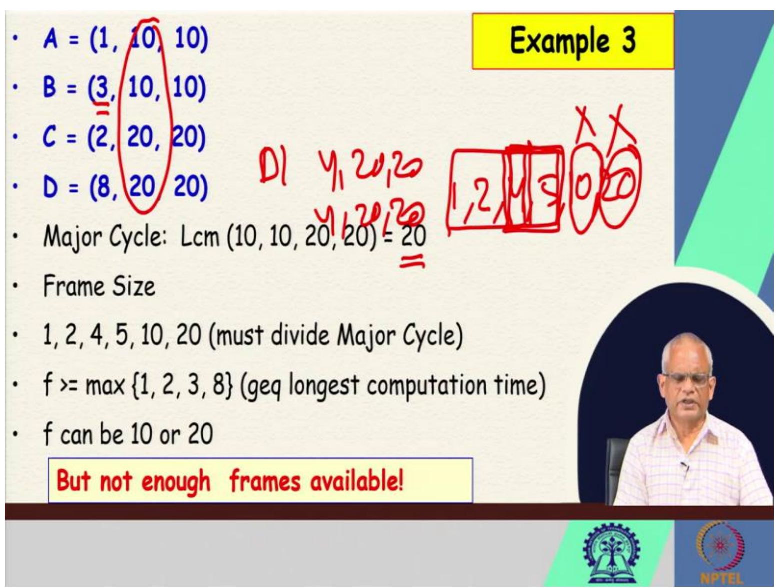
Real Time Systems
Professor. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Lecture 14
Event-driven scheduling

Welcome to this lecture. In the last lecture, we are discussing about the cyclic scheduler and we said that the most challenging thing in a cyclic scheduler for the programmer is to select a suitable frame size. The major cycle can be found out routinely as the LCM of the task periods, but then finding a frame size is the challenge and we had done two examples given task set, how to find a suitable frame size, we satisfies all the three constraints that we had mentioned. Now, let us do one more exercise, so that we become conversant with how to select a suitable frame size for a given task.

(Refer Slide Time: 01:09)



Now, we have a task set a task named A, where the execution time is 1, the period is 10 and the relative deadline is 10 and there is another task B, whose period is 3, whose execution time is 3, period is 10 and relative deadline is 10. And that is the third task execution time is 2 periods is 20 and the deadline is 20 and the fourth task execution time is 8, period is 20 and deadline is 20.

Now, the first thing we need to do is to find the major cycle which is LCM of the task periods, which is LCM of 10, 10, 20, 20, which is 20 and based on this, we need to find the frame size and frame sizes which are possible, which are basically, which divide the major cycle squarely they are 1, 2, 4, 5, 10, and 20. But 20 is ruled out because that will give us one frame per major cycle, but we have four tasks, we cannot use that, 10 also for the same

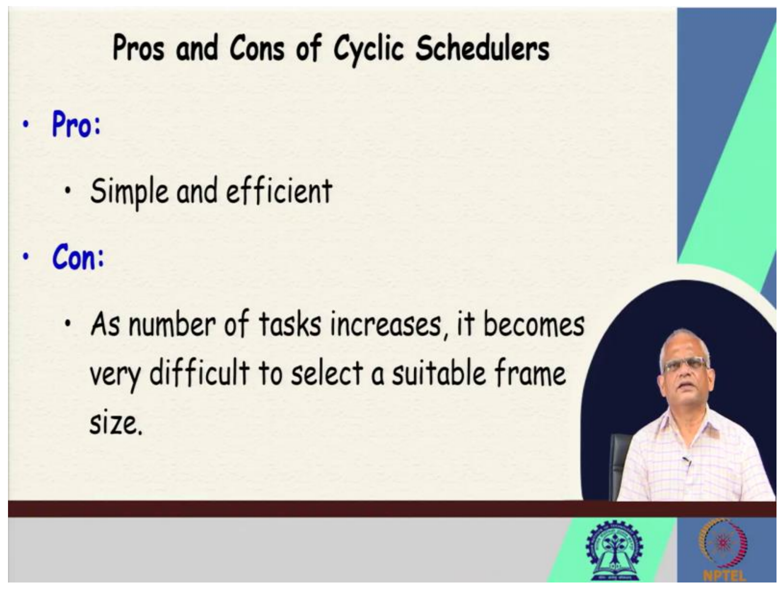
reason we cannot use 10 and neither can we use 1, because the frame size has to be larger than at least 8 because that is the largest task.

So, we cannot use any of this actually, because the frame size needs to be larger than 8 and 10 is not suitable, it leaves us only with two frames. So, we need to split this task 8 into two smaller task, it will be nice if we can split into 4 by 4, two tasks execution time is 4, but due to programming constraint one may be 5 another may be 3.

But still, we can try if we are able to split at least into 5. 3 or 4, 4. So, unless we split D, the task D whose execution time is 8, we cannot find a feasible frame. So, we can split it into 5, 3, so D1 is 4, 20, 20 and D2 is 4, 20, 20 and in that case, we will have five tasks and the LCM is again 20 where the major cycle remains the same and now, we have 4 and 5, ok 5 is still not possible, because that will leave us only 4 frames, but we need at least 5 frames.

So, 4 and you can check whether 4 satisfies the third constraint that is for each task, we have $2*F - gcd(F,P_i) \leq D_i$ leave you as an exercise to check whether we can use 4 for this problem, after the task, we split the largest task into 2. But what if we cannot find even 4 is not suitable, let us say hypothetically, then we can try to split it into 3 or 4. So, that is the way we go about in designing a suitable frame size.

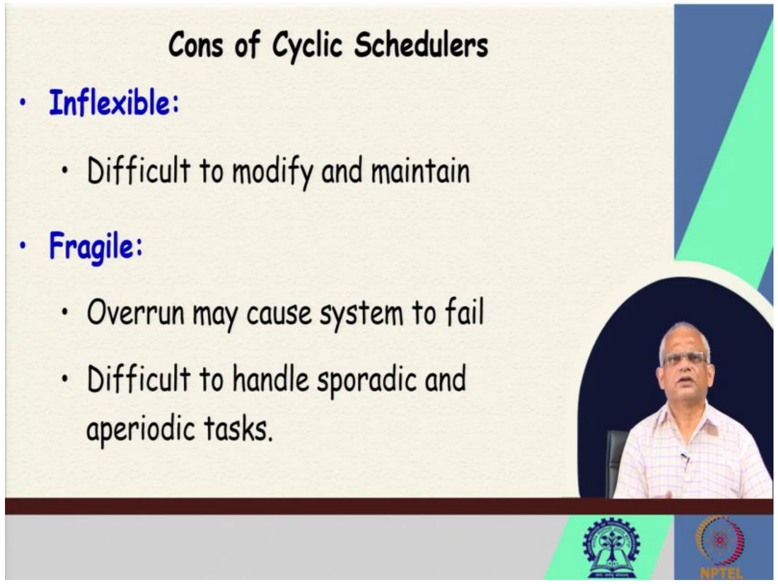
(Refer Slide Time: 05:59)



Now, before we conclude about the cyclic scheduler, the first thing is that it is a simple scheduler, the scheduler itself is the executive the operating system, small operating system, its main role is handling, task scheduling and few other things. Very simple code for the operating system and therefore, very efficient. But, one of the disadvantages of the cyclic

scheduler is that if we have a slightly sophisticated application, where the numbers of tasks are dozen or several dozen it would be difficult to find a suitable frame size.

(Refer Slide Time: 06:56)



And the other difficulties with the cyclic scheduler is that it is inflexible, in the sense that if a task size increases little bit, we need to change everything, we need to let us say due to maintenance, one of the task size increased from let us say 1 to 1.2. So, then we have to again do all that we had done in selecting frame size, maybe splitting some other tasks and so on.

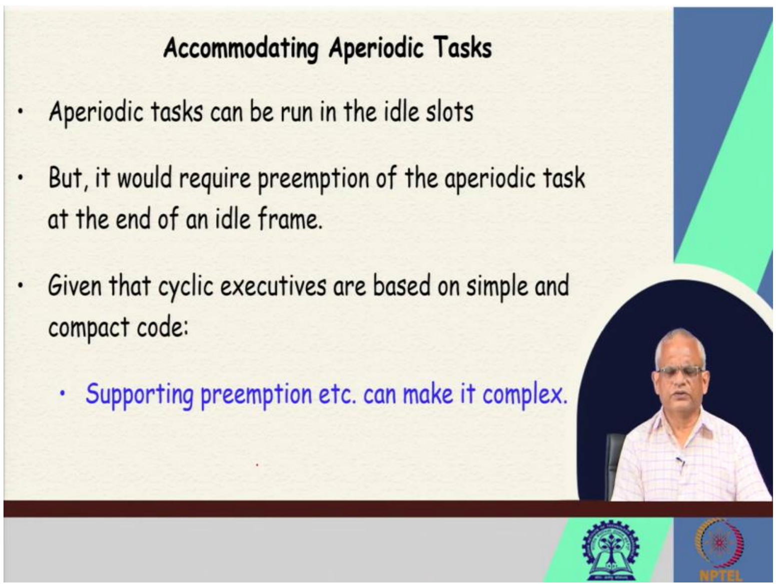
So, it becomes very difficult to modify and maintain and also, if there is a frame overrun, if task overruns its frame, the system will fail. Of course, you can do conservative design, like if the execution time, maximum execution time is 1.5 we will take large enough frames like 4 or something, but still it may frame, it may fail because it may overrun even 4, because the operating system is very simple, we can do nothing if some task takes longer.

And also, since the operating system is so simple, it becomes difficult to handle sporadic and aperiodic tasks. Possibly, we can assign the aperiodic tasks, see remember that aperiodic tasks are those tasks which have random arrival, but they have no deadline, whereas the sporadic tasks have random arrival, but have deadlines. Sporadic tasks will be much difficult to accommodate in a cyclic scheduler, but aperiodic task we can if the aperiodic task size is small, then possibly we can assign them to one of the frames, empty frames.

Whenever there is empty frame and that is a periodic task, we can have the executive assign it to one of them empty frames. But what if the aperiodic task is longer than a frame, then we need to preempt, the aperiodic task when the empty frame ends and then rerun it and run the

remaining part of the task in another empty frame. But that requires task preemption and a simple executive like cyclic scheduler may not have that support for preemption and resumption another empty frames, due to the simplicity of the cyclic schedulers.

(Refer Slide Time: 09:52)



So, aperiodic task can be run, if they are small enough in the idle slots. But if there are larger that would require pre-emption and given that cyclic executives are simple and compact code, supporting pre-emption, etc., it will make the cyclic scheduler complex and some of the advantages of the cyclic scheduler were extremely simple, verifiable, guaranteed run time, and so on. Those will be gone.

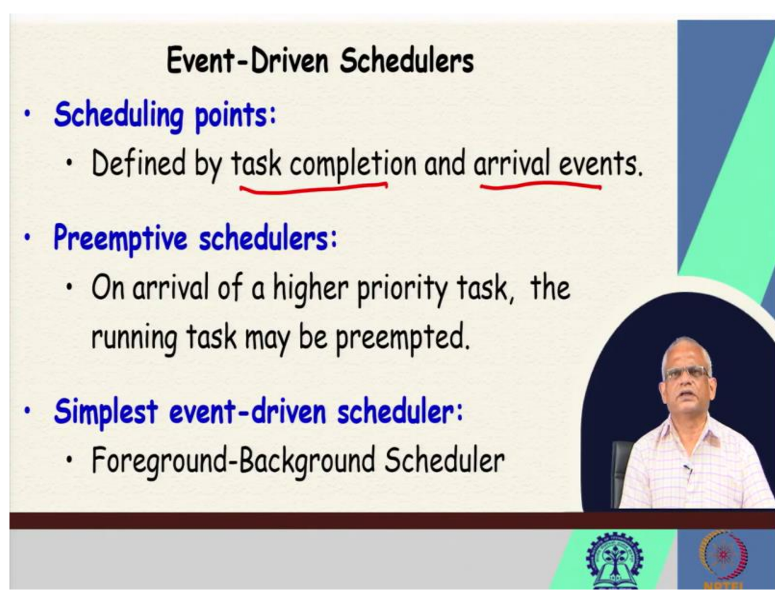
So, we need to weigh that whether we need to really support aperiodic tasks in a cyclic scheduler and if the aperiodic tasks are small enough, we can support it easily. Otherwise, you have to make the executive much more complicated and with that, we will complete our discussion on clock driven schedulers. Now, we will start our discussion on Event Driven Schedulers.

(Refer Slide Time: 11:04)

We had mentioned that in event driven schedulers, we have much more sophisticated applications can be run on this. In clock driven schedulers, we had very simple applications run. So, the event driven schedulers are more proficient. It means that given a task set, which will be very difficult to run on a clock driven scheduler, we can easily run on an event driven scheduler.

And therefore, more sophisticated applications are run on event driven schedulers, only very simple applications, we use clock driven schedulers, otherwise we will use event driven schedulers and also, this can handle the sporadic and aperiodic tasks. So, that is another major advantage and sporadic and aperiodic tasks occur in relatively complex applications. For example, handling a fire condition or an alarm is a sporadic task and needs to be handled in sophisticated applications and that is possible only in the event driven schedulers.

(Refer Slide Time: 12:38)

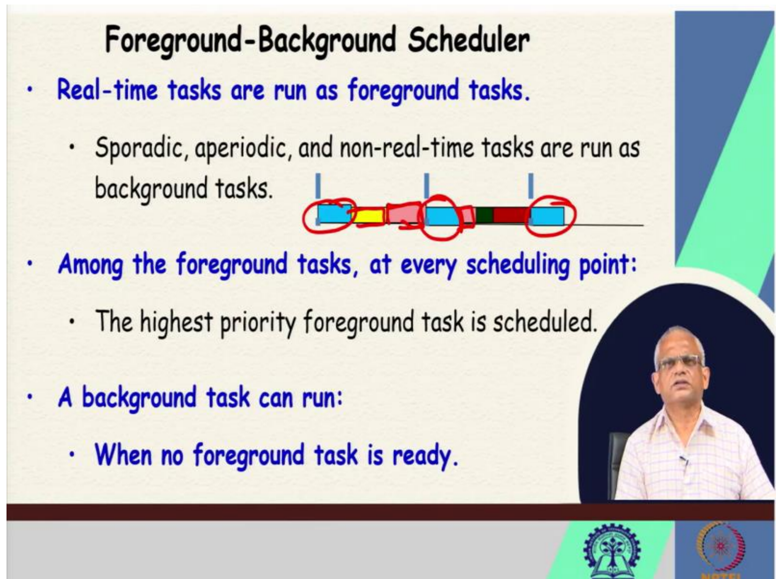


In a clock driven scheduler, the scheduling points are basically the frame boundaries or the interrupts received from a periodic timer. But here in event driven scheduler, as the name says, the scheduling points are not defined by a clock, these are defined by two things, one is task arrival event and task completion event. Task arrival event for periodic tasks can be a timer again, but task completion, a task may complete anytime. So, that is also a scheduling point and some of the tasks may arrive randomly.

So, the task arrival and task completion, these are the events which define the scheduling points and these are preemptive schedulers. The clock driven scheduling was not pre-emptive, a task run in its assigned frame to its completion, no preemption is required. Whereas here, these are preemptive scheduler.

A task may get preempted by a higher priority task agent when it arrives and there are many event driven schedulers we will discuss, a handful of this. But the simplest event driven scheduler is called as a Foreground-Background Scheduler. Let us first look at the foreground-background scheduler and then we will look at more sophisticated event driven scheduler.

(Refer Slide Time: 14:39)

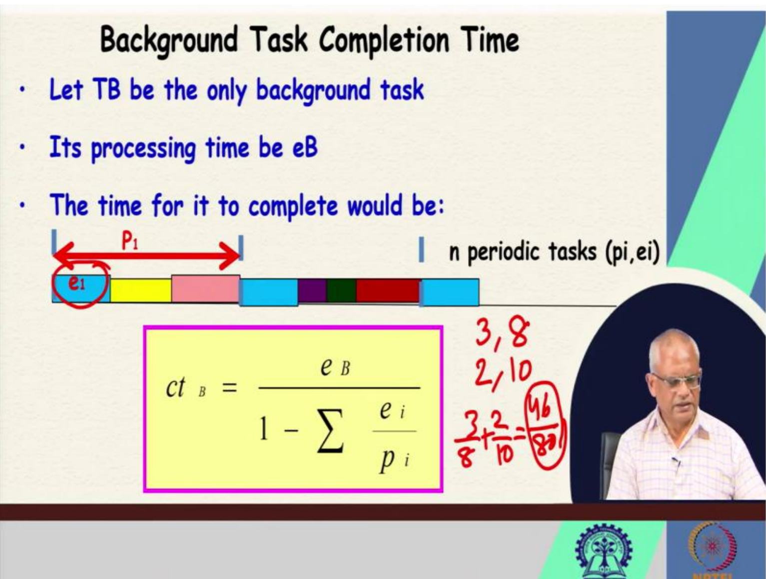


The foreground-background scheduler, here all the real time periodic tasks are run as foreground and the other tasks like non real time tasks, aperiodic tasks, etc. These are run as background tasks. So, here see that the green ones are actually the periodic tasks, sorry the blue ones. So, this is a periodic task and at certain time interval, fixed time interval this task keeps on arriving.

And when this task completes the other tasks can run, there may be multiple non real time tasks and they will be run on a first come first serve basis or first in first out. So, let us say the yellow task ran to completion and this orange task before it could complete again the blue has arrived.

So, the orange has got preempted and then after the blue completes, then the orange is write, the orange is taken up for execution. The black, that executed and then the red, again the red got preempted by the blue task as it came. So, there may be also multiple foreground tasks and at any scheduling point, the highest priority foreground task is scheduled and the background task is run, when there are no foreground tasks.

(Refer Slide Time: 16:25)



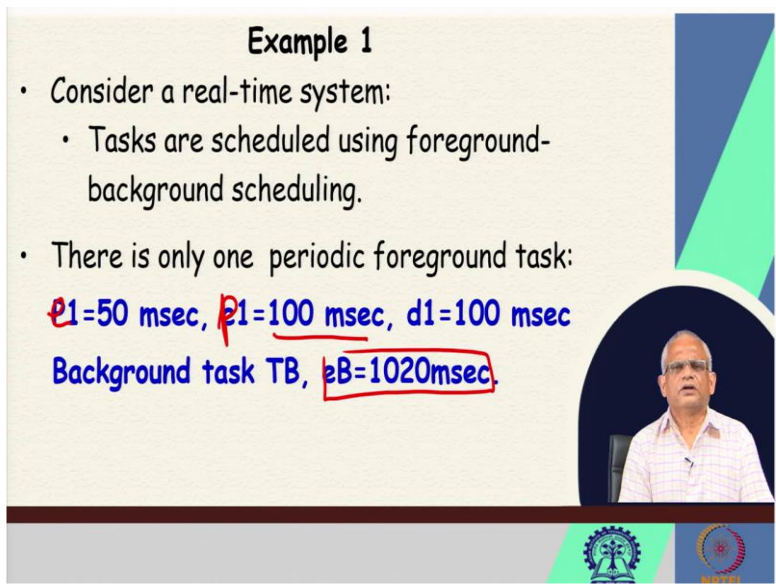
So, here, let us say we have only one background task, which is called as TB, T background and let us say the processing time for TB is eB. Now, how much time we need to complete the background task. So, that is one problem that we need to analyze the performance of these schedulers. So, we have n periodic tasks and these are defined by the execution time ei and their period pi. So, this is the execution time of a periodic task and e1 and P1, similarly e2, P2 and so on.

So, to find out how much time the background task will take to complete, we can use a simple formula here, eB is the execution time of the background task and Σ (ei / pi) is the total utilization of the processor due to all the periodic tasks. Let us say we have two periodic tasks. Let us say they take 3 execution time in 8 units, and let us say 2 execution time in 10 units, we have two tasks.

So, the utilization due to these two tasks is (3 / 8) + (2 / 10) which is (46 /80). So, what it means is that every 80 units time, 46 unit goes to execute these periodic tasks and had said that the background tasks will execute when no periodic tasks are there and that is given by 1- (46/80) = (34 / 80).

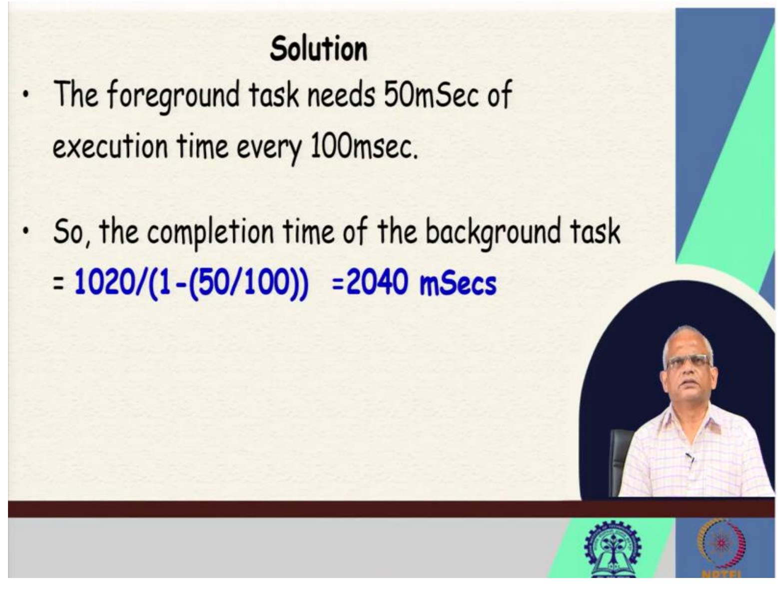
So, every 80-time units, the background task can run 34 units and how much will the background task take to complete that is (eB) / (1 - Σ (ei / pi)). So, very simple argument here, find the completion time of the background task.

(Refer Slide Time: 19:12)



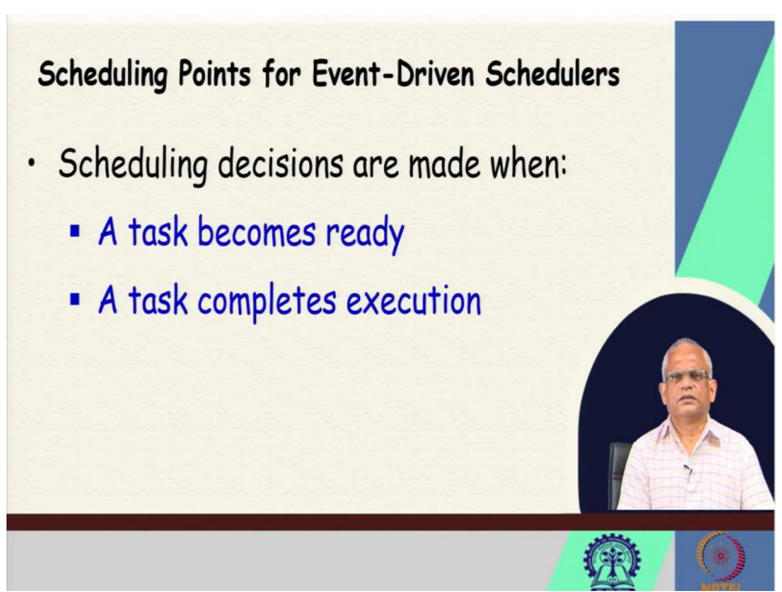
Now, let us just do one example. We have one periodic foreground task, whose the execution time is 50 millisecond and the period is 100 millisecond and deadline is 100 millisecond and the background task whose execution time is 100, is 1020 millisecond, execution time of the periodic task is 50 millisecond, period is 100 millisecond and deadline is 100 millisecond. So, what it means is that every 100 millisecond 50 millisecond goes to execute the periodic tasks or we can say that every 100 milliseconds, we can execute the background task for 50.

(Refer Slide Time: 20:08)



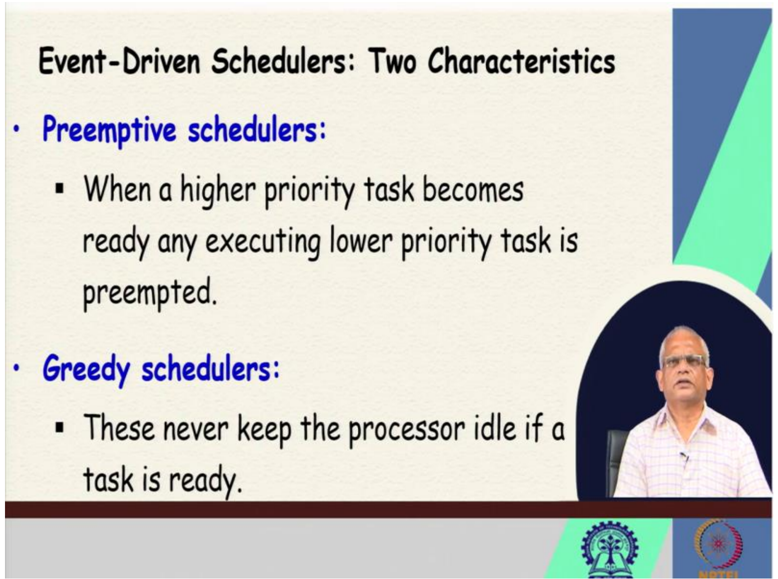
So, we can use that to write 1020 / (1-(50/100)), so this becomes 2040 milliseconds. You can try this out that the time to complete the background task is 2040 milliseconds.

(Refer Slide Time: 20:29)



The scheduling points of the event driven schedulers are actually two, there are two points where scheduling decision needs to be done. One is when a task arrives, the task becomes ready and the other is task completes its execution, these are the two instants and time on which the scheduler becomes active and then selects a task to run and runs it.

(Refer Slide Time: 21:09)

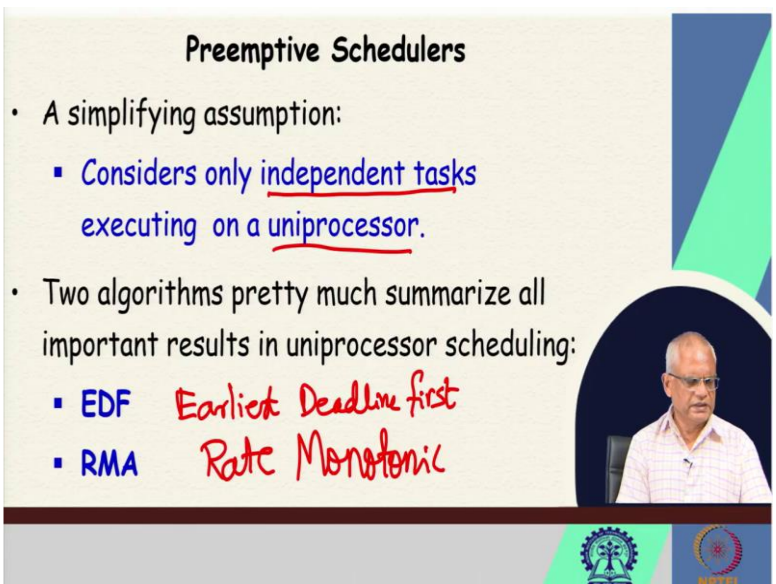


And these are as we were saying that these are preemptive schedulers. If a higher priority task has arrived, an executing lower priority task will be preempted. So, the operating system should support preemption. That is the context of the lower priority task will be saved and later when required that will be run.

And another characteristic of the event driven scheduler is that these are greedy. If a task is ready, then it will be run if there are no other higher priority tasks waiting. So, if as long as

there is a task to run, the processor is not kept idle and that is why you call it as a greedy scheduler.

(Refer Slide Time: 22:01)



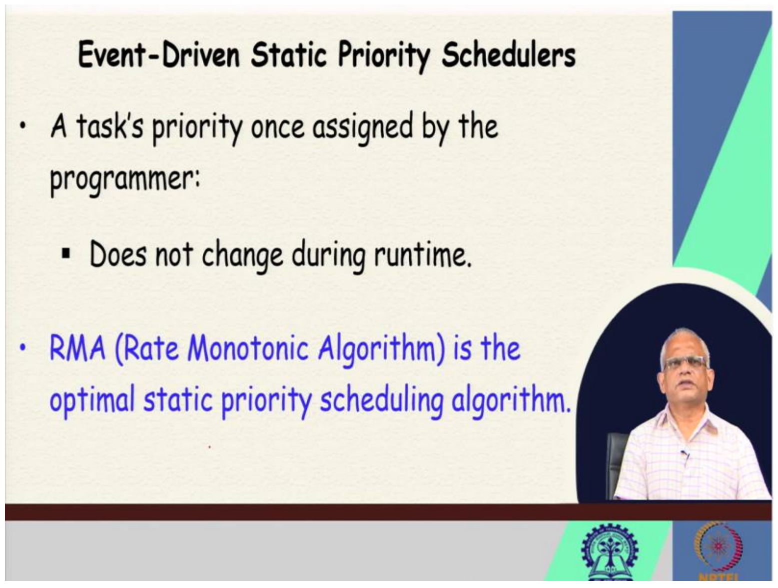
And initially, we will consider only independent tasks, executing on a uniprocessor. So, these are two assumptions, we will initially make to simplify the scheduling problem and later, we will remove those two assumptions. The first assumption is that all tasks, they are independent. It is not that after a task completes, the task will run then they become dependent and also, we will consider uniprocessor. Later we will remove these two restrictions.

And we will see how these event driven schedulers can be modified for a multiprocessor and when the tasks set are dependent, because that is a realistic situation, that we have a multi core processor or a number of processors in a distributed system and also, there is dependency among tasks that only after one task completes another task can be taken up for execution.

Now, if we make those two assumptions that independent tasks and uniprocessor then we have two scheduling algorithms, which summarize all the basic results that have so far been obtained in event driven scheduling on a single processor. So, if we know these two schedulers, so we know almost everything about event driven scheduling on uniprocessor. Of course, there are small variations of these, but if we know these two, then we know almost everything about event driven scheduling on uniprocessor.

So, let us look at these two scheduling techniques. Because they are a result of years of research, very popular and really used in many applications. So, let us look at these two and focus on these two. The first one is called as EDF, Earliest Deadline First, which means that the task whose deadline is the earliest that will be taken up first for execution. The rate monotonic scheduler, so that is another, so the rate monotonic scheduler and EDF these are the two major results or the major types of schedulers that are used in uniprocessor scheduling.

(Refer Slide Time: 25:21)



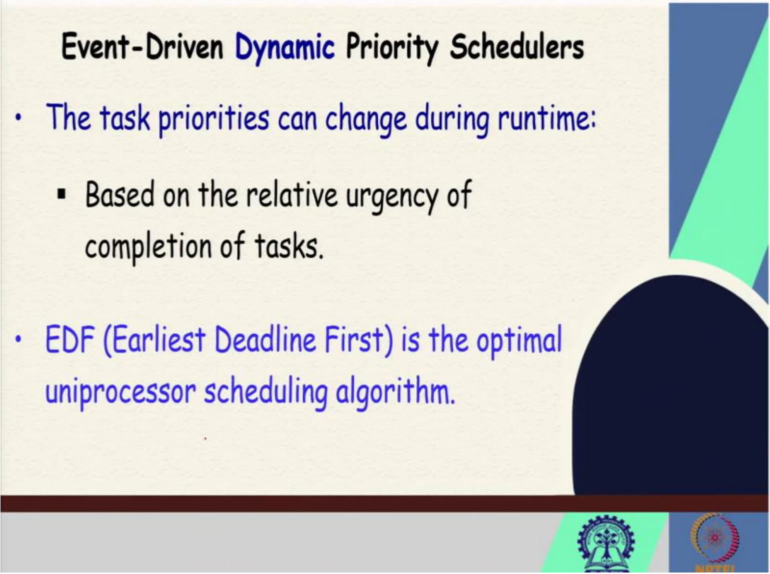
Now, the rate monotonic scheduler is a static priority scheduler. The EDF is a dynamic priority scheduler. In a static priority scheduler, the designer of the system or the programmer, he fixes or assigns priority to tasks and once it assigns priority that remains fixed throughout the applications lifetime. It does not change in the runtime.

Unless the programmer again changes the priority order and again, runs the system, it does not change. It does not change dynamically during runtime and that is why these are called a static priority scheduler. The programmer assigned, the programmer assigned priority remains throughout the running of the application.

And the rate monotonic scheduler is the optimal static priority scheduling algorithm, which means that this is the best scheduler, best static priority scheduler, if the rate monotonic scheduler cannot run, a set of tasks then any of the static priority algorithm, we may think of that would not be able to run these task set.

So, this is optimal. So, the rate monotonic schedulers are used extensively. If you develop a real time application, it is very likely that you use a rate monotonic scheduler, it requires very less support from the operating system as well. So, the scheduler is very popular.

(Refer Slide Time: 27:33)



On the other hand, a dynamic scheduler, dynamic priority scheduler, the task priorities change during runtime. The priority of a task may become very high when the task is about to, its deadline is approaching, but when its deadline is far away, the priority may be lower and for that all dynamic priority algorithms that we may think of, the EDF is the optimal uniprocessor scheduling algorithm.

So, that means that whatever dynamic priority algorithm that we may conceive, we may propose any dynamic priority algorithm. But if our algorithm, new dynamic priority algorithm is able to run some tasks set, then it will be runnable also by EDF, but not vice versa that if EDF is running then the other dynamic priority algorithms may not be able to run those. So, very important, schedulers both rate monotonic is optimal static priority algorithm and EDF is the optimal dynamic priority scheduling algorithm.

We need to look at these two in detail, need to understand exactly how to go about developing applications using a static priority algorithm and dynamic priority algorithms that is RMA, the rate monotonic and the EDF. We are almost at the end of this lecture. We will start looking at the rate monotonic scheduler and the EDF in the next lecture. Thank you very much.