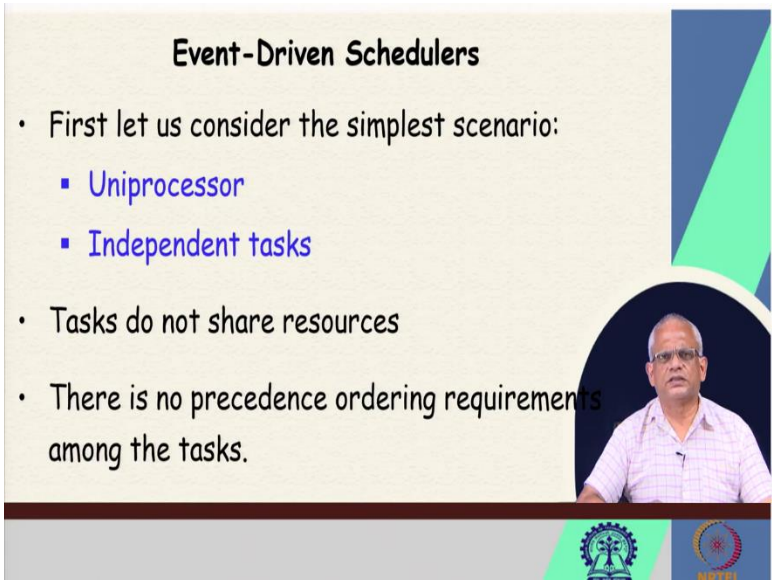
Real Time Systems
Professor. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Lecture 15
EDF Scheduler

Welcome to this lecture. So, far we have been looking at real time schedulers for last couple of lectures and we had so far looked at the clock driven schedulers. Which are the simplest schedulers used in simple embedded devices and in spatial, we had looked at the cyclic schedulers and then we had remarked that when the number of tasks increases and also there is need to accommodate asynchronous and sporadic tasks.

We need more powerful schedulers which are basically the event driven schedulers and we had looked at some very basic aspects of event driven schedulers like their characteristics, pre-emptive event driven and what are the events on which the scheduling, that is the scheduling points and so on. Now, let us proceed from there on and we will first look at the dynamic priority real time scheduling the EDF. So, let us proceed.

(Refer Slide Time: 01:25)



Event-Driven Schedulers

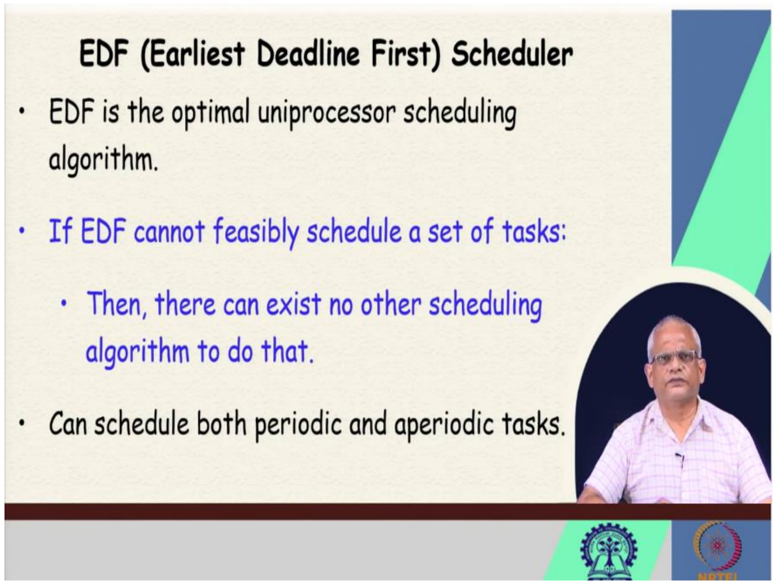
- First let us consider the simplest scenario:
  - Uniprocessor
  - Independent tasks
- Tasks do not share resources
- There is no precedence ordering requirements among the tasks.

The event driven schedulers is the focus of today's lecture and we will first discuss the simplest situation and see how scheduling occurs there and then we will relax these restrictions later on. One of the important restrictions is that we are looking uniprocessors. Even though in many embedded devices and desktops and laptops and so on multi-core processors have become prevalent.

But, to keep issue simple we will discuss about uniprocessors and now, even many embedded systems we have uniprocessor based systems and we will assume that all tasks are independent. In many practical situations some tasks may start only after some other task has completed and so on. So, those issues we are not considering later we will relax this constraint. But, right now the main constraint to simplify our problem is that we are considering uniprocessors and independent tasks.

And not only that, we also assume today's lecture, the tasks do not have any resource sharing requirements. They just complete, they just compute and proceed independently. Which is normally not the situation, many times tasks need to share results, completed results and partially completed results and so on, which we will bring in later. Right now for our simple discussion and event driven schedulers we will assume that there is no resource sharing requirement and also no precedence ordering.

(Refer Slide Time: 03:32)



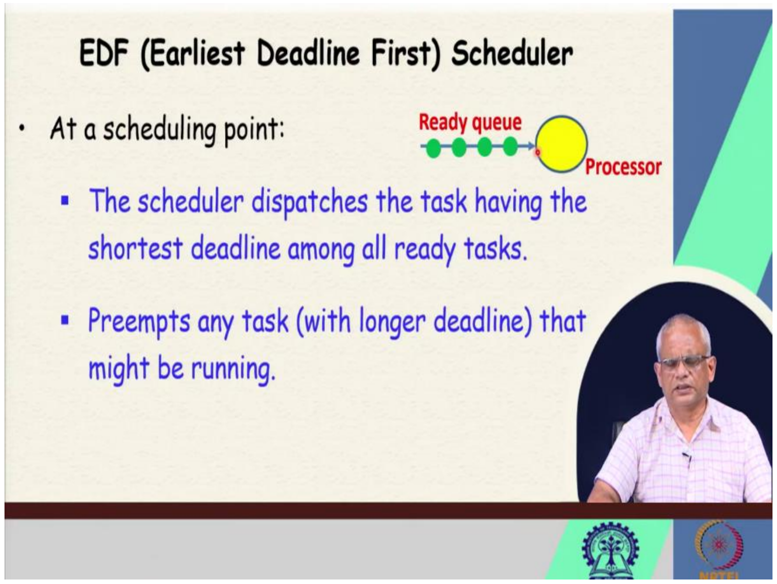
EDF (Earliest Deadline First) Scheduler

- EDF is the optimal uniprocessor scheduling algorithm.
- If EDF cannot feasibly schedule a set of tasks:
  - Then, there can exist no other scheduling algorithm to do that.
- Can schedule both periodic and aperiodic tasks.

The first scheduler that we discuss is the earliest deadline first scheduler whom you had said is a dynamic priority scheduler and it is the optimal uniprocessor scheduling algorithm. That means that if some tasks that cannot be scheduled using EDF. Then there is no other scheduler that are there or that can be devised which can schedule that task set. Including the static schedulers, none of the schedulers can schedule a task set even if it is unschedulable in EDF.

So, if EDF cannot feasibly schedule a set of tasks, then there can exist no other scheduling algorithm that can do it including the static and dynamic schedulers. So, if something is not schedulable in EDF, we can say that see on this processor, this task set cannot be run we need to modify the task set or modify the execution environment and one important characteristic of EDF is that both periodic and aperiodic tasks can be handled.

(Refer Slide Time: 04:59)



EDF (Earliest Deadline First) Scheduler

- At a scheduling point:
  - The scheduler dispatches the task having the shortest deadline among all ready tasks.
  - Preempts any task (with longer deadline) that might be running.

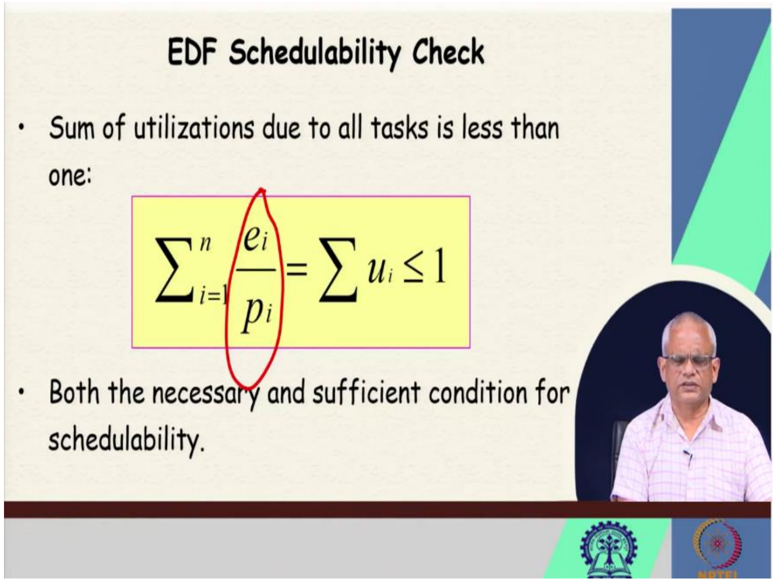
So, here we have a ready queue. The tasks which have become ready, they are in this queue as you can see here, as you can see here we have this ready queue and as the task come they are put in this ready queue and this is the processor and from this ready queue, the tasks are selected by the EDF algorithm to run the task which needs to run next on the processor.

And the scheduler, the EDF scheduler becomes active at every scheduling point and the scheduling points are when a new task arrives it looks at the queue and finds which one to run or when a task that is running currently on the processor it completes, finds out which tasks need to run. So, these are two events on which the EDF scheduler becomes active.

So, at a scheduling point, the scheduler dispatches the task having the shortest deadline among the ready tasks. So, to think simply the way the scheduler will work is that the tasks are there in a ready queue and at every scheduling point it will find out the deadline of all the tasks that are there, in the queue and then find the task which has the shortest deadline and then it will select that task to run.

But of course, this is not the way that the scheduler is implemented, because if we do that we keep them in one queue and scan it takes order n time o(n) time and which makes the scheduler very inefficient. We will see how to have an efficient implementation of the EDF. But, this is the main idea and also as task is selected when a task arrives, if that is selected to run, then the current task that is running need to be preempted.

(Refer Slide Time: 07:51)



EDF Schedulability Check

- Sum of utilizations due to all tasks is less than one:

$$\sum_{i=1}^{n} \frac{e_i}{p_i} = \sum u_i \leq 1$$

- Both the necessary and sufficient condition for schedulability.

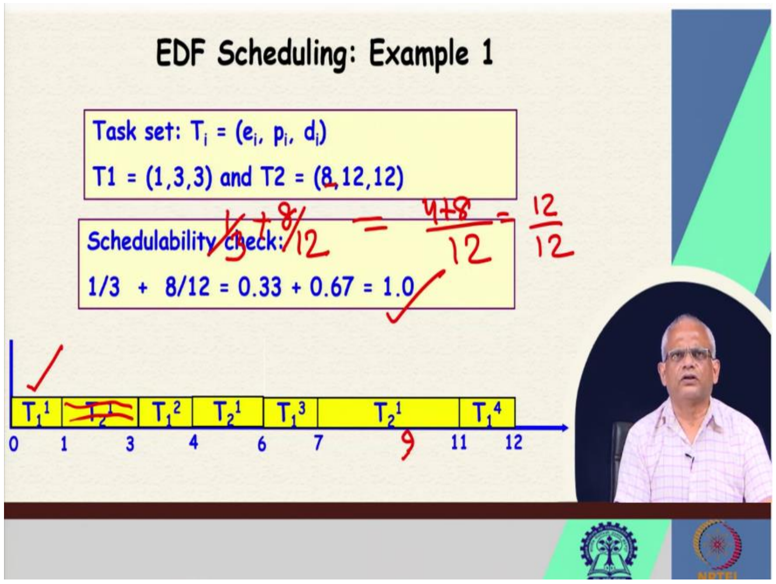
Now, given a task set, let us just have a mathematical expression. By which we can say that whether the task set is feasibly can be feasibly run or feasibly scheduled on the uniprocessor. The formula or the expression for that is very simple, it is $\Sigma^{n}_{i=1}$. There are n tasks t1 to tn and their execution time is e_{i} / p_{i}.

$\Sigma^{n}_{i=1} e_{i} / p_{i}$ for every task that is summation of e_i / p_i for every task in that task set which we write as a Σu_i or utilization, if it is less than 1, less than equal to 1 the task set is feasibly schedulable under EDF, e_i / p_i is called as the utilization of the task, e_i / p_i is the utilization of the task that means every e_i units of time, e_i units of time are required to run the task p_i.

So, the utilization due to task i is e_i / p_i and we do that summation for all tasks. Which we denote by $\Sigma u_i \leq 1$. The task set can be run under EDF and this expression we can think of that as long as the processor is utilized 100 % or less we can run the task set and of course, intuitively we know that we cannot have utilization more than 100 %. So, this scheduler we can imagine that it is a very optimal, it is the optimal scheduler.

As long as the utilization is less than 100 %, it can assure that the task set can be run and what better scheduler we can have. It is the best, the most optimal scheduler and this condition $\Sigma u_i \leq 1$, this is both the necessary condition for the task set to run and also the sufficient condition for the task set to run. So, as long as we have this satisfied, we can just take that the task set can be run.

(Refer Slide Time: 10:59)



EDF Scheduling: Example 1

Task set: $T_i = (e_i, p_i, d_i)$

T1 = (1,3,3) and T2 = (8,12,12)

Schedulability check:

1/3 + 8/12 = 0.33 + 0.67 = 1.0

$\frac{1}{3} + \frac{8}{12} = \frac{4+8}{12} = \frac{12}{12}$

Now let us take an example to see how we apply this criterion, let us say we have 2 tasks, T1 and T2. The T1 has execution time of 1 and period of 1, period of 3 and deadline of 3. T2 has execution time of 8 period of 12 and deadline of 12. Now, can this task set be run on a uniprocessor? We can apply the criterion we studied, we discussed in the previous slide which is sum of utilization due to the task is less than equal to 1.

Now let us find utilization due to task 1. The utilization due to task 1 is that the task needs to run 1 unit every 3 units. What about the utilization for task 2? Task 2 needs to run 8 units every 12 units, so the utilization of the processor due to task 2 is 8 by 12.

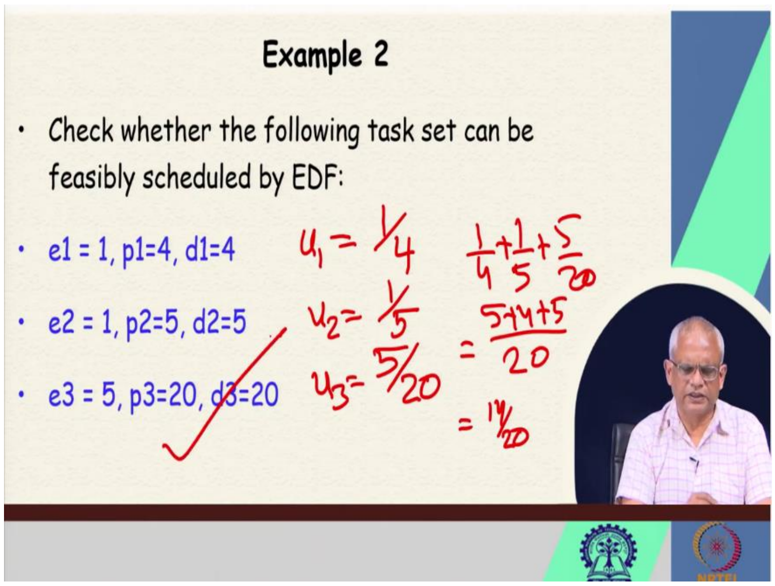
Now, if we simplify this it will become (4+8)/12, so that is 100 % utilization. But, then the criterion is that $\Sigma u_i \leq 1$ and therefore with 100 % utilization we can say that the task set can actually run. Now, lets try to draw the schedule manually and see that if it is really that the schedule given by the EDF, the task set will meet their deadlines.

So, here the shortest deadline first, the T1 and T2 tasks they all started 0 that is phasing, the phase of the task is 0, unless the phase is specified we assume that it is 0 and both tasks become ready at time 0. But, then the deadline of T1 is 3 and deadline of T2 is 12. So, T1 will be chosen by the scheduler, the EDF scheduler will choose T1 and T1 will run for 1 unit of time.

And then T1 completes and completion of the T1 at time instant 1 is a scheduling point for EDF and the EDF scheduler will become active, it will find that in the ready queue only T2 is there. So, it will dispatch T2 and T2 will run for 2 units of time here, until the next scheduling point which is 3. At 3, T1 is a periodic task and the next task instance for T1 arrives and then the next task instance of T1 which arrives at 3, the deadline will be 6, the absolute deadline and the absolute deadline of T2 will be 12 .

So, again T1 will be chosen to run, T2 will be preempted and T1 runs from 3 to 4 and T1 completes T2 is again taken up and then again T1 arrives and T1 again preempts T2 runs and at this instant 6, T2 T1 runs and then T2 runs, but then when T2 arrives at 12 after 6 it arrives at 9, T2 arrives at, T1 arrives at 9, but at 9 the deadline for T1 is 12 and for T2 is also 12. So, T2 will continue to run. According to this algorithm, no preemption will be there and then T2 completes at 11 and then T1 is taken up, so both T2 and T1 they meet their respective deadlines. If we even draw it for long enough duration, we will see that T1 and T2 meet all their deadlines.

(Refer Slide Time: 16:14)



Example 2

- Check whether the following task set can be feasibly scheduled by EDF:
- e1 = 1, p1=4, d1=4
- e2 = 1, p2=5, d2=5
- e3 = 5, p3=20, d3=20

✓

$u_1 = 1/4$

$u_2 = 1/5$

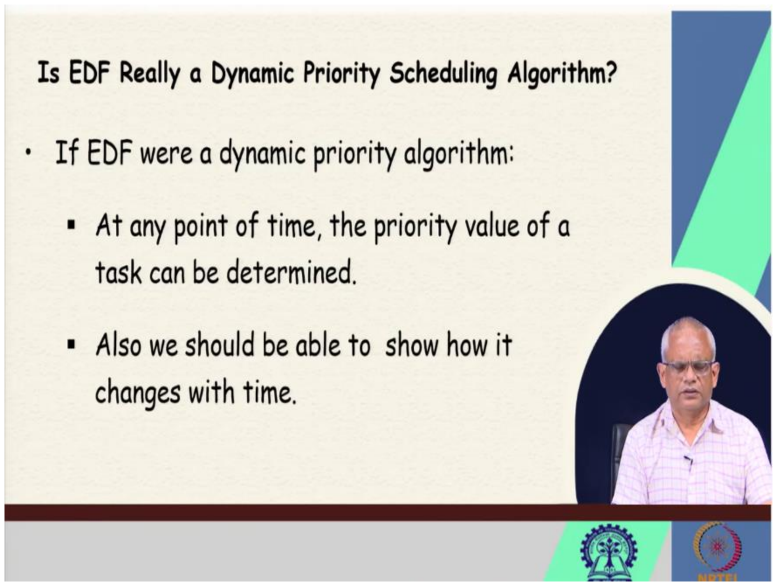
$u_3 = 5/20$

$\frac{1}{4} + \frac{1}{5} + \frac{5}{20} = \frac{5+4+5}{20} = 14/20$

Now, let us look at another example. Let us say we have two tasks the first task execution time of 1 and period and deadline of 4. Second task also execution time of 1 and period and deadline of 5 and also there is a third task, whose execution time is 5 and period and deadline are 20. So, can this task set be feasibly scheduled by EDF. To do this, we need to first compute the utilization due to all the three tasks which is pretty simple for u_1, the utilization $u_1 = 1/4$.

The utilization due to task 2 $u_2 = 1/5$. Utilization due to task 3 $u_3 = 5/20$. Now, the sum of utilization due to all the three tasks is $1/4 + 1/5 + 5/20 = 14/20 < 1$ and therefore the task set can be feasibly scheduled in EDF.

(Refer Slide Time: 17:54)



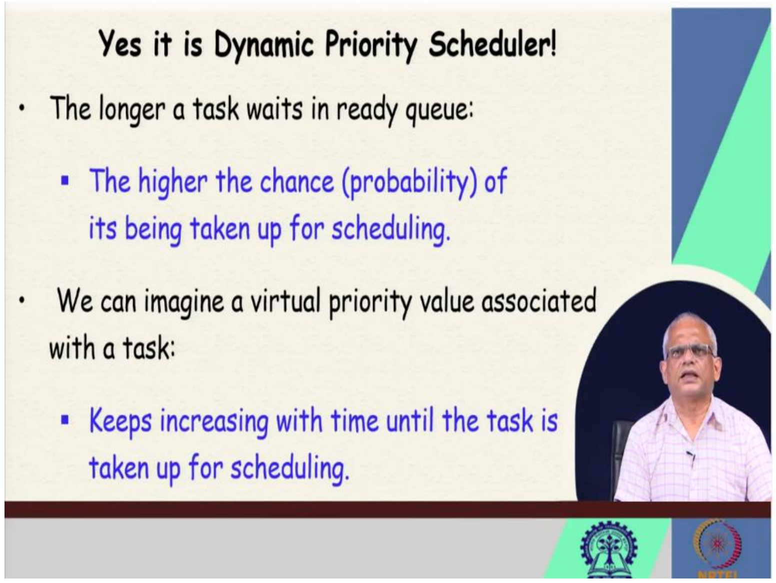
Is EDF Really a Dynamic Priority Scheduling Algorithm?

- If EDF were a dynamic priority algorithm:
  - At any point of time, the priority value of a task can be determined.
  - Also we should be able to show how it changes with time.

So, we know how to apply the simple criterion to check EDF schedulability. But, now let us answer a very basic question, we have so far been telling about EDF being a dynamic priority scheduling algorithm, but if that is the case then we should be a computing the priority or at least at any time we should have a number associated with a task which indicates its priority and how its priority changes with time, at what time instant, what is its priority and etc, we should be able to tell.

But till now the way, we described EDF we are not told neither of that, never said that what is the priority of the task is that a number, how does that number changed with time nothing. So, let us try to see a reason out why we called EDF a dynamic priority algorithm.

(Refer Slide Time: 19:05)



Yes it is Dynamic Priority Scheduler!

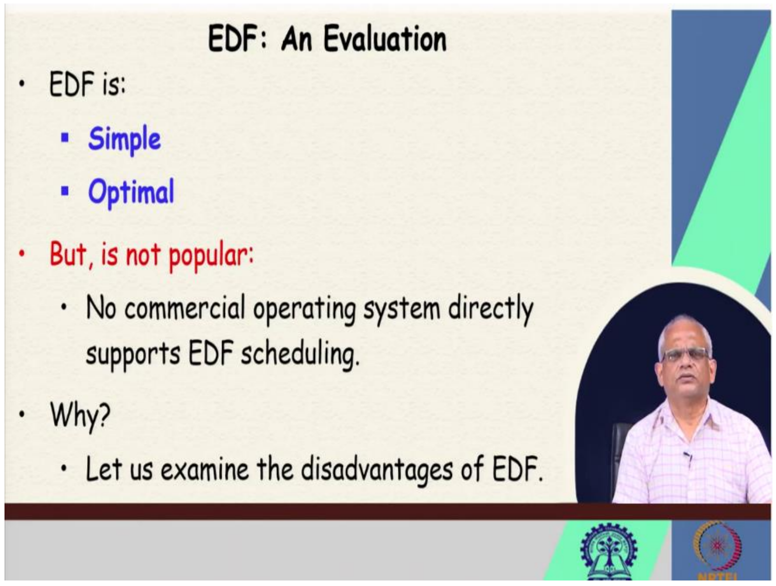
- The longer a task waits in ready queue:
  - The higher the chance (probability) of its being taken up for scheduling.
- We can imagine a virtual priority value associated with a task:
  - Keeps increasing with time until the task is taken up for scheduling.

It is a dynamic priority algorithm, we can reason this way that the longer the task waits in a ready queue, the chances of it being taken up for scheduling is it increases. So, we can think of a virtual priority which is associated with the task and that keeps rising with time. Because, here the tasks are selected by the scheduler based on what is their absolute deadline.

It is not the relative deadline but what is their absolute deadline. The job instances, which one has the shortest absolute deadline. That is the EDF criterion for selecting a task to run and therefore as the absolute deadline approaches, the priority of task can be considered to increase and we can imagine a virtual priority value that is associated with task, it keeps on increasing until the task is taken up for scheduling.

So, just to tell in a nutshell in EDF we do not calculate any absolute number, which gives a priority value of a tasks but we can think of a notion of a virtual priority associated with the task, which keeps on increasing with time until the task is taken up for scheduling.

(Refer Slide Time: 21:04)



EDF: An Evaluation

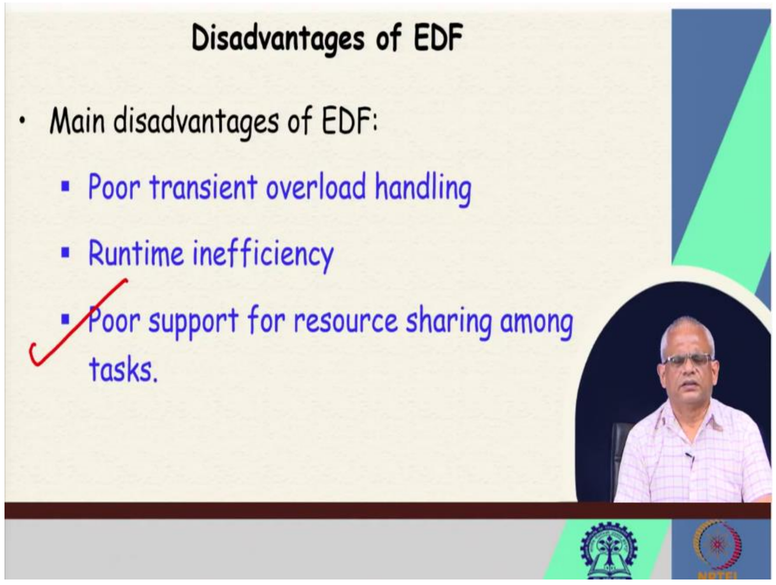
- EDF is:
  - Simple
  - Optimal
- But, is not popular:
  - No commercial operating system directly supports EDF scheduling.
- Why?
  - Let us examine the disadvantages of EDF.

Now let us the algorithm itself is very simple we just saw that algorithm itself is very simple, from the ready queue just to examine the tasks which are having the shortest absolute deadline and select that one to run and this is done at every scheduling point which is a task completion, task arrival and we said that it is a very simple algorithm, nothing much just examine ready queue and find out which has the shortest absolute deadline and select that one, as simple as that.

And also, it is the optimal algorithm. If EDF cannot schedule a set of tasks on uniprocessor feasibly scheduled that is all tasks submit their deadline, then no other algorithm, no other scheduler can feasibly schedule that task set. But very perplexing thing is that even it is so good about this scheduler it is simple; it is optimal but it is not popular.

Rarely used in embedded applications. Let us investigate what may be the reason that EDF is not popular, is there any drawbacks of this algorithm, which is preventing its use in real situations? Yes. So, it is not popular, no commercial operating system provides any direct primitive using EDF for EDF scheduling and the reason is not very far to seek, we will just discuss the disadvantages of EDF and we will immediately feel convinced that no unless the system is bit trivial will not use EDF.

(Refer Slide Time: 23:17)



Disadvantages of EDF

- Main disadvantages of EDF:
  - Poor transient overload handling
  - Runtime inefficiency
  - Poor support for resource sharing among tasks.

One of the main disadvantages of EDF is poor transient overload handling. What it means is that, for some reason if a task which is running under processor takes more time, maybe it got into some path which is taking more time or maybe it is waiting for some signal or some such thing resource, for which it is getting delayed, then the scheduler behaves very funny at this time.

That it does only the task which was running its priority keeps on increasing and as a consequence, even the critical tasks which are waiting, some of them maybe very critical and they may miss their deadline. The second problem is run time inefficiency. The simple implementation of the EDF we saw that it is o(n) order of n if there n is the number of tasks in schedule queue.

But, we might improve it into log n by using a different data structure, but still log n is not very efficient. We will see the rate monotonic we can implement in o(1) time, the scheduler takes o(1) time. So, compared to rate monotonic EDF is inefficient. But then what is the most problematic thing about EDF is poor support for resource sharing among tasks.

In any realistic application, the task should need to share resources and here for EDF we do not have a good solution. As long as there is resource sharing, the task may miss their deadline and this is the big problem. Because, practical situation is that task may need resources and then they share resources and as long as they share resources it is possible that the tasks miss their deadlines and in real situation we would not like that to occur. So, this possibly is the biggest disadvantage of the EDF.

Poor support for resource sharing and also the other two ones, poor transient overload handling and runtime inefficiency. These are some very basic problems with EDF and unless the application is fairly straight forward, where we do not have such critical task for which we are worried that some task getting delayed and the critical task missing deadline is not a matter of concern, runtime inefficiency is not a matter of concern resource sharing is not required only those situations we can use EDF. We are at the end of this lecture, we will stop here and we will continue from this point in the next lecture. Thankyou.