

Foundation of Cyber Physical Systems

Prof. Soumyajit Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Lecture - 13

Real Time Task Scheduling for CPS (Continued)

Welcome back to this course on foundations of Cyber Physical Systems. So, in the last lecture, we started with this proof on whether these non-EDD schedules are optimal or EDD schedules are better. So, let us move ahead from that point.

(Refer Slide Time: 00:47)

Proof

- A non-EDD Schedule must have a task τ_i preceding a task τ_j with $d_j < d_i$
- Since τ_i and τ_j are independent, reversing them yields a valid schedule.

Actual (Schedule 1)

• Max Lateness
 $L_{max} = \max(f_i - d_i, f_j - d_j) = f_j - d_j$
Since $f_i \leq f_j$ and $d_j < d_i$

Reversed (Schedule 2)

• Max Lateness
 $L'_{max} = \max(f'_i - d_i, f'_j - d_j) = f_j - d_j$
Since $f'_j \leq f'_i$ and $d_j < d_i$

So, I think we have been discussing this. So, we had these two options of a two-task system. We have an EDD schedule and we have this non-EDD schedule here, right. So, if you look at the non-EDD schedule, we have a task which has this deadline, in this case you have this task which has a deadline that is later on, right. This task i which has a deadline later on, it is executing first. But in this case, a task which has a deadline which is earlier is executing first, right.

So, this is the basic difference between these two tasks here. So, in one case we saw that well this is our max lateness equation. In the other case we have this, well the max lateness is given by these two these two differences, right. It is either this or this and we made one observation, that well, f_i' is nothing but this is equal to f_j , right. Because that is the point when both tasks finish. So, f_j must be equal to f_i' , right.

So, please ignore this part here and we will see that well, what really is the case? Can I show that in both, I mean, the EDD schedule will give me max lateness value that is smaller than the non-EDD schedule? If we can prove that we are done, we have shown that we have at least in the two-task system, whatever is the EDD schedule, that is actually minimizing the max lateness. So, let us move ahead. So, what we really have here is this situation, right.

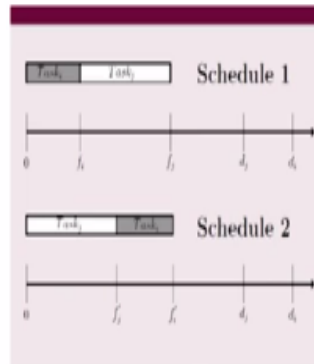
I mean, $f_i' - d_i$ and $f_j' - d_j$. So, let us see in, let us do a case analysis that suppose this is the resulting max or I mean in both cases what really happens? Is this situation better than this situation with respect to the lateness? That is how we will do a case analysis. And why we have to do the case analysis? Let us understand. I will just repeat this that these two intervals it is not that they are contained inside each other something like that, right.

So, this is all symbolic. So, we need to see in both cases what is happening. So, let us go forward and analyse this, both of these situations.

(Refer Slide Time: 03:31)

Proof (Contd.)

- Case 1: $L'_{max} = f'_i - d_j$. Since $f'_i = f_j$ and $d_j < d_i$, $L'_{max} = f'_i - d_j \leq f_j - d_j$. Hence, $L'_{max} \leq L_{max}$.
- Case 2: $L'_{max} = f'_j - d_j$. Since $f'_j \leq f_j$, $L'_{max} \leq f_j - d_j$. Hence, $L'_{max} \leq L_{max}$.



- Schedule 2 has maximum lateness no greater than that of Schedule 1
- EDD Schedule (Schedule 2) has minimum maximum lateness of all schedules.



So, there will be a case on, where we will be considering that well, this $f'_i - d_i$ is the maximum. So, let us see that situation, that $f'_i - d_i$, let us say that is the bigger one, right. Now observe one thing like we said earlier that f'_i is f_j , right. So, and of course we know that d_j here is smaller than d_i . That has been our initial task setting, right. So, if we have $f'_i = f_j$ and $d_j < d_i$, we have $f'_i - d_i \leq f_j - d_j$.

I mean that that is basically the same value. But earlier we were subtracting d_i and now we are subtracting a smaller value, okay. So, if I just subtract a smaller value here, right, then this is bigger, right. So, I have $f_j - d_j$ to be larger than $f'_i - d_i$. So, I will just repeat what we are doing is this is the max lateness, $f'_i - d_i$, right. Now what we are saying is let us remove f'_i with f_j , and let us see what happens. I mean, now instead of subtracting d_i , if I subtract d_j , right.

So, if instead of subtracting d_i as we, so essentially if I look at this figure. So, again this is same as f_j , right. They are both same. Instead of subtracting this if I subtract this, right. So, I have a smaller magnitude, right. But so, this is a smaller magnitude. So, if this is $f_j - d_j$, right, and this is a smaller magnitude but it is a negative. So, what I will have is a negative smaller magnitude is going to be larger than this one, right. So, in that way in this case I have this and if you observe what is $f_j - d_j$?

If you just turn over to my previous slide and that exactly is L_{max} . That means for the EDD schedule, the non-EDD schedule, what is maximum lateness, right? So, I can write this $L'_{max} \leq L_{max}$. So, in case, in this maximum lateness the first, the first argument is really the max, which is case one. We have shown the maximum lateness here is smaller for the EDD. Now take the other argument, which is $f'_j - d_j$, right.

So, if you take that one, then this is your maximum lateness L'_{max} , right. Now we know for one thing that well, $f'_j \leq f_j$, right because f'_j is here and f_j is here. So, I can always write, just replace f'_j with f_j . So, then if I consider this interval $f_j - d_j$, right. So, this again is the smaller interval smaller in magnitude. That means I can just say that well $f_j - d_j \geq L'_{max}$.

Because it is negative with a larger value with respect to that something negative with a smaller value will be larger than this, right. So, and this is again L_{max} , right. So, in this case again I can argue that $L'_{max} \leq L_{max}$. So, in effect using both the cases, what we are able to show is that this L'_{max} which is the maximum lateness is smaller than the L_{max} which is the maximum lateness for non-EDD, right. So, this is it, right.


So, when I am doing, when I am following EDD, I am minimizing the maximum lateness for the tools task system. In general, is a difficult proof but we can definitely show that for any number of tasks this argument will hold, right. So, this is the statement that we can make that schedule two has maximum lateness which is no greater than that of schedule one. So, we I mean that is that has been our entire hypothesis.

So, we can say that EDD schedule, that is the schedule 2, it has minimum value of maximum lateness of all schedules, because of all schedules, because in a two-task system you just have two possibilities. So, this is a simple proof but as you can see that there are some cases and arguments to be made here. But we are able to show that why EDD works.

(Refer Slide Time: 08:55)

Earliest Deadline First (EDF)

- ▶ Limitations of EDD
 - ▶ Does not support arrival of tasks
 - ▶ Does not support periodic execution of tasks
- ▶ EDD is extended to support these - EDF or Horn's algorithm
- ▶ Given a finite set of non-repeating tasks $T = \{\tau_1, \tau_2, \dots, \tau_n\}$ with associated deadlines d_1, d_2, \dots, d_n and arbitrary arrival times, any algorithm that at any instant executes the task with the earliest deadline among all arrived tasks is optimal with respect to minimizing the maximum lateness.



And in general, when you want to put EDD in practice you need to support arrival of tasks, right. Because the idea of EDD is that well you are given a static task set. So, if you extend this with arrival of tasks, then we have this algorithm for of Earliest Deadline First execution, the EDF algorithm. So, EDF is an algorithm or Horn's algorithm, it supports arrival of task and the, I mean, better to say periodic arrival of tasks, okay.

So, if I am given a finite set of non-repeating tasks with associated deadlines and some an arbitrary arrival times. So, observe this the arrival time here would mean the initial offsets of the tasks, okay. I mean, an arbitrary and any algorithm that at any instant executes the task with earliest deadline among all the arrived tasks is optimal with respect to minimizing the maximum lateness. So, this is what we have shown for the small example.

And we are saying that well this is true in general. You are given this task set and you are told that well for this task set you have this these are the current deadlines, right. And the tasks can arrive at any point, okay. And among all the different scheduling algorithms which you can use to order the execution of the tasks, that algorithm which uses the notion of earliest deadline for deriving the ordering of the tasks is the one which is optimal in minimizing the maximum lateness.

So, that is what we are arguing. Now if we can apply this hypothesis time and again. So, suppose now one new task instance arrives. So, your set of arrive tasks has changed, right. So, for this set of tasks you calculate that what are the deadlines, okay. And the one which has the earliest deadline you start executing it, okay. So, that is how EDF will work you have a set of task instances that are there in your queue. For them you check, you have checked that which one has the earliest impending deadline, right, I mean, and you execute that.

And in while doing this well another task may be added to the system and if that task has even earliest deadline, you will re-evaluate. That is why EDF is also pre-emptive, right. So, suppose while executing some task which was earlier having the earliest deadline, a new task has arrived your arrived set has changed, right. So, look at this argument. Here we are not talking about repetition.

We are looking at it for a single instance. We are saying that at this instance you have a set of non-repeating tasks and associated deadlines and arrival times. And you are saying that well among them whichever one has the earliest deadline you execute it. Now extend this argument to the periodic case. So, a new task will arrive. Whenever the new task arrives, it is almost I mean just look at it again that you have you have this same non-repeating task set, but it has extended with this new instance of arrival of some task, okay.

And then again you apply the algorithm. That means you see that among these pending task instances of different task types which one has the earliest deadline. If this one is something that is pending with respect to something that was executing, you preempt it and again execute. So, that is how EDF will really work. And it can be shown that among all such algorithms which are the, so now what is happening.

Whichever task has priority to be high or low, that is changing based on the instance of the task. Some instances may have lower priority but depending on arrival times depending on other tasks depending on how much time this task is waiting, some instances can have a lower priority or a higher priority, right. So, these are dynamic priority algorithm.

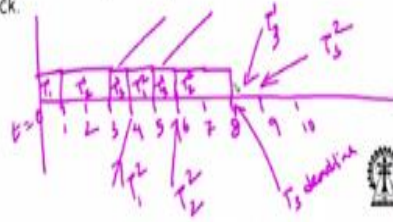
(Refer Slide Time: 12:56)

EDF : An Example

Are the following tasks with given specifications RM Schedulable?

Tasks	p_i (ms)	e_i (ms)
T_1	4	1
T_2	6	2
T_3	8	3

Utilization = $1/4 + 2/6 + 3/8 = 23/24 = 0.9583 > 3(2^{1/3} - 1) \approx 0.7798$. Hence, might not be RM Schedulable! Let's check.



So, let us take one small example here. So, let us look at this task set. You have three task T_1 , T_2 , T_3 , and you have their periods 4, 6 and 8 let us say in milliseconds, and executions time 1, 2 and 3. So, if you calculate their execution time, I mean, given this periods and utilization time, let us do their utilization check. That suppose I am trying to first see well you are these tasks together RM schedulable? Can I execute the RM algorithm on these tasks on this periodically repeating tasks, and still get a valid schedule.

But you already have that utilization constant. You know that with RM, if I have a three-task system, my maximum utilization that the RM algorithm will provide me is this. So, what is this? If you remember that formula for RM for computing this upper bound on the maximum utilization that an RM algorithm can give, $\mu_{max} = n(2^{\frac{1}{n}} - 1)$. So, just put $n = 3$, it gives you this 0.7798 something like that. So, that is the maximum utilization you can have for a task set of 3 periodic tasks that RM can give you, right.

But in this case if I have to satisfy this task set with this execution times the utilization, I really require is something like this. So, you see $1/4 + 2/6 + 3/8$, it goes to something like this so, this is 6, 8 and 9. So that is it, right. So, this is the requirement of utilization for executing this task set.

And the maximum utilization for a set of three tasks that RM can support is this, right. So, the requirement is much more than what can be supported by RM, right.

So, this might not be as per the math here it is not RM schedule. So, let us see that well its actually so. So, I mean let us try to create an RM schedule here. So, I am just trying to roughly draw a timeline here. So, if I follow Rate Monotonicity, T_1 has the highest period, followed by T_2 and then T_3 . So, all I mean, there is no offset here. That means all the instances, the first instances of all the tasks, we can assume that they have all arrived at, right at the point $t = 0$.

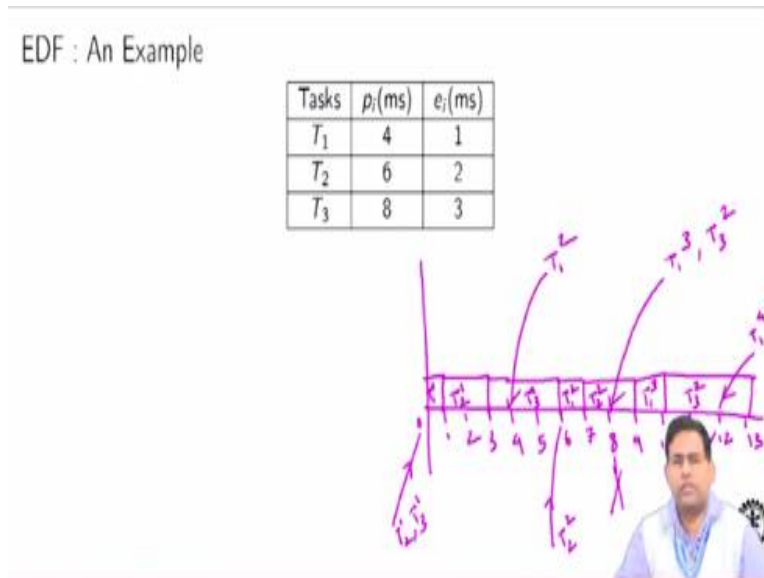
So, T_1 being the task with highest priority we will execute here, T_1 's first instance. And then T_2 will execute, right. T_2 will take two cycles. So, T_2 's first instance. Then T_3 which has an execution requirement of three cycles, will start to execute. But you see, I mean, T_3 is going to take three cycles that means T_3 is going to finish here. And then T_1 's first instance is done but T_1 's second instance is due to arrive here, right, at this point.

At this point T_1 's second instance comes and being of the highest priority. This is T_3 's is first instance and here you will have T_1 's second instance coming and getting executed. So, once that is done, T_3 's first instance can resume. So, it will resume, but then again T_2 's second instance will arrive at this point and that has a higher priority than T_3 , right. So, here T_3 is. In this case here, T_2 's second instance will start and it will take two cycles to execute and it will end here, right.

Now, so if you see T_3 's first instance which was supposed to take three cycles has it completed? Not really. Because here it consumed one cycle here it consumed the second cycle. So, here T_3 's first instance, if it gets one cycle free CPU it can execute. But the issue is this is deadline, right. This is T_3 's deadline. By this point we are we are considering period equal to deadline here and at this point T_3 's second instance has also arrived.

And T_3 was this is first instance was supposed to be executed inside its period equal to deadline equal to 8 millisecond. So, that is why, that is just to show, I mean with the example, that will RM will not really work here, tasks will start missing their deadlines.

(Refer Slide Time: 19:08)



So, now we can just check well is EDF going to work? Let us try to construct this schedule again, but now with respect to EDF. So, you have T_1 's first instance starting and completing right there. And now let us see, I mean, what is EDF doing. So, at any point whenever a task arrives it will see which one is the earliest deadline and it will just execute it, right. So, all the first instances have arrived at $t = 0$, and T_1 as the earliest deadline so, that is now executed, right.

So, when is T_1 's next instance going to come? So, here T_1 's second instance is going to come, here T_1 's third instance is going to come. T_2 's second instance is going to come here. This is where T_2 's first instance has come. This is where T_2 's third instance will come. T_3 's first instance is also coming here, and T_3 's second instance is also coming at 8, okay, and that is how it is. So, if you see here T_1 's first instance is done and T_2 's first instance is pending.

T₃'s first instance is also pending and T₂ has a earlier deadline with respect to T₃. So, T₂ can execute for two cycles and it will get done here. And then now comes the thing. T₃'s first instance will now start. Because now nothing is pending, right. So, T₃'s first instance starts. But then T₁'s second instance comes right here. Earlier in RM, T₁'s second instance was preempting T₃'s first instance. Will that happen now? Actually no.

Because you see, T₃'s first instance that has deadline up to when? Of course, up to T₃'s second instance which is right here. And T₁'s second instance has come. T₁'s third instance, well that also has a deadline which is right here, right. So, it is like a tie I mean none of them I mean I cannot say here that T₁ has a higher priority. So, in that case I will just choose that well whatever was executing let it just execute. So, I will just let it consume this time scale and T₃'s first instance will just end.

So, as you can see T₃ is second instance, I will just repeat what is going to come here at 8 and T₁'s. So, that is the deadline of this. T₃'s current instance the deadline is this. And T₁'s second instance which came in between for it also that deadline is this. So, I am not preempting at all. So, this is a key difference here with respect to EDF. So, nobody misses their deadline and T₃'s first instance gets executed.

Now after this well T₁'s second instance was pending and T₂'s second instance has also come at this point. But T₂'s third instance, and that is quite late here. But T₁'s second is this earlier deadline. So, definitely T₁'s second instance will get executed. And after that T₂'s second instance can just start. Because right now, nothing was pending. Now at this point while T₂'s second instance was executing something at higher priority and something at lower priority both came, with respect to RM.

But they are not really having a higher priority because for both of them the deadline is much later. The actual deadlines, T_1 's third instance is this, T_1 's fourth instance the deadline is here. And T_3 's second instance has come here and that deadline is also far away, right. So, now nobody will preempt and T_2 's second instance will also complete, right. And then after this well, you will execute T_1 's third instance. And then you will execute T_3 's second instance.

And in this way this task set will I mean if you draw is further, I mean I do not think there is any point in for me to continue this. So, if you just keep on drawing this further, you can see that well for this task set at least things are nice and schedulable. And you can just schedule this task set with EDF. Now the question is well how do I know that a given task set is, I mean, schedulable with EDF.

(Refer Slide Time: 25:39)

EDF : An Example

Are the following tasks with given specifications RM Schedulable?

Tasks	p_i (ms)	e_i (ms)
T_1	4	1
T_2	6	2
T_3	8	3

Utilization = $1/4 + 2/6 + 3/8 = 23/24 = 0.9583 > 3(2^{1/3} - 1) \simeq 0.7798$. Hence, might not be RM Schedulable! Let's check.




Well for EDF you can actually achieve a utilization which is up to 1, unlike RM, for which the achievable utilization is bounded like this. With EDF the upper bound is one.

(Refer Slide Time: 25:51)

More on EDF

- ▶ Dynamic priority scheduling algorithm
- ▶ Optimal w.r.t. feasibility among dynamic priority schedulers
- ▶ Minimizes the maximum lateness
- ▶ Results in fewer preemptions
- ▶ An EDF schedule with less than 100% utilization can tolerate increases in execution times and/or reductions in periods and still be feasible
- ▶ Not optimal if there are precedences



So, if we just want to summarize this. It is a dynamic priority scheduling algorithm, that means the priority assignment to task instances is a dynamic, say, for the same task different instances can have different priorities. It is optimal with respect to feasibility among all dynamic priority scheduler. So, just like for all fixed priority pre-emptive schedulers, RM was optimal with respect to feasibility. EDF is optimal with respect to feasibility among all this dynamic priority schedulers.

Of course, the pre-emptives also. It minimizes the maximum lateness like we showed using that example for the two-task system. And the good thing about EDF is, it results in fewer pre-emption's. If you remember our previous example, in many cases, where RM would have actually a task arrival would have resulted in a pre-emption because of the fixed priority thing. But we actually saw that well, although a task arrived but for the existing task which is executing due to its pending deadline it had higher priorities.

And that is why there was no pre-emption. So, that is why we are saying that typically EDF will result in fewer pre-emption's which those efficient because more for a practical implementation pre-emption's lead to a context switch over it because the processor has to restore the state of some system, some task stored the state of some executing task etc., etc. So, all that time is not lost. An an EDF schedule which has less than 100 percent utilization, it can tolerate increase in execution

time or reduction in periods and still remain feasible. Because it has some slack. So, suddenly due to some uncertainty the execution time increases or a period reduces, it can tolerate it better. Because it can go up to 100 percent and if it is less than 100 percent it has got that slack available, right. However, if you see all the algorithms we have discussed till now, none of them have taken precedence relation among tasks into action into consideration.

And what we will do in the next little lecture is we will now bring in these precedence constraints, and see that well with such precedence relations well how task scheduling can be done in a real time system. Thank you.