

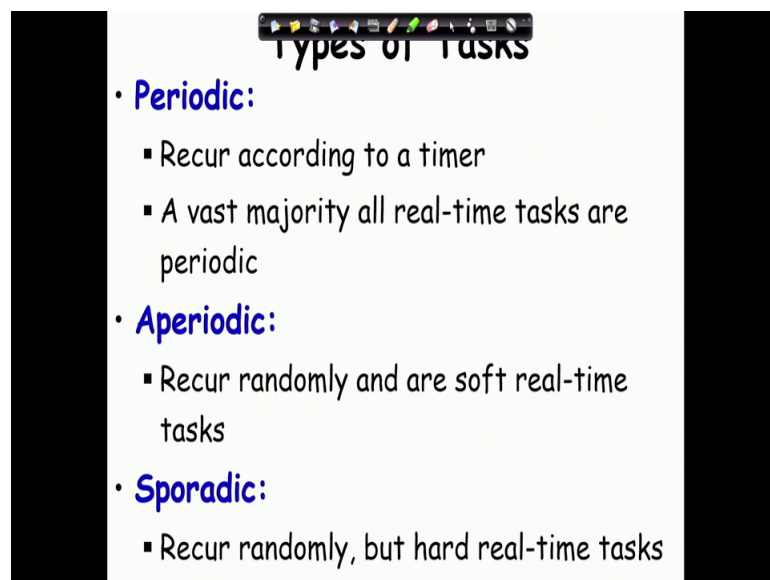
Real Time Operating System
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 02
Basics of Task Scheduling

Welcome back. So, in the last lecture we had a very basic introduction to real time systems, what is a real time and what is a real time operating system, how is it is different from a traditional operating system and we had said that one of the major requirements for a real time operating system is that it should help the tasks meet their deadlines.

And then we had seen that embedded applications are becoming numerous and that is a important application area for the real time operating system and then we were looking at what are the different types of real time systems hard, soft, firm real time systems what are the examples of that. And now let us continue from there and let us see what are the different types of tasks, that a real time system might have to handle.

(Refer Slide Time: 01:17)



Types of Tasks

- **Periodic:**
 - Recur according to a timer
 - A vast majority all real-time tasks are periodic
- **Aperiodic:**
 - Recur randomly and are soft real-time tasks
- **Sporadic:**
 - Recur randomly, but hard real-time tasks

So, real time operating system the tasks can be broadly saying three types periodic, aperiodic and sporadic. A periodic task as the name says the tasks recur according to a timer for example, let us say a chemical plant the parameters of the plant are sensed periodically and then the computer decides whether to take a corrective action and if here

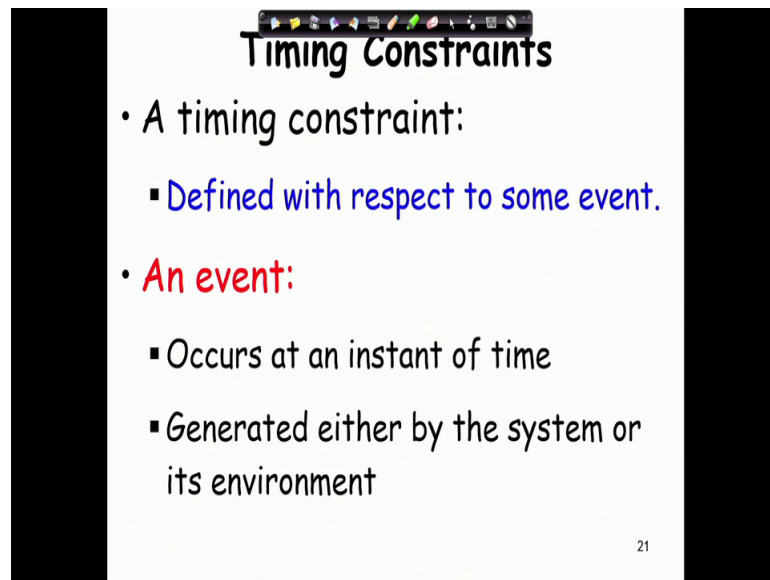
is a corrective action like changing the temperature or chemical concentration so on it does within certain time.

So, we will see that for all real time applications the periodic tasks are majority, a large number of tasks are actually periodic tasks. We might have aperiodic tasks, the aperiodic tasks they recur randomly and are soft real time tasks. For example, let us say an operator gives a command let us say to increase the temperature or to reduce the rate of chemical reaction. So, when the operator will give a command is not known. It can be anytime t can be a random and also the response to this command is basically soft real time once the user gives the command it may take several seconds and there is no hard deadline by which you need to do it.

On the other hand there may be sporadic tasks which are again random, but these have hard deadlines with them for example, let us say a fire condition occurs in a industry in a chemical plant then the sensors check the condition and they initiate several actions may be sounding a alarm, may be stopping the reaction may be starting a water shower and so on. So, when such alarm will occur a fire condition is unpredictable, it is random. But then the action must be initiated within certain time for the plant to contain such emergency situation.

So, as we were saying let me just recapitulate. We were saying that in any real time application a vast majority of the tasks are periodic some tasks may be aperiodic and these are basically soft real time tasks these are called randomly and another category are the sporadic tasks these are real time tasks, but they occur randomly and these are rare aperiodic and sporadic tasks number is very less compared to the periodic tasks.

(Refer Slide Time: 04:53)



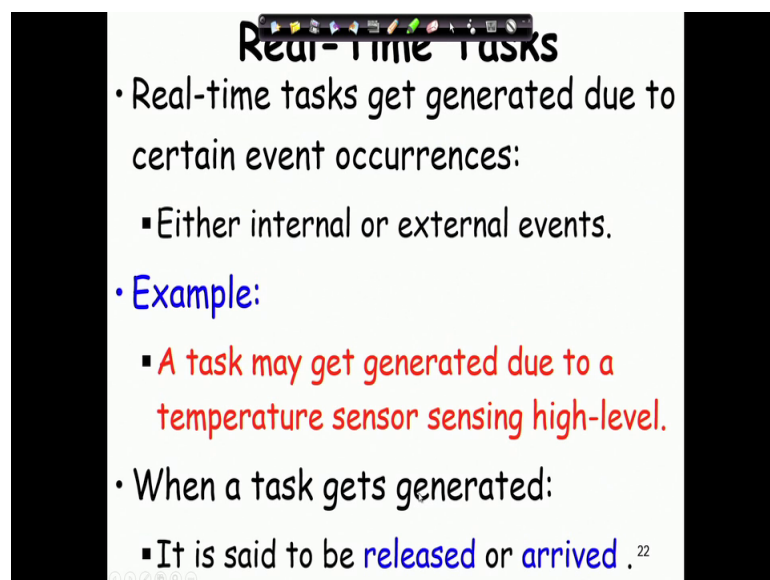
Timing Constraints

- A timing constraint:
 - Defined with respect to some event.
- **An event:**
 - Occurs at an instant of time
 - Generated either by the system or its environment

21

Now, another thing that we want to mention is that the timing constraint on a task is defined with respect to some event. So, once an event occurs then we start counting the time and we say that before certain time the result must be produced and the event may be either produced by the system or by the environment we will see examples of both as we proceed.

(Refer Slide Time: 05:25)



Real-time Tasks

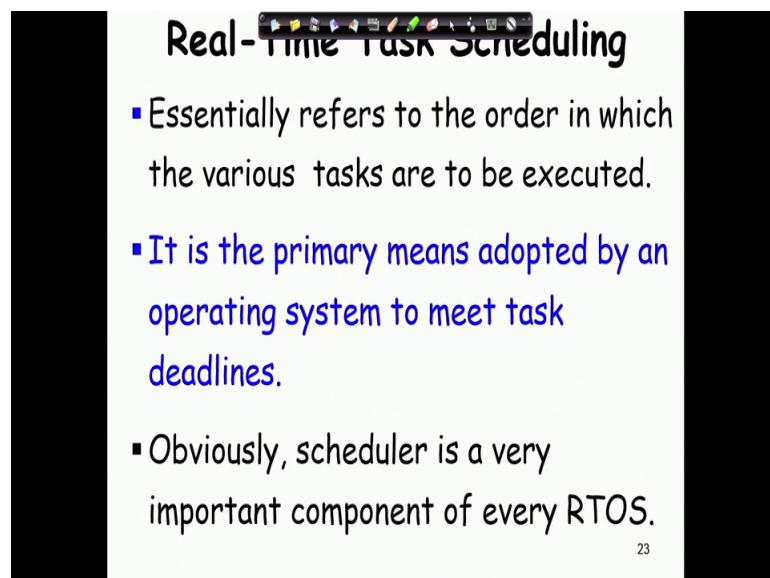
- Real-time tasks get generated due to certain event occurrences:
 - Either internal or external events.
- **Example:**
 - A task may get generated due to a temperature sensor sensing high-level.
- When a task gets generated:
 - It is said to be released or arrived .²²

In a typical application there are large number of real time tasks and these get generated due to event occurrences some of these event may be internal events or may be external

events. Just to give an example that a task may be generated when the temperature sensor senses a high temperature then the task would be to reduce the temperature may be to start a water shower and so on.

So, this event is got generated by an external event of temperature increasing to certain extent. The internal event may be that may be the memory out of memory or may be some task is taking too much. So, those are the internal events. So, where never a task gets generated due to an event we say that task is released or is generated or even say that the task has arrived.

(Refer Slide Time: 06:44)



Real-time Task Scheduling

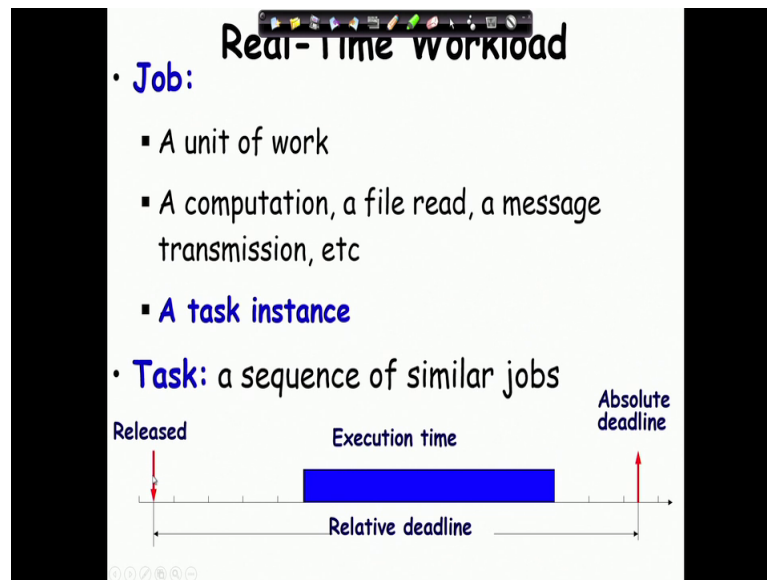
- Essentially refers to the order in which the various tasks are to be executed.
- It is the primary means adopted by an operating system to meet task deadlines.
- Obviously, scheduler is a very important component of every RTOS.

23

Now, with this background let us look at the real time tasks scheduling. The most elementary description will say that the tasks scheduler decides on the order in which the different tasks to be executed by the computer which one to be executed first, which one to be executed next and so on. So, it is a tasks scheduler whose role is to decide which task is to be executed next. And we already said that the tasks scheduler are important part of all real time operating systems and they are the primary means by which the operating system helps the applications to meet their deadlines.

As we look at different real time overrating systems the first thing we will mention is that what is the type of scheduler. So, every real time operating system the scheduler is a very important component.

(Refer Slide Time: 08:00)



Now, let us look at some basic terminology because we will be using this terminology as we proceed. A job is a unit of work for example, perform a computation like check if certain condition is reached or certain condition is satisfied to read a file, to store some data, to communicate with another tasks etcetera these are examples of jobs. And these are task instances.

Whereas, the task is a sequence of similar jobs for example, let us say one task may be to handle the temperature of the chemical plant exceeding some threshold value. So, each time the temperature sensor measures it and it periodically measures a task is generated to check the temperature value whether it is above the threshold and then if it is above the threshold then take some corrective action. So, every few milliseconds or so, when the temperature sensor gives its input through an interrupt then the task gets generated.

And when the task gets generated we say that the task is released. So, that is shown here that the task release time and then the operating system starts to execute the task after some time. So, this is the start up executing this the release time and this is the start up execution and it may complete after some time and then a deadline is associated with real time tasks and if the deadline is measured and reported with respect to time zero we say that it is a absolute dead deadline. Whereas, if the deadline is computed or is reported with respect to the release time then we call it as a relative deadline that is what is shown here the relative deadline is starting from the start release task release to the deadline

whereas, the absolute deadline is starting from task time zero in this area that is a accepted terminology that an instance of a task is called a job.

(Refer Slide Time: 10:53)

Task Instance (Job)

- A task typically recurs a large number of times:
 - Each time triggered by an event
 - Each time a task recurs, an instance of the task is said to have been generated or released.
- The i th time a task T recurs:
 - Job or Task instance T_i is said to have arrived.

25

A task is typically recurs a number of times, periodic tasks occur they recur periodically based on some clock interrupt whereas, aperiodic and sporadic tasks they occur randomly and each time a task recurs we say that an instance of the task or a job has been generated or released. And when the i th time the task t recurs we say that the job or task instance T_i said to have arrived.

(Refer Slide Time: 11:46)

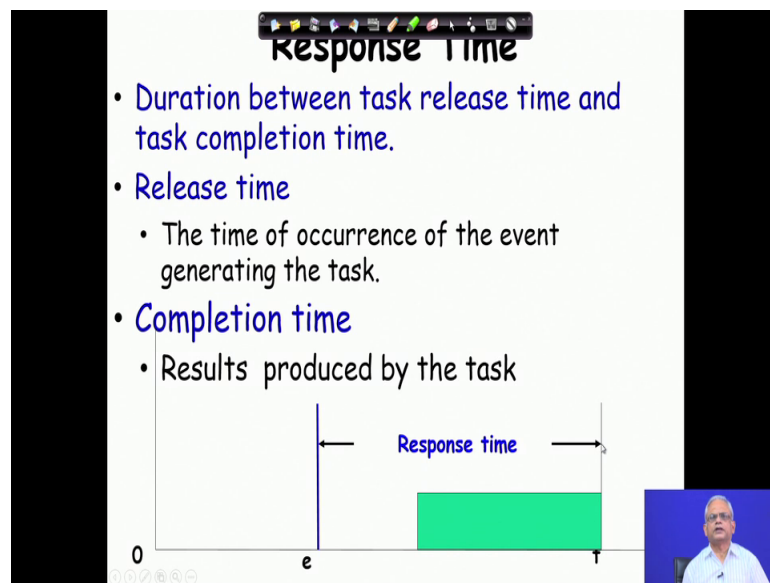
Relative and Absolute Deadlines

- **Absolute deadline:**
 - Counted from time 0.
- **Relative deadline:**
 - Counted from time of occurrence of task.

2

And we already explained the absolute deadline is a terminology we use to give absolute time for the deadline starting from time zero whereas, relative deadline is counted from the release of the task. So, this already shown earlier. So, here because this is important terminology the relative deadline and the absolute deadline I am just showing it again from the task release to the deadline is the relative time whereas, from time zero to the deadline we measure, we indicate it by the absolute deadline.

(Refer Slide Time: 12:24)



Now, let us define another important terminology that we will be using is the response time. The response time is the time from the release of the task till the task completion time. So, as the task is released at time T the task execution starts after sometime and then it executes and completes at certain time. So, the time from release of the task to the completion time of the task we call it as the response time.

(Refer Slide Time: 13:14)

Response Time

- For soft real-time tasks:
 - The response time needs to be minimized.
- For hard real-time tasks:
 - As long as the task completes within its deadline,
 - No advantage of completing it any early.

28

And we had said that in the traditional overrating system which handles basically soft real time tasks the response time needs to be minimized. And even in the real time application in the real time operating system if the task is a soft real time task we need the operating system to minimize the response time of the task whereas, for the hard real time and sporadic task minimization of the response time is not the objective, the objective is to meet the deadlines of the hard real time and sporadic tasks. For hard real time tasks as long as the task completes within its deadline there is no special advantage in completing it any earlier than the deadline.

(Refer Slide Time: 14:15)

Phase of a Periodic Task

- Phase for a periodic task:
 - The time from 0 till the occurrence of the first instance of the task.
 - Denoted by Φ .

29

Now, let us look at another term phase of a task because this also an important terminology as we look at the task scheduler we will often refer to this phase of a task. In any real time application there are many tasks and a large majority of that are periodic tasks, but one thing to note is that the periodic task do not really start from time zero many periodic tasks do not start the task they start after elapse of certain time. So, the events keep on occurring here at certain time and then they start, but interlay the task for some time does not exists. So, this is called as the phase of the task.

Just to give an example to make this point clear we will take the example of a rocket.

(Refer Slide Time: 15:13)

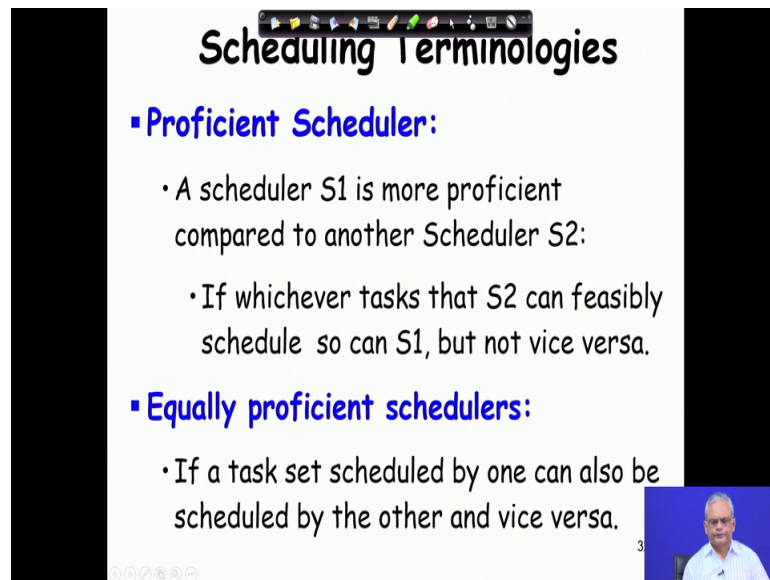
The slide is titled "Phase Example" and features a timeline diagram at the top right. The timeline starts at 0 and shows a series of vertical bars representing task instances. The first instance starts at 2000 mSec and is followed by several more instances, each separated by 50 mSec. A red arrow points to the first instance. Below the diagram, there are three bullet points:

- The track correction task starts 2000 mSecs after the launch of the rocket:
 - Periodically recurs every 50 milli Seconds then on.
 - Each instance of the task requires a processing time of 8 mSecs and its relative deadline is 50 mSecs.

At the bottom right of the slide, there is a small video inset showing a man speaking. The number "3" is visible in the bottom right corner of the slide.

So, once the rocket is fired initially it goes on its own momentum for certain time and then we do not have a task correction taking place. But after some time the task the trajectory correction tasks starts let us say that the track correction tasks starts after 2000 milliseconds of the launch of the rocket. So, the phase of this task the task correction task is 2000 milliseconds. And then once the tasks starts the first instance of the task starts after 2000 millisecond and then it periodically recurs every 50 millisecond and the execution time may be 8 millisecond and deadline is 50 millisecond.

(Refer Slide Time: 16:26)



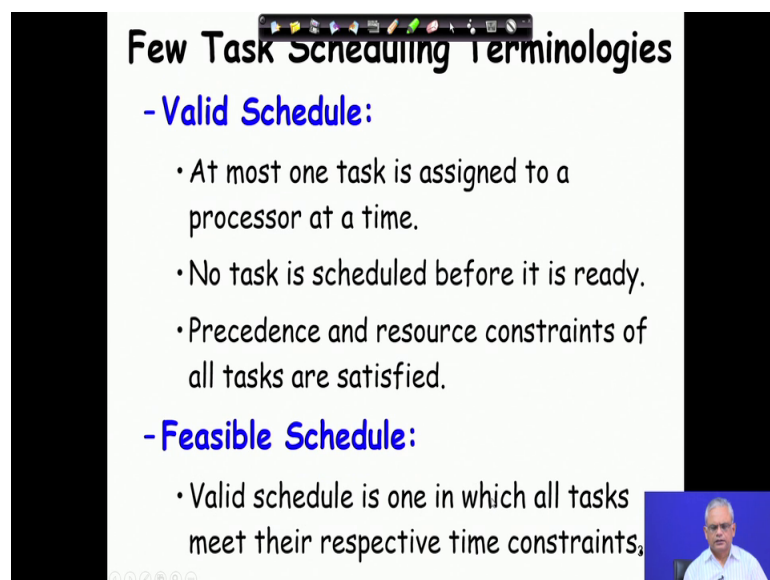
The slide is titled "Scheduling Terminologies" and contains two main sections. The first section is "Proficient Scheduler:" which includes two bullet points: "A scheduler S1 is more proficient compared to another Scheduler S2:" and "If whichever tasks that S2 can feasibly schedule so can S1, but not vice versa." The second section is "Equally proficient schedulers:" which includes one bullet point: "If a task set scheduled by one can also be scheduled by the other and vice versa." There is a small video inset in the bottom right corner showing a man speaking. The slide number "3" is visible in the bottom right corner.

Scheduling Terminologies

- **Proficient Scheduler:**
 - A scheduler S1 is more proficient compared to another Scheduler S2:
 - If whichever tasks that S2 can feasibly schedule so can S1, but not vice versa.
- **Equally proficient schedulers:**
 - If a task set scheduled by one can also be scheduled by the other and vice versa.

So, that is the description of this track correction task that the phase is 2000 millisecond after the rocket is fired, for 2000 milliseconds no track correction takes place and after 2000 millisecond the first track correction task starts and then it recurs after 50 millisecond and each track correction task requires 8 millisecond of execution time. And the deadline for each task once it is released is also 50 millisecond.

(Refer Slide Time: 17:07)



The slide is titled "Few Task Scheduling Terminologies" and contains two main sections. The first section is "- Valid Schedule:" which includes three bullet points: "At most one task is assigned to a processor at a time.", "No task is scheduled before it is ready.", and "Precedence and resource constraints of all tasks are satisfied." The second section is "- Feasible Schedule:" which includes one bullet point: "Valid schedule is one in which all tasks meet their respective time constraints." There is a small video inset in the bottom right corner showing a man speaking. The slide number "3" is visible in the bottom right corner.

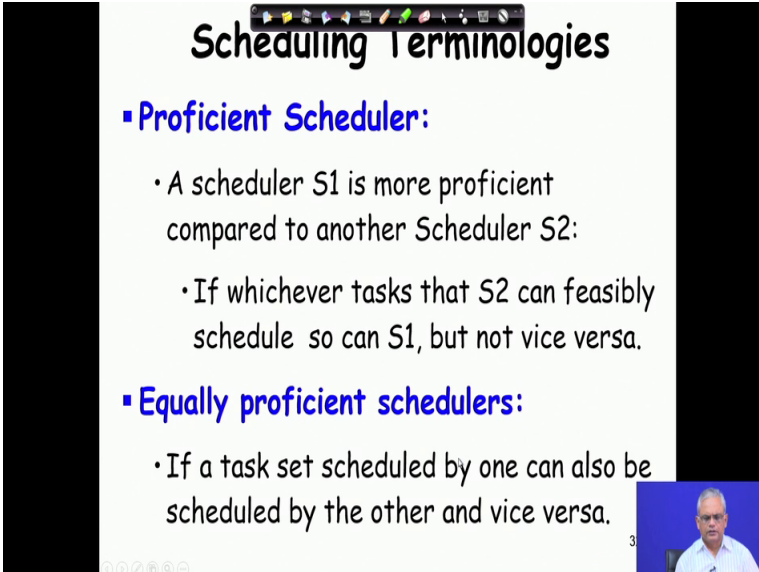
Few Task Scheduling Terminologies

- **Valid Schedule:**
 - At most one task is assigned to a processor at a time.
 - No task is scheduled before it is ready.
 - Precedence and resource constraints of all tasks are satisfied.
- **Feasible Schedule:**
 - Valid schedule is one in which all tasks meet their respective time constraints.

We look at another terminology which we will also use is the valid schedule and the feasible schedule.

A valid schedule is one where the task is assigned to a processor the task is not scheduled after it is released. So, only after it is released or generated then the scheduler schedules it and its constraints that it should proceed some other task etcetera is satisfied, but then it does not say that whether the deadline will be met. In addition to these aspects if the deadline is also met then we call it as a feasible scheduler. So, for a valid schedule if the task deadlines are also ensured to be met then we call the schedule prepared by the scheduler as a feasible scheduler.

(Refer Slide Time: 18:07)



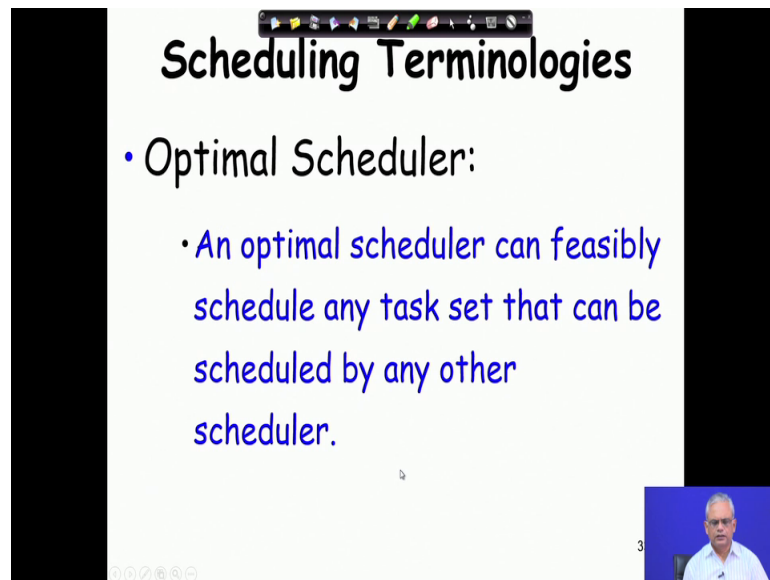
Scheduling Terminologies

- **Proficient Scheduler:**
 - A scheduler S_1 is more proficient compared to another Scheduler S_2 :
 - If whichever tasks that S_2 can feasibly schedule so can S_1 , but not vice versa.
- **Equally proficient schedulers:**
 - If a task set scheduled by one can also be scheduled by the other and vice versa.

3

We have another term as a proficient scheduler. We will see that given a task set for a application some scheduler cannot really ensure that the tasks meet deadline whereas, there may be another scheduler which let us these tasks set to meet the respective deadlines then we say that the second scheduler is more proficient. We say that a scheduler S_1 is more proficient than S_2 if whenever S_2 can feasibly schedule a task set, so can S_1 , but not vice versa. And two scheduler we say equally proficient, if for any task set that one scheduler can feasibly schedule, the other scheduler can also feasibly schedule.

(Refer Slide Time: 19:04)



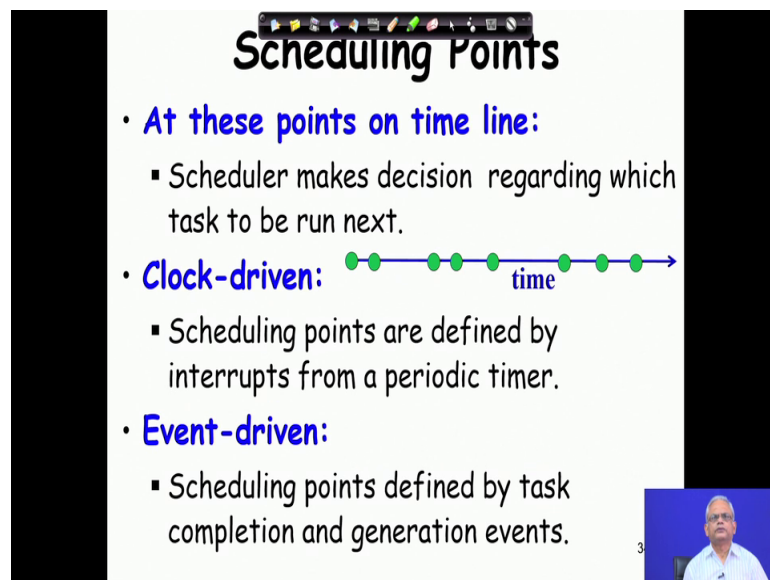
Scheduling Terminologies

- **Optimal Scheduler:**
 - An optimal scheduler can feasibly schedule any task set that can be scheduled by any other scheduler.


3

And an optimal scheduler is one which can feasibly schedule a task set which any other scheduler can schedule.

(Refer Slide Time: 19:18)



Scheduling Points

- **At these points on time line:**
 - Scheduler makes decision regarding which task to be run next.
- **Clock-driven:** 
 - Scheduling points are defined by interrupts from a periodic timer.
- **Event-driven:**
 - Scheduling points defined by task completion and generation events.

3

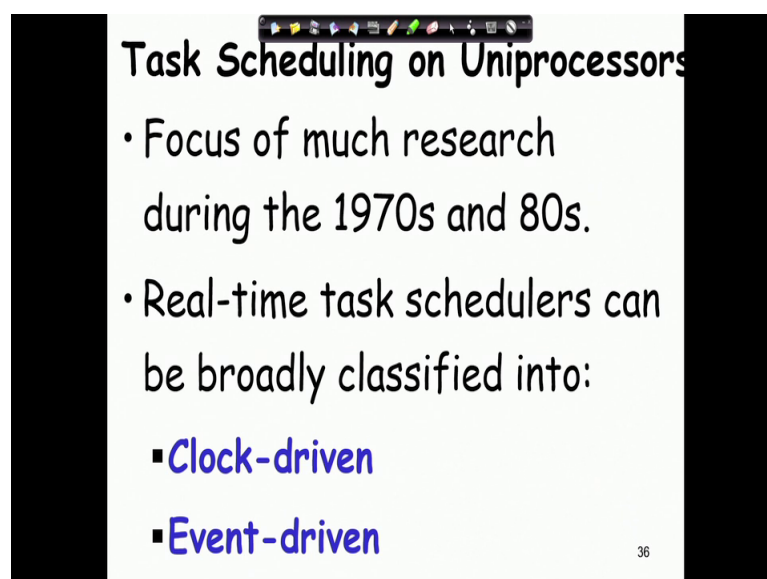
Now, we look at one more important point before we look at the standard schedulers that is about the scheduling point. We can think of that a scheduler is a software component which becomes active at certain points of time and then decides which task to run next. So, on the points of time at certain points of time the scheduler task becomes active and

then decides which task to run next and this points where a decision is made regarding which task to run next is called as the scheduling points.

Of course, in a clock driven scheduler the scheduling points are defined by interrupts from a periodic timer. So, they will be policed uniformly on the timeline and they are generated by clock interrupts and the task the scheduler becomes active as soon as the clock interrupt occurs and decides which task to run. On the other hand in event driven scheduler the scheduler does not become active based on a clock interrupt, but it is based certain events as the name says event driven the events that are that cause these schedulers become active are not generated by the clock, but by the completion and generation of tasks.

So, if some task gets generated the scheduler becomes active to check whether this new task which has come up is it to be scheduled right now and also whenever a task completes the scheduler becomes active and then decides which task to schedule. We will see that the clock driven schedulers are simple schedulers whereas, the event driven schedulers are more complex sophisticated schedulers and they have any advantages over the clock driven scheduler, but the clock driven schedulers are the ones which are used in simple chip applications whereas, the sophisticated real time applications the event driven schedulers are used.

(Refer Slide Time: 22:06)



Task Scheduling on Uniprocessors

- Focus of much research during the 1970s and 80s.
- Real-time task schedulers can be broadly classified into:
 - **Clock-driven**
 - **Event-driven**

36

Now, let us look at the different schedulers that are valuable now that we have basic knowledge on the real time systems, the task characteristics, terminologies of tasks types of tasks and so on. Let us look at first tasks scheduling on uniprocessors. Actually tasks scheduling algorithms were the focus of the research in the 1970s and 80s. Lot of results became available during those days and many of those results are even referred till now so.

So, based on research done on those days we have mainly two types of schedulers one is called as clock driven and event driven. So, if you remember what we said in the earlier slide is that if the scheduling points are generated by a clock interrupt these are called as clock driven scheduler. So, here based on a timer the scheduler becomes active periodically and that is a clock driven scheduler these are simple schedulers. Whereas, in a event driven scheduler the scheduling points are defined by certain events and what are those events we said that there are two main events the task starting event or releasing event and the completion events.

(Refer Slide Time: 24:11)

• Endless Loop	• No Tasks, Polled
• Simple Cyclic Executive	• Single frequency
• Multi-rate Cyclic Executive	• Multiple frequencies
• Priority-based Preemptive Scheduler	• Interrupt driven
•	

So, we will look at various types of schedulers one is that endless loop it is just a simple program without an operating system there are no tasks and it just checks certain conditions in a loop and then acts, and starting with no operating system we have simple cyclic executors. There is a single frequency at which the tasks execute, the period of the

tasks the same and once a timer interrupt occurs the tasks are taken up for execution and then the system waits.

On a multi rate cyclic executive as the name says that there are multiple frequencies different tasks arise with different periods and we will discuss about the multi rate cyclic executive and how does it handle these different types of tasks and help to meet their deadline. So, these do the simple cyclic executive and the multi rate cyclic executive these are used in rather simple applications whereas, the ones that are sophisticated are the priority based preemptive schedulers these are the event driven schedulers. We will discuss our good portion of our discussion is on the priority based preemptive schedulers, but the next lecture we will discuss about the cyclic schedulers.

Thank you.