

Foundation of Cyber Physical Systems

Prof. Soumyajit Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Lecture - 12

Real Time Task Scheduling for CPS (Continued)

Hello and welcome back to this lecture series on Foundations of Cyber Physical Systems. So, I believe in the last lecture we have been talking about Rate Monotonic Scheduling and we were analyzing this nice example here.

(Refer Slide Time: 00:40)



Rate Monotonic Scheduling (RMS)

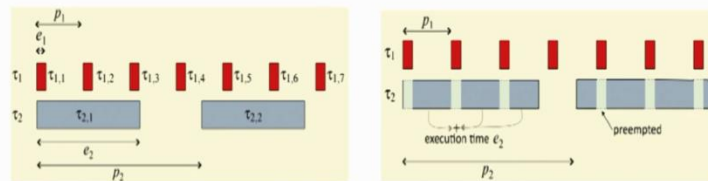


Figure: No Nonpreemptive schedule is feasible

Figure: Preemptive schedule giving priority to τ_1

Note that, Preemptive schedule giving priority to τ_2 is not feasible²

²Ref: "Introduction To Embedded Systems: A Cyber-Physical Systems Approach"- Edward Ashford Lee, Sanjit Arunkumar Seshia



So, we were trying to show that well if you look at these task instances, like we have with two tasks τ_1 and τ_2 . If we consider that the schedule is going to be non-preemptive, it will not be feasible because the tasks will start missing their deadlines and then we showed an example schedule where I am giving priority to this task τ_1 here. So, and when I am going to give priority to τ_1 , we have a physical, feasible schedule.

And in case we try the alternate option that is we start we try to give priority to the task τ_2 . We also discuss that why it should not be feasible because in case I am giving higher priority to τ_2 , τ_1 's instances will start missing the deadline. So, this is from where we left it off.

(Refer Slide Time: 01:44)

Rate Monotonic Scheduling (RMS)

- Among all preemptive fixed priority schedulers, RM is optimal with respect to feasibility, under the assumed task model with negligible context switch time.

Figure: The non-RM Schedule gives higher priority to τ_2 . It is feasible if and only if $e_1 + e_2 \leq p_1$ for this scenario.

Now let us just go forward with further discussion on this topic. So, this idea that we are considering the priority, I mean, considering to prioritize tasks based on their period, that means if the smaller the period you give higher priority to the task, this idea was called a Rate Monitoring Scheduling that we discussed. So, the higher the rate I mean you give higher priority. So, the priority is monotonicities with respect to the rate of the task, okay.

And we also kind of hypothesize that well among all such fixed priority schedulers, that means you are going to you are not going to change the priority of the tasks, okay, and as long as they are preemptive in this class of all possible scheduling algorithms, RM is going to be optimal with respect to feasibility. That means as long as some other non-RM schedule gives an algorithm gives the schedule for a given task set, RM will also be able to give a schedule.

And this is optimal also considering that the context switch time is negligible. That means you do not need to account for any time that is spent in preempting one task and bringing another task to the CPU, we are considering that those are kind of happening spontaneously. If you look at this

example, we are also saying that what is, this is the two-task scenario. So, the, I mean, when you are trying to give priorities to two tasks you have only two options whether to give higher priority τ_1 or higher priority τ_2 .

So, if I consider the non-Rate Monotonic schedule which will definitely now give higher priority to τ_1 τ_2 , the question is when does this non-RM schedule also becomes feasible? So, the condition will be very simple. So, if you see, this is where τ_1 's first instance has come, right. So, let me just mark it out. So, this is where τ_1 's let us call it τ_1^1 that has arrived and it should be executed somewhere between inside this interval, right.

And then this is exactly where τ_1 's second instance comes and it should be in executed inside this interval, right. Now this is exactly where τ_2 's first instances arrived and it must be executed inside this period of τ_2 , right. This is the period. Now the question is if the non-RM schedule that means has to be feasible that means I want to execute τ_2 's instance before τ_1 's instance. Well, let us put τ_2 here so τ_2 will eat up e_2 amount of time of the CPU, right. e_2 is τ_2 's execution time.

And only then the CPU is free to actually execute τ_1 because we are we are giving τ_1 less priority, right. So, now τ_1 will take this e_1 amount of time and now note that this is where I have the deadline of τ_1 , right, because after this $p_1 = d_1$, the deadline, I have τ_1 second instance coming, right. So, only as long as I have this $e_1 + e_2$ to be less than or equal to p_1 and this is a feasible schedule.

It is clear to see that, well if this constraint is not violated, that means, let us say $e_1 + e_2$ is greater than p_1 , then τ_1 's execution will not end here inside e_1 , it will spill over on the other side and that will lead to a deadline miss for τ_1 , right. So, that is why this is feasible if and only if this $e_1 + e_2 \leq p_1$ holds.

(Refer Slide Time: 05:32)

Overview Processor Level Task Scheduling Scheduling CAN Bus Level WCRT Analysis

Rate Monotonic Scheduling (RMS)

Figure: The non-RM Schedule gives higher priority to τ_1 . For the RM schedule to be feasible, it is sufficient but not necessary, for $e_1 + e_2 \leq p_1$

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

So, for the non-RM schedule this is an absolutely necessary it is an if and only if kind of condition. But what of the RM schedule? So, for RM schedule that is not a necessary condition. That means even if $e_1 + e_2 \leq p_1$, this is not satisfied, still RM can give me a satisfiable answer. I mean why because well in our in case of RM τ_1 will execute first, right. So, we will have kind of τ_1 finishing up right here, right.

And then I do not need this to be satisfied. The reason being well τ_2 will be starting at this point, right. Although it arrived at this point it is starting right at this point and as you can see, I mean, it is taking this much amount of time. But suppose even if it does not finish here, no problem. Let us preempt it with τ_1 's next instance because that has come. And this is RM so I am giving more time here. So, when this one finishes whatever is left of τ_2 that can work it up here, right.

So, as long as τ_2 is small enough to be split and executed in this gaps and τ_2 's execution ends before this point of time, right. It can even execute a bit of it here, right. So, then we are fine, right, because by this time τ_1 's 1, 2, 3, 4, right. These four instances of τ_1 here and then this and then this and then these four instances of τ_1 have executed. And inside this interval p_2 , I need to execute only one instance of τ_2 , right.

So, as long as these holes that are left, they are good enough to fill in one full instance of τ_2 maybe in a split manner because τ_2 is getting continuously preempted, and then we are fine. So, this is not required. So, it is not necessary but if it is satisfied then it is, I mean, its anywhere going to be schedulable. So, this is sufficient that is why we say please observe the wording here in case $e_1 + e_2 \leq p_1$, it is sufficient for a sufficient condition for making RM schedule to be feasible, but it is also not necessary.

That means even if this is not satisfied, still as long as I, the scenario that I just described, that occurs, that there are enough holes inside p_2 , the timeline p_2 , to fit in due to τ_2 's one instance, it is okay as per RM scheduling. So, that is that is the point we are trying to make here.

(Refer Slide Time: 08:23)

Rate Monotonic Scheduling (RMS)

- ▶ Given a preemptive, fixed priority scheduler and a finite set of repeating tasks $T = \{\tau_1, \tau_2, \dots, \tau_n\}$ with associated periods p_1, p_2, \dots, p_n and no precedence constraints, if any priority assignment yields a feasible schedule, then the rate monotonic priority assignment yields a feasible schedule.
- ▶ Implementation : use timer interrupts
- ▶ Utilization

$$\mu = \sum_{i=1}^n \frac{e_i}{p_i}, \quad \mu \leq n(2^{1/n} - 1)$$

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

So, in general this is what we can say that you are given a preemptive fixed priority scheduler and you are given a finite set of repeating tasks. So, this is your task set. Ideally, I would like to have these brackets here. So, this is a set of n tasks and you have their associated periods p_1 to p_n and let us say the tasks are independent, that mean there is no dependency. We do not have this kind of thing that only when τ_1 ends τ_2 can start or only when τ_2 and τ_3 ends only then τ_4 can start.

As long as we do not have this kind of period is a, I mean, precedence constraints here. So, this is what they are called precedence constraints or dependencies, right. As long as that is not the case,

then like we said that RMS is, I mean, is optimal with respect to feasibility, which means in such case when I have a set of independent tasks and associated periods, as long as any, I mean, any priority assignment to the tasks is going to give me a feasible schedule, I have a guarantee that Rate Monotonic priority assignment is also going to give me a feasible schedule. So, that is why it is optimal with respect to feasibility. Now how is this implemented? We have made this point earlier also, we have told that well Rate Monotonic scheduling is very easy to implement. The idea would be simple, that well, you have you can have a dispatcher which will schedule τ_1 which will give τ_1 to the CPU, and also, it will set up a timer interrupt, right.

Because it knows exactly when τ_1 's next instance needs to be executed, right. And whenever some other task comes let us say τ_2 , τ_3 they are coming, right, whenever they arrive it is just going to wake up it is going to check that well as long as the highest priority task is running, its completely uninterrupted. But for every other task there would be an interrupt that is set through let us say a watchdog timer or similar setups. An interact would be set in the CPU, right.

So, whenever, I mean, it is going to be implemented through a timer kind of interrupt. So, whenever the timer expires what does that mean? The timer expires you will get a hardware interrupt, right, the dispatcher will wake up. Now it has to check that well through its interrupt service routine it is going to check that well this timer has expired for which task, right? And that you can know by looking at the interrupt value, I mean, it is basically the interrupt is with respect to which task, right?

And it is just going to see that well what task is executing. If a task of a higher priority is executing just do not bother, right, and just let it just execute. And otherwise, it is if not so then, if there is no higher priority as executing and whatever is executing is our lower priority, preempt it, save its context, I mean, save whatever is the register state of the processor. I mean in that tasks contest and make this task which is of higher priority execute.

Because the interrupt has come, means the corresponding timer has expired. That means a new instance of this task is going to execute. Now one thing I will like to make clear whenever I am saying, whenever we keep on saying that well one new instance of task arrives, I mean, typically what does it mean? In most cases what it would mean is well, I mean, the task does not really arrive, right. It is a program in execution.

So, you have the same executable executing with periodicity. What really is going to change is that this let us say this is a sensing task. There are sensed values that are going to arrive at some periodicity based on the physics of sensors and their implementation we discussed earlier and the same code is going to execute again and again with this different input values and that is what is the different task instances, right.

So, for the first sensor input you have task 1's instance 1. For the second set of sensor input, you have task 1's instance 2 and that is how it will work. So, in most systems this is how I mean this is what we mean by a task arrives basically its inputs have arrived through a sensing system. So, well going forward. This is an important thing important calculation we have. So, when you have this set of tasks executing, okay, at τ_1 to τ_n with period p_1 to p_n , and we are going to allocate the CPU based on some algorithm like rate monitoring scheduling. Well, what is the utilization of the CPU that how efficiently is the CPU really used? So, this is actually calculated by this metric,

$$\mu = \sum_{i=1}^n \frac{e_i}{p_i}$$

As you can see that is you just sum up well for each task it has an arrival interval of p_i , right, and inside this interval the amount of time it must get access to the CPU is e_i , right.

So, this e_i/p_i fraction is basically this tasks time quant to be taken from the overall CPU's time quanta, right. And in this way each task will consume some bandwidth of the CPU and the total bandwidth consumed in this way is what we call as the utilization. Now in RM, in the case of RM

schedules, it can be proved that this utilization is always upper bounded. So, let us understand one thing. You will always like to have a scheduling algorithm which ensures that you use the CPU's bandwidth fully, right.

When, do what do I mean by using the bandwidth fully? Because well the bandwidth is a maximum available limit, right. So, let us say ah this is CPU time I am plotting. Right now, you execute some task, then again you start another task, then again you start another task, like that. So, apart from let us say this position, the CPU is always occupied. Let us say now right now there is a bit of empty burst and then again, another task occupies the CPU, right.

So, the amount so utilization is kind of a measure which tells me that am I really occupy making, am I really engaging the CPU, am I really using it as much as possible, right. Ideally, I will like to have utilization to be almost near to one, that means, I am always able to efficiently utilize the CPU, right. The same concept of utilization will also utilization as well as scheduling will also apply to communication on bus, like we have discussed earlier.

Scheduling can happen inside a processor, computation scheduling. Scheduling also happens on a bus for communication scheduling, if the bus is the resource and the processor is the resource. The consumer is some process or the consumer is some message on the bus. Now it can be shown that well in case of RM, your utilization is always going to be much below 1. 1 is the idealization I would like to approach, right.


And as you can see from this formula with n increasing, this value is going to be smaller, right. So, well that I mean that is the thing RM has given you this upper bound, it can be proved actually and the proofs are there in the papers on real time Rate Monitoring Scheduling. We are keeping it out of scope here but anybody interested you can always study or discuss with me. So, this is like a bound on Rate Monotonic Scheduling. But the question is well can we do better than this?

(Refer Slide Time: 16:30)

Overview Processor Level Task Scheduling scheduling CAN Bus Level WCRT Analysis

How to get better utilization ?

- ▶ We relax the fixed priority constraint and show that dynamic priority schedulers can do better than fixed priority schedulers
- ▶ Given a finite set of non-repeating tasks with deadlines and no precedence constraints, a simple scheduling algorithm is earliest due date (EDD), also known as Jackson'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 no precedence constraints, an EDD schedule is optimal in the sense that it minimizes the maximum lateness, compared to all other possible orderings of the tasks.



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

How do I get a better utilization? Is it really possible? Well, so the, as you can see that this bound holds for Rate Monotonic Scheduling. So, that means if I can design a more, I would say design a smarter scheduling algorithm, it may definitely be possible to have a better utilization. So, let us try and understand how the scheduling scheme that we discussed can be made smarter. What are the constraints that we have in the current scheme?

If you observe the current scheme, one issue that is there, is you are always assigning tasks some fixed priority. You are as you are saying that τ_1 has always the highest priority, higher priority than τ_2 , or let us say you are saying τ_2 always has a higher priority than τ_3 . And when I say always, I mean for all instances of τ_1 , or I mean for all instances of τ_2 . Now let us understand that this may be a bit of what should I say it is the bit of I mean a conservative approach.

Why? Because certain instances of certain tasks may need to be handled in a more urgent manner. Because maybe it is deadline is approaching fast, okay. So, let us say some task, has arrived and then well immediately it got the CPU, right. But then again let us say some task, arrived it got the CPU, but again it got preempted by another task, again it got the CPU, again it got preempted by another task. So, with time what is happening is this task it still has some amount of stuff to execute but its deadline is slowly getting nearby, right.

So, a smart scheduling algorithm will like to prioritize this task based on how much of it is left to execute with respect to its deadline, okay. So, when I am talked about deadline earlier, we said that well this is the task instance and this is its deadline and after that there is a next task instance. But if you think that well the task can get preempted or the task may be the task never actually got the CPU, it has been waiting and waiting and now its deadline is almost nearby, right.

So, with time I should have an option, that well, with time a task deadline can a task's priority can increase, okay. Or similarly it can decrease in some other cases, okay. So, this is one issue with Rate Monotonic Scheduling. That well, if I give fixed priority, then I am really not modelling the dynamic scenario that well this task instance of some specific task it is waiting for how much time? This dynamic scenario is not really modelled there.

So, the alternative and smarter option can be to use dynamic priority schedulers. So, what does dynamic priority mean? That well the priority of the task can vary across its instances. So, we will see some examples. So, suppose I am given a set of non-repeating tasks, and they have their deadlines, and there is no precedence constraint, then an alternative algorithm will be, which actually takes care of this kind of dynamic priority, is called Earliest Due Date (EDD), or also known as, its known as the Jackson's algorithm.

So, we are trying to design. But you may be wondering why we are talking about non-repeating task? Well, EDD's original design was based on that. But we will soon see that in for all practical purposes the real algorithm we use for repeating task is just a variant of EDD. So, suppose you are given this kind of finite set of non-repeating tasks and they have their deadlines and there is no precedence constraints, then all that the EDD algorithm does is it will try to minimize the maximum lateness and what it will do is it will simply see which task has the earliest deadline, okay. So, whichever as the earliest deadline or earliest due date as per the name it will try to give it the highest priority, okay. Now it can be shown that in among all such algorithms, okay, this one is optimal. That means it is optimal in the sense that it minimizes the maximum lateness.

So, suppose I am given this τ_1, τ_2, τ_n this sequence of tasks, I mean, this collection of tasks. You can, you can schedule them in various possible ways, right. A schedule is nothing but an ordering of the execution of these tasks. What we are claiming is that the EDD schedule will minimize the maximum lateness. I hope you will remember what is the maximum lateness. So, maximum lateness means among these tasks set that the, I mean, you have tasks with some start times and you have task with their corresponding finishing times.

So, you have to take the, I mean, for each task you can calculate their lateness values and finally among those you can see what is the maximum lateness value, right. Now this is the algorithm which will minimize the maximized lateness, and that it is optimal in with respect to this metric. And that means if I kind of arrange these tasks in any other possible order, and also in this order, it is in this order that the maximum lateness will be minimized.

(Refer Slide Time: 22:04)

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 $d_j < d_i$

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

So, let us try to understand this claim with respect to a simple two task system first. So, suppose you have these two tasks, this task i and task j. So, these examples have all been taken from that book by Lee and Seshia only, okay. So, what we will do is we will try to create two schedules. One is a non-EDD schedule and other is the state EDD schedule. So, if you see in this example of schedule 1, so you have task i and that has a finishing time f_i , right.

And we are saying that after task i, we are starting the task j, right, and that has a finishing time f_j . But what we are looking at is the, if you see the deadlines of the tasks, j's execution was more urgent because the deadline is earlier and this i's deadline is later on. But these are non-EDD schedule, that is why i is preceding the j'th task, okay. Now, I mean, if I take the EDD schedule, it will be the reverse of this, because as per the earliest deadline j will execute first and i will execute later on, right.

So, let us compute the maximum lateness in both cases. So, maximum lateness for both is what? In, I mean, in the first instance, I mean, I can always write this equation,

$$L_{max} = \max(f_i - d_i, f_j - d_j)$$

that $f_i - d_i$ is the lateness of task i, similarly $f_j - d_j$ is the lateness of task j. So, the max of this is the maximum lateness of this two-task system. Now if you see in this case well what is $f_i - d_i$, it is this much and what is $f_j - d_j$, it is this much. And well both are kind of negative values, right.

So, if you are going to take the max it should be $f_j - d_j$. Mind both are the negative values. So, the one with the smaller magnitude would be the maximum one here, right. So, now in this case this is of course happening because of this constants $f_i \leq f_j$, right. It is earlier and d_j is also earlier than d_i , right. So, this difference magnitude is smaller. So with the sign, I mean, with the sign this is the maximum, right.

Now if you look at the other case what can we say? So, here you have $f_i' - d_i$ and $f_j' - d_j$, right. So, what can we write here? So, this is your f_j' and d_j , right, and you have this f_i' and d_i , right. So, if you look at this scenario, we are claiming that well again in this case, we can say that the difference should be, well you see, in both cases that I mean the value should be same that is $f_j - d_j$, right. Question is why?

The reason is well, what we are reversing is the task set, right. But if you look at what is f_i' , f_i' is nothing but it is equal to f_j , why? Because this is the finishing term, finishing time of the second task, that is f_j in the first case, and f_i' is the finishing time of the task in the second case, right. So, what you really have here, is this f_i' is basically equal to f_j . Why? Because definitely, I mean one task is ending and the next task is starting and only their order is changing. So, the final time when both the tasks finish that has to be the same, right. That is why we can always claim that will this is it. So, from this point, we will argue this further. So, we will finish this lecture here and continue again from this point. Thank you.