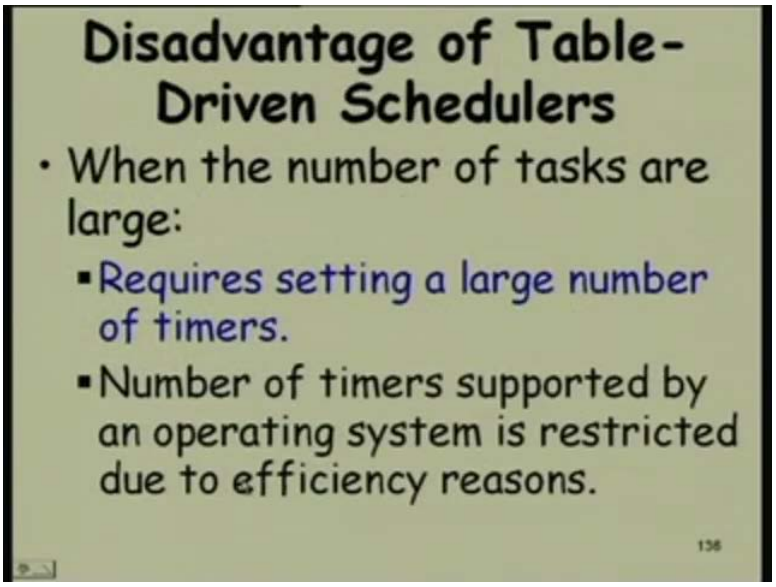
Not.

No obviously, because what do we do when the next instance of p 1 occurs, it is not repeating from here. So, we need to store the table the schedule table until all of them have 0 time of see here. It was 0 is 0, and the situation that occurred in 0 that same situation must occur at the point up to which we need to store the schedule. So, when will that situation occur in the LCM just think of it.

So, that is the least common multiple of all the periods at which the situation is as if all of them are starting again from 0 or in other word we must store the table up to a point of time such that that time divides. Sorry, all the periods divide this time, if we store up to T then T by p i should be an integral number, and the lowest that we can do is the L M lowest common multiple of all the periods. So, that is why you have written here that for a periodic scheduler for a sorry a table driven scheduler need to store the table size up to this time for the time LCM P 1 P 2 up to P n.

And then the schedule is taken up one by one timer is set and handled, and then it is repeated for ever after the table ends again start from the beginning. This is a very simple scheduler nothing much for the scheduler to do except in consulting the table, and as soon as the table ends starts from the 0 very efficient nothing much to do very little run time over. It can have just few instructions, but main problem is that it is inflexible very difficult to accommodate dynamic tasks.

(Refer Slide Time: 29:52)



Now, see one thing is that we will have to do a large number of timers settings here right, each time we set a timer right some times for 100 some times for 150 sometimes 70. So, this is a major problem here for this table driven scheduler each time. We need to set a timer now can we do better. So, let us look at that. So, we have this cyclic scheduler actually, if you go to an embedded industry find that this is the one the cyclic scheduler is the one, which is extensively used not the table driven one.

Because of the reason, we said that each time you know this occurs in millisecond and millisecond you set a timer just imagines the efficiency. So, right here you do not set any timer. Let us see in the cyclic scheduler no timer setting will be necessary extensively used almost any real time small real time development. You want to you do, and you use a cyclic scheduler large majority of the small embedded applications use these cyclic schedulers. So, let us look at it here unlike the table driven scheduler. We do not store it for period at the entire period of LCM of something, and then keep on repeating here, what we do is we also actually we store a schedule here also.

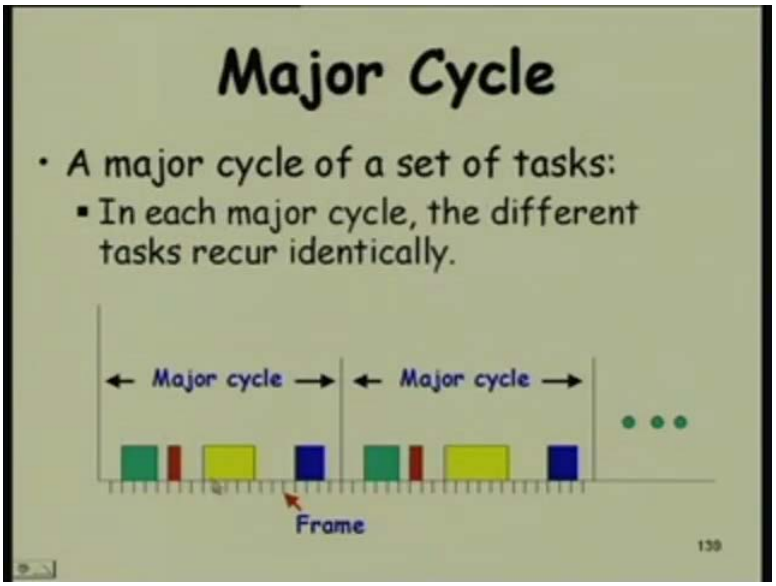
But the schedule, we divide it into two parts the major cycle, and the minor cycle - the major cycle is the one which repeats the schedule repeats in the major cycle right, we have some diagrams to show you, and each major cycle is divided into minor cycles, and the minor cycles, we will call them as frames.

(( )).

A k ax's are also known as yeah right a k a is also known as… So, these are equivalent term a minor cycle also known as a frame. So, a major cycle is there which the period for which the schedule is stored is… And then the major cycle is divided into minor cycles and a task can start either not either it only at the start of a minor cycle, it cannot start anywhere.

See in the table driven scheduler, the task can start in anywhere at 100, 150 anywhere, but here it will let us say task it will either either start at 10 or 20 or 30 or 40; if the frame size is 10, then it has to start a multiple of 10 right. So, there is a minor major cycle after which again, the schedule is repeated and each major cycle is divided into equal sized minor cycle called as frames. And we can the scheduler wakes up only at the start of a frame see here no setting of timer each time is necessary. We will have the scheduler waking up only after every frame it has not set any timer anytime is not it basic difference between a table driven and a cyclic scheduler.

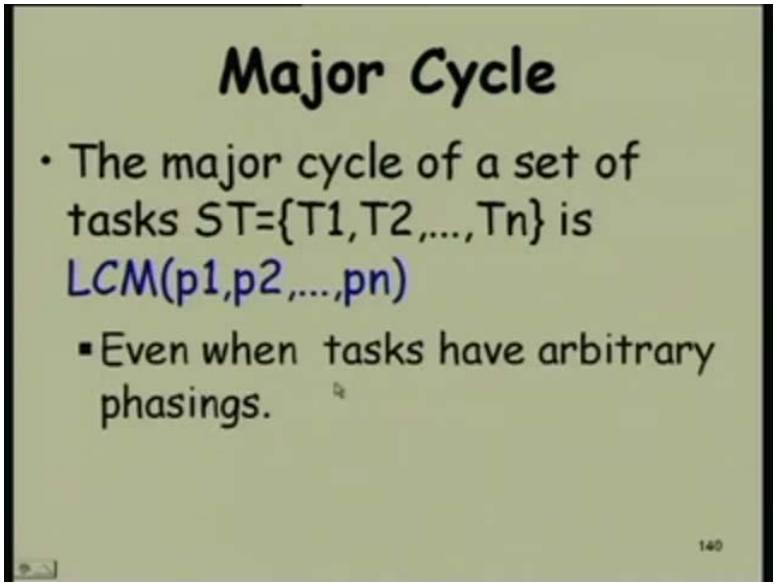
(Refer Slide Time: 33:40)



So, see this that the major cycle the tasks are repeated here, the schedule is repeated, see here this one was running here, this one is running here, then the red one, yellow one etcetera. So, the schedule is repeated in each major cycle, but the major cycle is divided into this minor cycles. And the tasks can start at a minor cycle or at the beginning of a

frame, but it can end before a frame. So, some time may be wasted nothing is running here, because the task started at a minor cycle or a frame, but could use only few frames and part of one frame, this much is wasted, but in a periodic sorry in a table driven scheduler as soon as this completes we could have started another one. So, that way possibly it uses the processor time less efficiently, but it has a major advantage no timer setting is necessary.

(Refer Slide Time: 34:52)



So, the major cycle again can be given by p1, p2, p n. We will see that we will also sometimes need to change this, we will we will see in situations under which and then this will occur, even when the tasks of arbitrary phasing right. We know what is a phasing. So, if p1, p2, p n are the periods of T1, T2, T n; then we had said that the schedule will repeat after this I mean normally we will see some situations, where it we need to store slightly larger if I not designed our scheduler properly, but I mean the cyclic scheduler, but this is the one we can do best we can reduce the major cycle below l c m p1 p2 p n is it not?

Sir, nearer to major cycle at least one time off for the major cycle.

No, no timer is set for major cycle.

How that is the system will come to know the one major cycle is over.

See it just keeps on checking the table right each frame at the beginning of a frame the timer interrupt comes, the scheduler awakes a and then it tries to see what is there in the table as soon as the table ends it takes up the next from the beginning. Sir beginning,

No timer is set.

This first only once it has to be set.

Yeah it.

Yeah I think that is a good point he says that in a cyclic scheduler, we need to set a periodic timer only once right is good point. That the cyclic scheduler a periodic timer. Let us say 10 millisecond or 15 millisecond as you will decide on the frame size a timer is set only once which de-markets the frame boundaries. Frame boundaries for the major cycle one. Major cycle, we do not bother major cycle, we will store the schedule up to the major cycle, but no timer setting is necessary for a major cycle. We only set one periodic timer as you said which gives interrupts only at the frame boundaries and we keep on tasks one by one from the table keep executing them it is frame and then as soon as the schedule ends. We start from the beginning.

Sir, does it mean that we will have to set the timer first each of the frames that are...

No, no that is the periodic timer see there is a there are we will as we discuss into more detail that in any operating system, there are two types of timer. One timer is a one set timer; a one set timer is where we just set it once for 100 milliseconds after 100 it just gives one a periodic timer is one, where we just set it once that every 50 or every 15 or 10 you just give us an interrupt just keeps on giving every 15. We do not set it each time set it only once keeps on giving interrupts is that.

So, in that way it is much more efficient only once setting is necessary there each time a task completes. We have to set and we are talking of milliseconds just imagine that every millisecond, if you need to set the timer. It is a big work right, if inefficiency and here even, when the tasks are arbitrary phasing. So, we had seen this simple case I think this is the one where we had seen here that when that we had consider the zero time, the task p1 has arrived p2 had arrived with respect to the zero time and. So, on they have kept on repeating here see here they have kept on repeating.

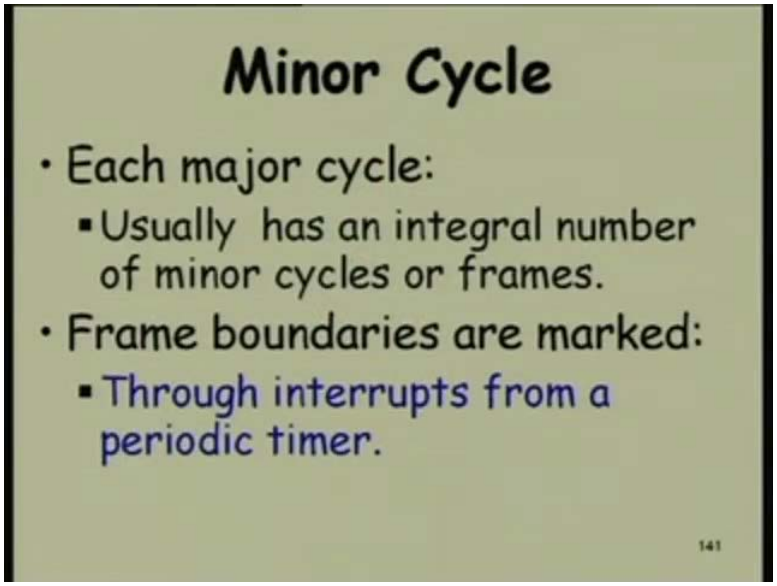
But what will be the situation, I mean what will be the period size or the size of the table when the tasks are arbitrary phasing; for example, one task up to sometime no event occurred, and then I mean up to some time the task did not arrive at all. And after that time, it just kept on recurring every 10 millisecond, and what if another task nothing was arrived until this time and then it kept on arriving here.

So, what will be the duration for which you need to store the table? Yes when the tasks have arbitrary phasing.

(( ))

Anybody would like to answer this question. So, I had set this question to one of your predecessor batch and then said you come up with a proof, and they had actually some of them had we we said that we will give a bonus mark here and you please try this.

(Refer Slide Time: 40:16)



And then they had come with this I I think the result is there later. So, they have a proof nice proof, let us show you and it remains the LCM p1, p2, p n. Let us see that. So, before that lets the minor cycle as we said that typically the major cycle contains an integral number of minor cycles.

The minor cycle keeps on repeating. Now it is a periodic timer basically, but it is not compulsory that a major cycle will have an integral number of minor cycles. When you

divide that, but we will see that unless, it divides the table size really becomes large, because you cannot repeat the table I mean at the end until the major, and minor cycle are at the same time completed at the same time. You cannot repeat the table.

So, the table size becomes very large. It is more than LCM p1, p2, p n. So, this is the condition I was referring to that we sometimes have to store the table size more than LCM p1, p2, p n. And that is the situation, if you are designed a cyclic scheduler where the frame does not divide integrally does not perfectly divide the major cycle, then we will have to store for a the schedule for a much larger time than LCM p 1, p 2, p n.

So, the frame boundaries are marked through the interrupts of a periodic timer now each task can run in one or more frames. It cannot run on a fraction I mean it cannot use fraction of a frame, and give another frame to some other task. No, it will will assign one complete frame or more than one frame to a task cannot give a fraction of a frame to a task one or more frames are assigned to a task.

And we will see that, if you are talking of cyclic scheduler the most important problem, before the designer of a real time program is how to find the frame size the most important question that he has to answer, because this is the this will become the minor cycle.

And at this point the tasks will get scheduled; if it is too small then it will be very bad totally inefficient, if it is too big then also it is bad. We have to find out the exact frame size for which it will work, if it is anything other than that there will be a problem right it will be a bad scheduler. So, we will just stop at this point.

(Refer Slide Time: 43:10)

And we will continue from the next class that is the important problem of how to select an appropriate frame size here, we will take some examples problems here try to do that and there are very important concepts here in designing the actual frame size. And we will also, let you do some problems here, we will we will stop here.