

Foundation of Cyber Physical Systems
Prof. Soumyajit Dey
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 14
Real Time Task Scheduling for CPS (Continued)

Hello and welcome back to this lecture series on Foundations of Cyber Physical Systems. So, we will just start right over from where we left off. So, in the last lecture we have been discussing about this EDF algorithm and we showed that well it is optimal among all such dynamic priority algorithms. However, we also said that well some of these algorithms do not, these algorithms that we have discussed till now, they do not support this kind of precedence relations among tasks.

So, let us see what kind of algorithms can be used to actually execute tasks which have this kind of precedence constraints and we will start with one which is known as Latest Deadline First or LDF.

(Refer Slide Time: 01:14)

The slide is titled "Latest Deadline First (LDF)". It features a task graph diagram with four nodes labeled T1, T2, T3, and T4. T1 is at the top left, T2 is at the top right, T3 is at the bottom left, and T4 is at the bottom right. Arrows indicate dependencies: T1 to T2, T1 to T3, T2 to T4, and T3 to T4. A pink circle highlights T4, and a pink arrow points to it from the text "The last task to execute is the one on which no other task depends that has the latest deadline".

- ▶ Optimal with precedences
- ▶ Constructs the schedule backwards, choosing first the last task to execute
- ▶ The last task to execute is the one on which no other task depends that has the latest deadline
- ▶ Does not support arrival of tasks

Foundations of Cyber Physical Systems | Soumyajit Dey, Associate Professor, CSE, IIT Kharagpur

So, if we can just look at these slides, LDF is an algorithm which is optimal with respect to precedence relations. And what essentially it does is you are given a task graph, a task graph is kind of where you have the nodes as the tasks and the edges, forward going edges, they are

actually capturing the precedence constraints that which task must execute and after that which does can execute something like that.

So, an LDF algorithm will analyze such a graph and it will construct the schedule backward. It will first choose the last tasks, the tasks that are the leaf nodes of such a tree. We will see that. And the reason why it is done like that is that the last task does not really have anybody dependent on it. So, let us say you have a task graph like this that only after what this graph says is only after T_1 executes, I can execute T_2 and T_3 .

And T_2 and T_3 do not have any dependency and let us say after both T_2 and T_3 executes, I can execute T_4 . So, these kind of constraints are taken as input by this LDF algorithm. And it will construct the schedule backwards, which again means that it will first try to compute what should be the start time of this task T_4 . It does not have anybody who is going to depend on T_4 , right.

So, it will just figure out the start time of the independent task which is right at the leaf node or the end. And then it will construct backward. It will analyze tasks which are kind of predecessors of T_4 like T_2 and T_3 and then it will again analyze and figure out what should be the start time of task which is the precedence, which is the predecessor of T_2 and T_3 . So, that is how it works.

Now in this variant, we are not supporting arrival of tasks and we will soon see that will how that support can also be brought in.

(Refer Slide Time: 03:21)

Overview Processor Level Task Scheduling scheduling CAN Bus Level WCRT Analysis

EDF with Precedences (EDF*)

- ▶ Supports arrivals, precedences and minimizes the maximal lateness
- ▶ Adjust the deadlines of all the tasks
- ▶ Suppose the set of all tasks is T
- ▶ For a task execution $i \in T$, let $D(i) \subset T$ be the set of task executions that immediately depend on i in the precedence graph
- ▶ For all executions $i \in T$, modified deadline is defined as,

$$d'_i = \min(d_i, \min_{j \in D(i)} (d'_j - e_j))$$
- ▶ EDF* is then just like EDF with modified deadlines

Foundations of Cyber Physical Systems Soumyajit Dey, Associate Professor, CSE, IIT Kharagpur

So, when we modify LDF with this support for arrival of tasks, that is what we call as EDF with precedences or EDF*. So, this is the algorithm which handles precedences, arrival of tasks and minimizes the maximum lateness. So, that means at any point of time, given a set of tasks, including the ones which have arrived, you figure out which task to execute, okay, and again you recompute that upon arrival of new tasks to your task set, so that is how it works.

So, what it does is it will recompute the relative deadline of the tasks. So, you have a set of tasks which are pending to execute and some of them have arrived now some of them may have arrived a bit earlier. So, what you are doing is you are recomputing the deadlines of the tasks with respect to their precedence constraints. So, suppose right now your set of tasks is T , so for a task to execute, let us say task i will execute, okay.

Let d_i be the set of task executions that immediately depend on i in the precedence graph. So, that means once task i executes d_i of these tasks can execute, okay. So, what we are saying is we will recompute the deadline. That means whatever is given to you let us say that well this task must execute inside this time, is not really the true deadline. Because, if I just schedule the task based on it is deadline, any task which is going to depend on this task, it can only execute once, this guy has finished, right. And that may be quite late for those dependent tasks to satisfy their own absolute deadlines, we will see what it means. First, let us just look at this symbolic equation and in with soon with an example we will see that will how such a symbolic relation

can be computed. So, what we are saying is suppose let us say this is your task i and d_i is containing 2 tasks j and k , right.

And you have somehow computed that exactly when j and k must start, right. And we want to see that well what should be the modified deadline of the task i or what do I mean is i have already computed these modified deadlines of j and k , okay. So, that is more like a relation here we are we are trying to develop between these modified deadlines of i , j and k suppose, these are known, right.

So, with respect to these modified deadlines, that means the deadlines inside which j really must execute. So, d_j let us say is the original deadline of task j , but we are saying that this modified deadline is indicating the time inside which j must really execute, so that anything downstream after j which depends on j will have enough time to execute and satisfy their own deadline.

So, suppose these downstream activities we have already figured. Let us assume that. So, now with that being figured out this modified deadline for j is d_j' , similarly for k is d_k' . And we want to compute this d . That when really, I must finish it is execution. So, the way to do it is well, if this is its real deadline then it must start at some time which is $d_j' - e_j$ and similarly $d_k' - e_k$, right.

So, among these two values, which one is the minimum, that we will figure out, okay. Because, if we can figure out that, that among these two values which one, I mean, among these two values this deadline minus execution time, if we can figure that thing out. That tells me that well, that much time must be, I mean, with respect to d_j' , I can leave out e_j and I can say that that is the deadline of i or with respect to d_k' , I can leave out an e_k amount of time.

And I can see that well the rest of whatever remains that is the real deadline of I, okay. So, I compute those differences and see who among them is minimum, right. So, whichever is smaller is going to be now compared with the real deadline of this ith task. That is another minimization. And that will tell me that well that is when this ith task must finish its execution, because after that, let us say it happens to be the case that d_k' is smaller, I mean, $d_k' - e_k$ is smaller, that means and let us say that value is even smaller than d_i .

That means for $d_i' = d_k' - e_k$. So, let us say at this point this ith task will actually finish. That means I am just leaving enough time to finish d_k inside its deadline d_k' , got it. So, that is how we compute these modified deadlines and we do it from the leaf nodes of the tree backward in this precedence graph, okay. So, essentially once these deadlines are computed, the rest of it is just EDF.

So, what we are doing is we are reevaluating the deadlines by satisfying the precedence constraints of the graph. And then based on the deadlines whichever is earlier, we will dispatch to the processor and that is how we will work it out here.

(Refer Slide Time: 09:13)

The slide displays a slide titled "An Example" with two main sections: "Precedence Graph" and "Schedule".

Precedence Graph: A directed graph with nodes 1, 2, 3, 4, 5. Node 1 is the root. Node 2 is a child of 1. Node 3 is a child of 2. Node 4 is a child of 3. Node 5 is a child of 3. Deadlines are given as $d_1=2$, $d_2=5$, $d_3=5$, $d_4=3$, $d_5=5$. Handwritten notes show the calculation of modified deadlines d_i' for each node:

- $d_1' = \min(2, \min(5-1, 5-1)) = 1$
- $d_2' = \min(5, \min(3-1, 5-1)) = 3$
- $d_3' = \min(5, 5) = 5$
- $d_4' = 3$
- $d_5' = 5$

Schedule: A Gantt chart showing the execution of tasks 1 through 6 on a processor. The execution time for all tasks is 1 unit. The tasks are scheduled in the order: 1, 3, 2, 4, 5, 6. The timeline is marked from 0 to 6. Handwritten notes indicate that the tasks are dispatched based on their modified deadlines.

So, if you see here, we have taken one example. So, you have these values that are given, okay. So, you have a precedence constraint graph. So, d_1 is some 2, d_2 for this one is 5. This is also the example has been taken from the same book that we have been talking about in this course.

That is a kind of a reference test do at least up to this much of coverage we have done. And d_3 the deadline for node 3 is 4, similarly deadline for node 6 is 6.

I mean, suppose these are given, but mind that these are the actual deadlines. So, if we just now try and work out the relative deadlines backward. So, these are the leaf nodes here, so their deadlines will remain same. So, d_4 will be 3, d_5 will be 5, d_6 will be 6, no problem with that, okay. What about d_2 and d_3 . Let us figure that out. So, let us pick up this d_2 here, I mean and try to see what should be its modified value.

So, if we apply. We'll just apply this relation, just note it once again. So, for 2, we have two successors, 4 and 5, right. So, you are going to compare here with d_2 , and then you are going to do a $\min(d'_4 - e_4, d'_5 - e_5)$, that is it right. And $d'_4 = d_4 = 3$, $d'_5 = d_5 = 5$. So, d_2 is 5. So that makes this one to be 2. And similarly, if we just do this for d_3 , value of d_3 is 4. It has got only one successor which is d_6 , right.

So, that means you do not need these minima here. This is just a 6, $d_6 - e_6$. By the way, all the execution times here e , those values are 1, so that will come out as 4. So, then this 6 will get to be 4, okay. So, now you have this set will deadlines d_1 and d_2 both as 2. And so not d_6 , this is d_3 which remains to be 4. So that is there is no much change here because also anyway it does not hurt because d_6 is 6, so there is quite lot of time, $6 - 1$, 5.

So, whatever was it, I mean, that remains but here you can see this guy, the deadline has changed significantly. One can even say that this example is kind of built to highlight this issue because somehow, we have kept this high. And kept the deadline of the latter nodes to be smaller at least this one. The idea is we are trying to show that how this guy is forcing the actual runtime deadline of these two change, okay.

So, even in your precedence constraint graph if there is some issue like some intermediate node has a higher deadline, it will automatically get pushed to the left by nodes later on, if they have a smaller deadline. So, that that is kind of what we are trying to show here. And then, if you just apply this EDF* or LDF, in this case is only for a single graph no new task is arriving, so, LDF and EDF* are same.

So, which tasks are really going to execute? so task 1 will definitely execute, right. I mean, then after this task 1, of course when a task executes it is precedence order has to be satisfied by the schedule, right. So, 1 executes and after that 2 and 3 both of them are free to execute. Now, if I have taken this LDF, what do I have? I mean, if I have taken LDF or EDF*, what is happening is this d_2 's recomputed deadline is 2 and d_3 's recomputed deadline is 4.

So, it will schedule d_2 , right. So that is why you see of course LDF and EDF* like I said has to be same, so it is it takes the second task. And after the second task which task to take? Well, now you see, once the second task is taken out of the equation, once the second task is taken out of the, this is done, this is done. Now once this is done, you have a set of 3 possible tasks to execute. So, this is like your scheduling frontier now. This is done, so these two tasks are free.

And the initial node is done, so this task is also free, right. And among them you have deadlines like 3, and this is 5, and this is 4. So, which one you execute? So, you will execute like 4 here. So, because that is the deadline 3, so you execute the node 4. Node 4 as deadline 3, node 5 deadline 5, node 3 deadline 4. So, smallest deadline value is 3 for node 4. So that is what will come, so once this is done you have now 5 and 3 node numbers with relative with deadlines 5 and for 3 it is 4, right. So, 3 will come, right.

So, once this is done and now you are scheduling frontier will also include this guy. But then again if node 5 has deadline 5 and 6 has deadline 6. So, 5 will go and then 6 will come. Had this been EDF, had there been would there have been any change? Yes of course. So, what would have happened is if it is EDF none of this re-computation would have ever happened,

right. So, it would have taken d_1 , right. Precedence constraints will force that well one has to be executed.

Then, after that these two guys are free. No re-computation of deadline. So, this is 4, and this is 5. So, node 3 will go in instead of node 2 and then, well, after that you have so node 3 has gone in and then you will have node 2, right. So, and after node 2, you would have executed 4 because I mean, after this you would have executed node 2, you would have executed 4, then 5 and 6, okay. So that is how EDF would have done it

So, that is about executing and you know, I mean, executing schedule, I mean, tasks with precedences in a deadline efficient manner. Now, we have discussed Rate Monotonic. We have discussed this new algorithm, that is EDF*. And just for you to study, there is another algorithm called Deadline Monotonic algorithm. So, you can just take this as a take home exercise and start and study about this.

We will also try to give you some problems around all these algorithms when we do the tutorials. So, that is about some real time scheduling algorithms. And the next thing that we will be discussing now is some amount of timing analysis of such multitask scheduling on processors. So, we are calling this topic as processor-level worst-case response time analysis.

(Refer Slide Time: 18:40)

Overview Processor Level Task Scheduling scheduling CAN Bus Level WCRT Analysis

Response Time Analysis in A Processor

- ▶ A task set $\{T_i\}$.
- ▶ Each task is represented as $T_i \sim \{p_i, D_i, e_i\}$.
- ▶ Each task has a fixed unique priority (DM or RT or any other scheme).
- ▶ All tasks are preemptive. For each task $T_i : D_i \leq p_i$.
- ▶ Response time of a task T_i is denoted as R_i .
- ▶ Our objective is to compute the response time R_i such that for each task $T_i : R_i \leq D_i$.

Foundations of Cyber Physical Systems Soumyajit Dey, Associate Professor, CSE, IIT Kharagpur

Now, let us try to understand what is response time. I think we already have defined this. Like you have a task that has been released and then that does not mean that well the task will be executed immediately, right. And the task will eventually get executed it may get preempted and this can keep on happening, right. And eventually when the task finishes, so you have this interval when the task started and when the task finished, right.

And when the task was released and when the task is finished and this is what we call as a response of the task. So, typically for a periodic task we will be representing them using this kind of a 3 tuple for task T_i . The tuple will contain period p_i , deadline D_i and the execution time e_i , right. Now, the tasks can have a unique fixed priority, okay. You can have a Deadline Monotonic scheme or a Rate Monotonic scheme or any other scheme like that.

All tasks are preemptive and for each task we are assuming that the deadline is less than or equal to the period. So, there is a catch here. So, if we do not mention the deadline, you can assume that deadline is equal to period. But we can mention the deadline. Of course, the deadline has to be less than or equal to the period in that case, okay. So, when I am talking about fixed priority algorithms, the priority may be decided by the rate or the period, or the priority may be decided by the deadline. That is another variant.

Now, suppose we want to figure out what is the Response Time of the task? Response Time we have just defined, right. Now, why do I want to figure out what is the response time? Because for a real time task, we require that the response time must be less than the deadline. So, when a task's period starts, we know that well unless there is any offset or something specified, that is the point where that task instance has been arrived, has been injected into the system or the task instances arrived into the system.

And from that point up to the point where the task finish, this is what we call the Response Time and in a hard real time system we want the task to finish before it is deadline, right. So, this is a constraint which is important, $\forall i R_i \leq D_i$.

(Refer Slide Time: 21:09)

Response Time Analysis in A Processor

- ▶ Response time with fixed priority preemptive scheduling for a given task set is given by,

$$R_i = e_i + \sum_{\forall j \in hp(i)} \lceil \frac{R_i}{p_j} \rceil e_j$$
- where $hp(i)$ is the set of tasks of higher priority than T_i .
- ▶ The above response time is calculated recurrently like below,

$$R_i^{n+1} = e_i + \sum_{\forall j \in hp(i)} \lceil \frac{R_i^n}{p_j} \rceil e_j$$
- ▶ The initial condition is $R_i^0 = 0$ and the terminal condition is $R_i^{n+1} = R_i^n$

Foundations of Cyber Physical Systems | Soumyajit Dey, Associate Professor, CSE, IIT Kharagpur

So, we will like to know how to compute this quantity R_i ? Now, if you are given a fixed priority preemptive scheduling scheme for a given task set, the response time for the i th task is typically given by this kind of an equation. So, for the i th task,

$$R_i = e_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{p_j} \right\rceil e_j$$

and you are carrying out this summation over all tasks who have a higher priority than this i th task, okay.

So, for all those tasks what you are doing is you are having that response time of i divided by this period. So, you see that for all task we have assumed this tuple. So, p is the period, so for all those high priority tasks for the corresponding, let us say you have picked up some j value what is the period of that task you divide it, right and then you multiply this by the execution time. So, let us understand philosophically what is happening.

So, I am saying that well right now let us say I have an approximate value that well the response time of this i th task is this. So that is one interval, right. So, inside this interval how many instances of the j th task can occur? So that can be figured out by dividing this R_i by this inter, by the period of the j th task p_j , right. And this is your R_i , this this interval is, this is R_i , okay. Now so I get that many task instances. Now as I have said, I have taken the summation only over task which have got higher priority than the i th task.

So, it is expected that all those instances of the j th task will come and they will preempt the i th task and they will consume the CPU in for each instance for this e_j amount of time. So, inside this response time i will be interrupted this, R_i/p_j number of times, and each interruption will be of an interval of size e_j , right. So those times my task maybe it was trying to execute that, T_i was trying to execute, but that high priority task j is coming. It is interrupting me and it is consuming the CPU.

So, I am just figuring out how much interference I have from this j th task. I think now it will be a bit clear. And similarly, I can figure out the interference on me, the i th task caused by all such other j values of tasks which are of higher priority than me. Because, all of them are going to interrupt me. All of them are going to interrupt me for their execution time, and that phenomenon is going to happen that many times that they come inside this response time, okay.

Now the question is well, this response time value is not known, right. So, if you see this is like a recurrence equation, what we are doing is we are getting an estimate and we are refining that estimate by applying this equation, okay. So, that is why let us say my n th estimate is given by R_i^n , I apply this equation to get the $(n + 1)$ th estimate,

$$R_i^{n+1} = e_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^n}{p_j} \right\rceil e_j$$

okay. So, initially we will assume that this value is 0. So initially I will only have e_i as my initial estimate of my response time.

And then we will compute this recurrence and we will keep on computing it, until unless I get two consecutive iterations of this recurrence giving me the same value. So, that means this computation has reached a fixed point and that is my response time value. Now notice this symbol here, so that is like the ceiling symbol because this ratio may be fractional. In case of fractional I will just try to round it up to the next whole number and that is why we take the ceiling here.


(Refer Slide Time: 25:15)

Processor level Response Time Analysis Example

For the following task specifications let us analyse the response times and find out whether the control task meets its deadline constraint.

message	p_i (ms)	D_i (ms)	e_i (ms)	Type	Priority
T_1	30	15	5	Security Task	2
T_2	20	12	8	Sensing Task	1 (highest)
T_3	30	30	12	Control Task	3

- ▶ It is a fixed priority scheduling.
- ▶ Note that, the control task has the same deadline as its periodicity.
- ▶ We need to check whether the response time of the control task T_3 , $R_3 \leq D_3$.



Foundations of Cyber Physical Systems Soumyajit Dey, Associate Professor, CSE, IIT Kharagpur

So, let us take a simple example here of response time analysis. So let us say I have CPU, an electronic control unit in a vehicle which has got some tasks this T_1 , T_2 and T_3 . So, these are not messages, these are, let us say periodic real time tasks. And they have the periods and deadlines and execution times as mentioned. Let there be, I mean, tasks have different types one is the security monitoring task.

That means it will execute once every 30 millisecond and it will try to finish its execution inside 15 millisecond. And each time it executes it will consume 5 millisecond of the CPU and it is monitoring what activities are going on where, whether some values or is monitoring are wrong and then it will raise an alarm something like that. It has got a priority 2, and then there is a sensing task which is also running periodically.

There is a control task which is running periodically and the sensing task has the highest priority. Let us just consider this as a very symbolic example, okay. So, these are fixed priority scheduling we are considering. So, note that, because that, their priorities are I mean with respect to the periods here. Now note that the control task has the same deadline as its periodicity, this one, okay.

Now we need to check whether for this control task the deadline, I mean, is greater than or equal to the Response Time value.

(Refer Slide Time: 27:00)

Overview
Processor Level Task Scheduling
Scheduling
CAN Bus Level WCRT Analysis

Processor level Response Time Analysis Example

message	p_i (ms)	D_i (ms)	e_i (ms)	Type	Priority
T_1	30	15	5	Security Task	2
T_2	20	12	8	Sensing Task	1 (highest)
T_3	30	30	12	Control Task	3

Response time for T_1

$R_1^0 = 0$

$R_1^1 = e_1 + \left\lceil \frac{R_1^0}{p_2} \right\rceil e_2 = 5$

$R_1^2 = e_1 + \left\lceil \frac{R_1^1}{p_2} \right\rceil e_2 = 5 + 8 = 13$

$R_1^3 = e_1 + \left\lceil \frac{R_1^2}{p_2} \right\rceil e_2 = 5 + 8 = 13$

Hence, $R_1 = 13 \leq D_1 (= 15)$

Foundations of Cyber Physical Systems
Soumyajit Dey, Associate Professor, CSE, IIT Kharagpur

So, if you just look into this calculation, so just like we said, the initial estimate is 0. Now you apply this equation once, okay. So, we are computing it for the control task. So, that means this has a priority 3, so it can be interrupted by which tasks? It can be interrupted by this sensing task and the security task, right. So, if you see here. So, our target is to compute for this control task, right. But let us start with this one first. We are trying to see for this security task, okay. So R_1 .

Now, R_1 has priority 2. So, the only one of higher priority than that is T_2 , okay. So, it can be interfered with by this T_2 . So, when I factor in, I will just factor in the execution, let us say this is the response time estimate I will of R_1 . I will divide it by the period of T_2 and I will multiply this number of instances with execution of it T_2 that is e_2 , right. So, initially this is 0, so all I get is e_1 that is 5 and then I will refine it to 13, and then I will soon see that well it is stabilizing at 13 or not changing. So, for T_1 it is getting satisfied.

(Refer Slide Time: 28:38)

Processor level Response Time Analysis Example

message	p_i (ms)	D_i (ms)	e_i (ms)	Type	Priority
T_1	30	15	5	Security Task	2
T_2	20	12	8	Sensing Task	1 (highest)
T_3	30	30	12	Control Task	3

Response time for T_2

$$R_2^0 = 0$$

$$R_2^1 = e_2 = 8$$

$$R_2^2 = e_2 = 8$$

Hence, $R_2 = 8 \leq D_2 (= 12)$



Now, let us see for so this is for T_1 . Now let us see for the sensing task is the highest priority, so there is absolutely no interference. So, this kind of ceiling-based terms will never come. So, it is all about the execution time only. So, it is just a I mean response time is exactly equal to it is execution time whenever it comes it gets the CPU this fixed priority scheduling. So, there is no issue at all. Period, I mean, deadline is 12 sorry.

Here this should be 12, not 20, and 8 is always, I mean, this execution time that which is equal to the response time is always less than the deadline here no problem at all with that.

(Refer Slide Time: 29:30)

Processor level Response Time Analysis Example

message	p_i (ms)	D_i (ms)	e_i (ms)	Type	Priority
T_1	30	15	5	Security Task	2
T_2	20	12	8	Sensing Task	1 (highest)
T_3	30	30	12	Control Task	3

Response time for T_3

$$R_3^0 = 0$$

$$R_3^1 = e_3 + \frac{R_3^0}{P_2} \cdot e_2 + \frac{R_3^0}{P_1} \cdot e_1 = e_3 + 2$$

$$R_3^2 = e_3 + \frac{R_3^1}{P_2} \cdot e_2 + \frac{R_3^1}{P_1} \cdot e_1 = 12 + 8 + \frac{12 \cdot 5}{30} = 25$$



Now what about T_3 , right? With periodical deadline is about 30 and we want the response time to be less than that. So, let us see. So, for T_3 , if we just try to compute that value. So, the 0th

estimate is of course, I mean, we can either write this or we can just do the other that means interference of the second task, interference of the first task, but of course I have taken this to be 0.

So, this is coming just is e_3 , or you can simply take R_3^1 as e_3 . You can just start writing from there. So, that is not a problem. So, this is actually 12 and then what about the second, the refining the estimate. You can just write the equation. So, this is 8. This will come as let us see. So, this is 5 and similarly you can get this as 8 and you will get this as 25. So you can now start with 25.

(Refer Slide Time: 31:35)

The slide displays a table with the following data:

message	p_i (ms)	D_i (ms)	e_i (ms)	Type	Priority
T_1	30	15	5	Security Task	2
T_2	20	12	8	Sensing Task	1 (highest)
T_3	30	30	12	Control Task	3

Below the table, the text 'Response time for T_3 ' is written. Handwritten calculations in pink ink show the iterative process of finding the response time R_3 :

$$R_3^3 = 12 + \left\lceil \frac{25}{20} \right\rceil \cdot 8 + \left\lceil \frac{25}{30} \right\rceil \cdot 5 = 12 + 2 \cdot 8 + 5 = 33$$

$$R_3^4 = 12 + \left\lceil \frac{33}{20} \right\rceil \cdot 8 + \left\lceil \frac{33}{30} \right\rceil \cdot 5 = 12 + 16 + 10 = 38$$

$$R_3^5 = 12 + \left\lceil \frac{38}{20} \right\rceil \cdot 8 + \left\lceil \frac{38}{30} \right\rceil \cdot 5 = 12 + 16 + 10 = 38 = R_3^4$$

The final result, 38, is circled in pink. A small video inset of the presenter is visible in the bottom right corner of the slide.

So, again you have 12. So, these are ceiling relations. So, you get 2 and here you get a 5. So that is 33, okay. And if you keep on doing this, you can go to the fourth estimate, which should again be 12 plus you can see this is 33, right. So, the previous estimate is 33. If you keep on doing this, so this is again going to give you, I mean, it is rounded up to the next whole number so that is 2 and 16. This is again rounded up to the next number, right.

So, you get 38 and then you can just check that if you get the 5th estimate. I am just trying to show you that this really is going to terminate. So, you see it has because, again you are you are getting the same values because it is 10 and this 38 which is equal to your previous estimate, so you can now stop. So, what you get now is the response time for task 3 and that comes to

be 38 which if you compare with the deadline is kind of is going to overshoot the deadline, right.

So, that is a bad thing so if we are doing this kind of Rate Monotonic scheduling and if you do the response time analysis this is clearly violating the deadline. So, that is it, that is how we go about doing processor scheduling and, I mean, computing response time response time for task sets with fixed priority. So, with this we will end today's lecture. Thank you for your attention.