Real-Time Systems

Prof. Dr. Rajib Mall

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Module No. # 01

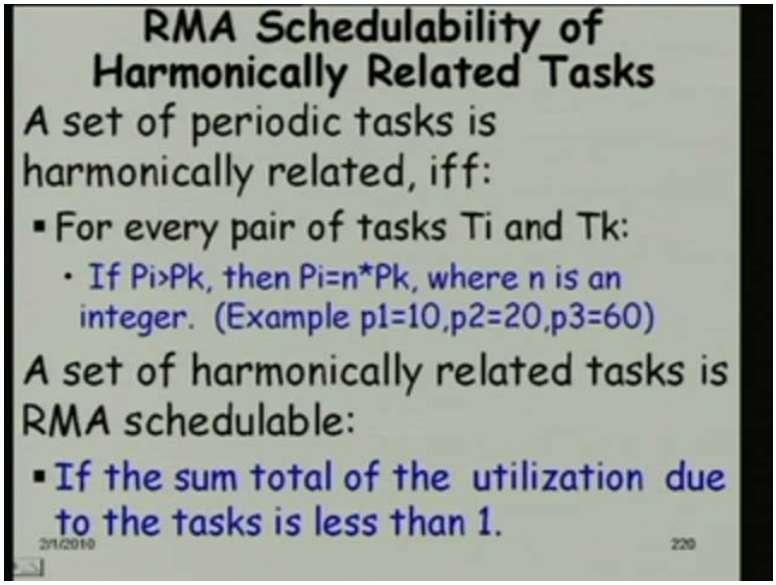
Lecture No. # 11

Deadline Monotonic Scheduling and Other Issues

Good morning, let us get started. So, so far we have discussed about the rate monotonic scheduling and the schedulability criterion; given a set of tasks, how do you know that the task set is schedulable, and then we had looked at few properties of rate monotonic algorithm, and yesterday, towards the end we were discussing about implementation of the rate monotonic scheduling.

So, today let us start from that point onwards, and towards the end of the last lecture, we are looking at harmonic harmonically related set of tasks. So, let us start from that point.

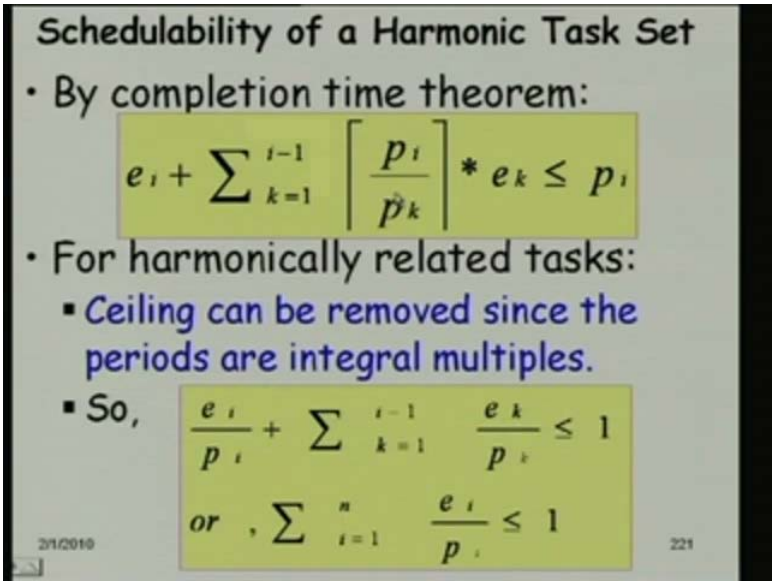
(Refer Slide time: 00:59)



So, we are saying that we call a set of periodic tasks as harmonically related, if for every task T i and T k, if as long as the period of the task T i is greater than P k, then P i will be an integral multiple of p k. So, wherever the period of a task is greater than the period of another task, then the period will be an integral multiple. So, this is a harmonically related set of tasks. The periods are 10, 20 and 60. So, 20 is a multiple of 10 and 60 is a multiple of both 20 and 10.

Now, for set of harmonically related tasks, the schedulability criterion is that the sum of utilization due to the tasks is less than 1. So, even 100 percent utilization is possible, even if there are large number of harmonically related set of tasks, it is not very hard to... So that the utilization achievable is 100 percent.

(Refer Slide Time: 02:26)



Let us just review the completion time theorem. So, we were testing the schedulability of a task set, by checking whether the expression, this expression, holds for every task t i. Now, for a harmonically related set of task, the ceiling can be removed, because whenever the period is larger than another period, it is an integral multiple, so, the ceiling is not required here, the ceiling can be removed.

So, if we remove ceiling, then we get, e i by p i from this expression, we get e i by p i plus sigma k equal to i minus 1; e k by p k is less than equal to 1. So, we have arranged the set of tasks in increasing order of that periods.

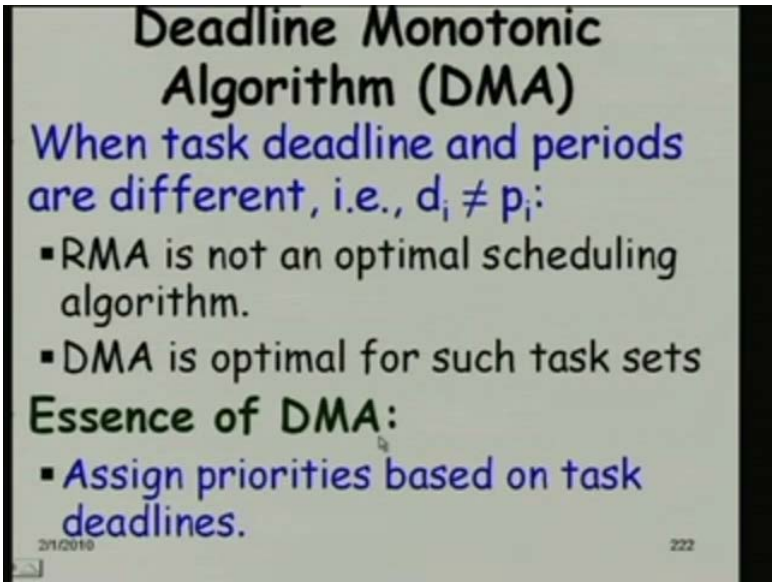
So, the first task i equal to 1 is the task which is having… k equal to 1; when k equal to 1, the task has a lowest period or the highest frequency and we are checking for the i th task. So, for the i th task, we need to check whether e i by p i plus sigma k equal to 1 to i minus 1, e k by p k is less than equal to 1 or since the ceiling is removed, we can bring this inside and we can just change the limits here. We can write i equal to 1 to n, e i by p i is less than equal to 1; does that appear ok?

Should have been k equal to 1 to i, is it not? But what you have written is i equal to 1 to n, this is for the task t n - the last task - and we do not have to check this for every task, is it not? What do you think - is this enough or is this check enough, or we need to check this expression for every task?

So, for the n minus oneth task, we need to check i equal to 1 to n minus 1, e i by p i is less than 1; for n minus 2, we have to check i equal to 1 to n minus 2, e i by p i is less than 1. What do you think - should we check for all tasks? Not necessary, because those are part of this expression, is it not? As long as this holds, all those will hold, because this utilization also includes the utilization of the later tasks, the nth task n minus 1 task and so on.

So, check here that the schedulability condition from the completion time theorem just reduces to the utilization due to all the tasks is less than 1. So, even if we have a large number of tasks, which are harmonically related, we can have even 100 percent utilization.

(Refer Slide Time: 05:51)



Now, let us look at a variation of the rate monotonic algorithm, which is the deadline monotonic algorithm. The rate monotonic algorithm, so far, we have been telling that it is the optimal scheduling algorithm, but we had made some assumptions. It is optimal scheduling as long as the period and deadline are the same, but when the task deadline and the periods are different, rate monotonic analysis algorithm is not really an optimal algorithm; the deadline monotonic algorithm is actually the optimal one.

The...As I was saying, that the deadline monotonic algorithm is based on the rate monotonic algorithm. Here, we do not assign priorities based on the task frequencies. Here, we assign priorities based on the task deadlines, that is the only difference; otherwise, rest everything is similar.

Now, let us see some properties of the deadline monotonic algorithm, have we checked schedulability and so on. And then, we will make few comments in this. So, when do the rate monotonic... See here, deadline monotonic algorithm, based on the task deadlines the priorities are allocated. Here, based on the rate or the frequency, the priorities are allocated. So, when should they produce identical schedules? What do you think?

P divided by of course. So, when p i equal to d i, they are actually the same algorithm. So, whenever the deadline is the same as the period, both are identical algorithms and produce identical schedules.

But, when the deadlines are arbitrary, it is possible that the deadline monotonic algorithm can produce a feasible schedule, when the rate monotonic algorithm fails. But, the rate monotonic algorithm will always fail, if the deadline monotonic algorithm fails. Please construct an example, where the deadline monotonic algorithm produces a feasible schedule, when the rate monotonic algorithm fails.

Is DMA static algorithm?

Of course, everything else is similar; see it is also a static algorithm. Everything else is similar to the rate monotonic algorithm, which is static priority algorithm, but only thing is the priority is assigned based on the deadline rather than the rate.

Previously, you have said that RMA is optimal static algorithm. So, when RMA fails, then all other should also fail.

No, that is what I have been trying to tell, that to start with we had told it in very simple words, that rate monotonic algorithm is optimal algorithm, what we left unsaid is that, when the deadline and the period are the same, it is the optimal algorithm; that we left it unsaid, just for simplicity, but now we are slightly making it more complicated.

We are saying that see, there will the possibility that there are some tasks, which might have their deadlines and periods different. Normally, we will see that for a large set of tasks, the deadline is the same as the period, but it is possible that for few tasks, deadline can be different from the period. So, for that rate monotonic algorithm is no more optimal.

So, here we are saying that it is possible that deadline monotonic algorithm can produce a feasible schedule, when the rate monotonic algorithm fails to provide a schedule. It is not difficult to construct an example, but please construct an example for this. We will not really wait for this one, we will just leave you as as a work; it is very simple to do it.

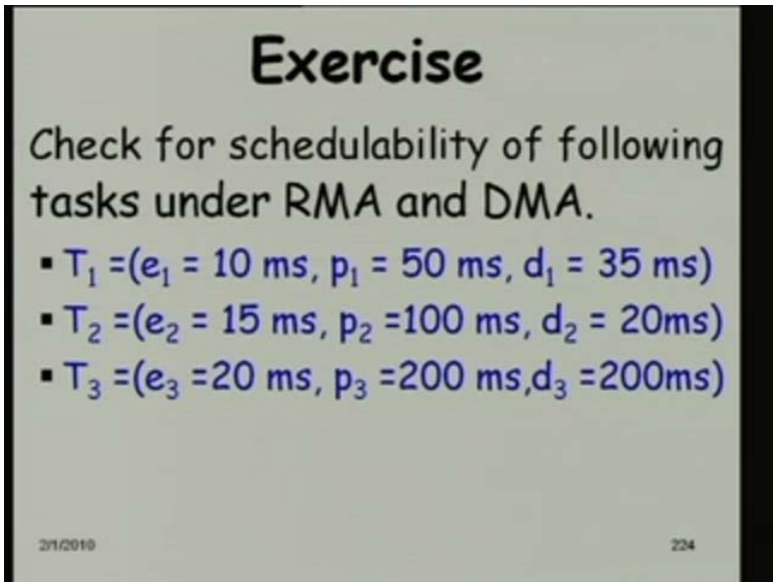
Deadlines are less than the period probably, that will be the case.

No, deadlines are less than period, but the still rate monotonic might produce a feasible schedule, is it not?

Yeah, but…

So, just show that something which is where the deadlines will not be met, using rate monotonic, but it will be possible to meet them using deadline monotonic, but the rate monotonic will always fail, if the deadline monotonic fails.

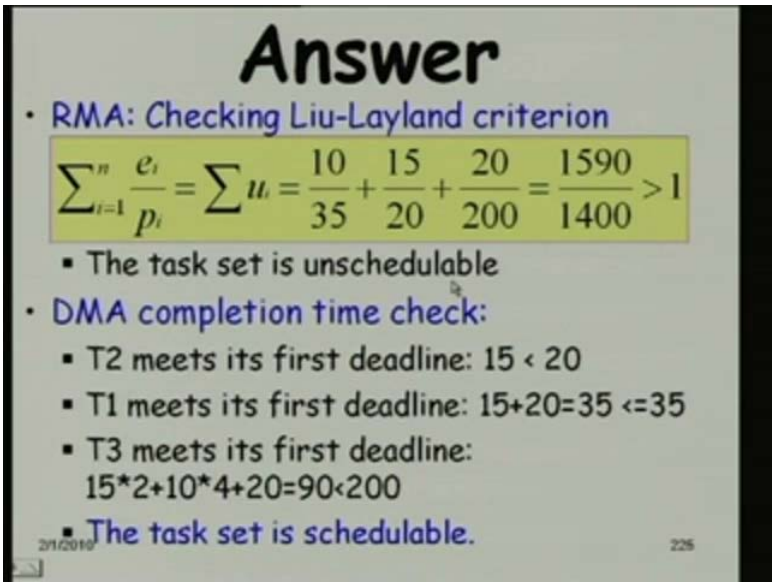
(Refer Slide Time: 10:44)



So, let us just check this set of tasks. Let us say, the task one, execution time is 10 millisecond, period is 50 millisecond, and the deadline is 35; e 2 is 15, execution time period is 10 millisecond and deadline is 20 millisecond; and for e 3 the period and the deadline are the same.

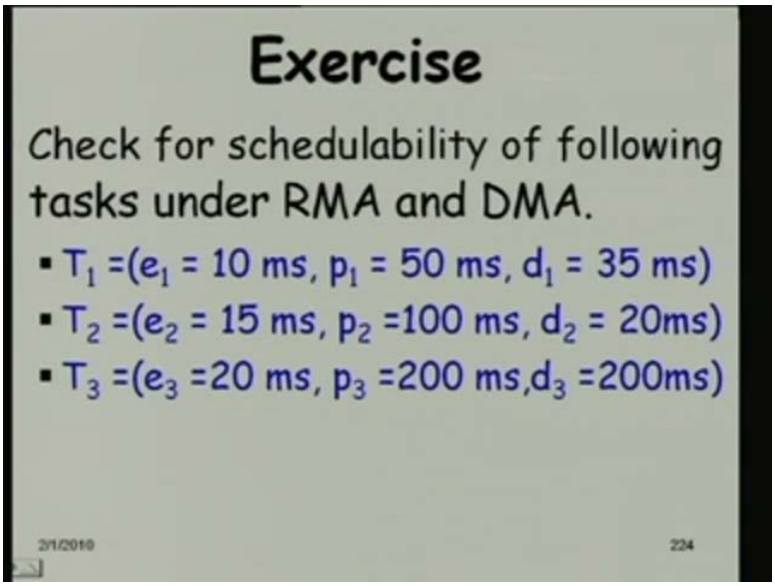
So, let us check the schedulability of the task set, under RMA first. Please, try that. Find the utilization for task 1, which will be 10 by 50, is it not?

(Refer Slide Time: 11:47)



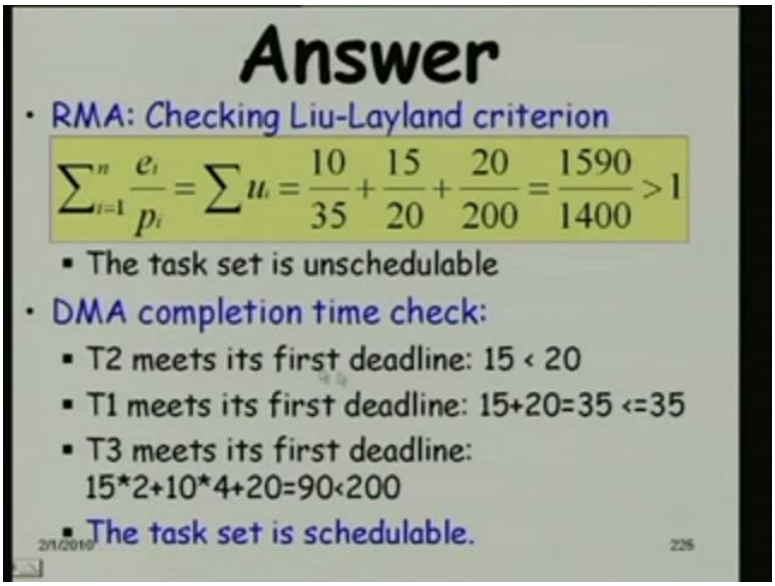
So, let us not wait for your answer, because those who have already done, we can, they can crosscheck the answer. To check the Liu-Layland, we find that the utilization is greater than 1, but let us do the completion time check for the deadline monotonic algorithm. See here, implicitly, see for even for applying RMA, we have taken that the 10 must be completed within 35; 15 must be completed by 20 and so on.

(Refer Slide Time: 10:44)



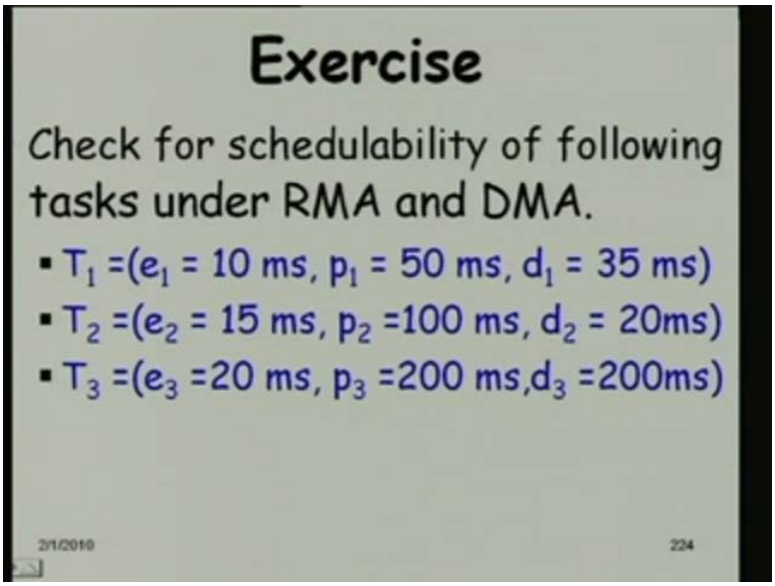
So, here, see here, one thing is according to the rate monotonic algorithm, T 1 has higher priority compared to T 2, is it not? Because of their periods, but according to the deadline monotonic algorithm, T 2 has higher priority than T 1, is it not? So, possibly that makes the difference and we are checking whether T 2 will meet its first deadline. So, 15 is the execution time, and deadline is 20 and it meets it.

(Refer Slide Time: 11:47)



For T 1, which is the next priority, that is the second highest priority, 15 plus 20 is 35, it will just meet and for T 3, both T 1 and T 2 we will have to consider their occurrences. See here, the occurrences have to be with respect to the period, is it not? Implicitly, we in the calculation, we have used that; how many times they occur is based on their period, not on the deadline. See, here, this is 20 is the deadline, but actual period is 200. Is it not?

(Refer Slide Time: 10:44)



No 100, actual period is 100. So, the deadline is 20, but the actual period is 100. So, we have used those to compute this, and we find that 90 is less than 200. So, all the three tasks will meet their first deadlines.

and it will be...

p i; no…

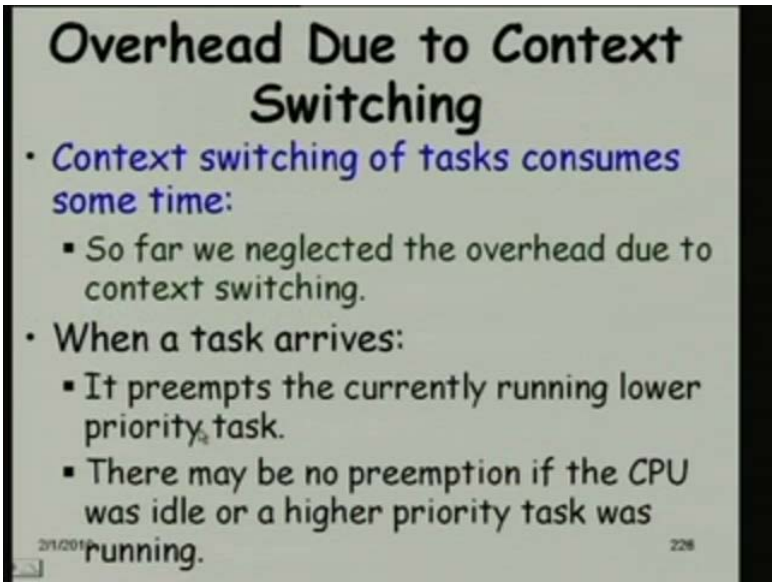
See, we are trying to find out, how many times the task will occurs in that interval, and whenever it occurs, it will start executing the higher priority tasks.

There in the Liu-Layland, we had checked e a by d a is sigma e i by d i. Here you have said, e i by d i.

Yes. It should be e i by d i.

No, see if we can check e i by p… Here, you are saying this would be d i, is it? So, minimum of p i and d i; minimum of p i and d i. Here, we should have made this correction. So, the task set is unschedulable by the rate monotonic, but is schedulable on the deadline monotonic.

(Refer Slide Time: 15:53)



But, just one question - that why do then people use rate monotonic in programming and not deadline monotonic? As you are saying that majority of the schedulers are rate monotonic schedulers, not deadline monotonic schedulers. So, what do we think?

Deadline may be difficult to anticipate sir.

No, deadline can be found out; for every task deadline can be found out.

Relative deadline sir, probably because of real-time distance over deadline flow, periods are same.

Yeah. So, in most of the situations, majority, worst majority of the tasks, the deadline and the period are the same; it is very unusual to find tasks for which the deadline and period are different, and for that we need to tailor the rate monotonic algorithm to handle those issues, rather than trying to use the deadline monotonic algorithm.

But, if we use the DMA itself, then if p and e are equal, it will behave like RMA.

So, what he says is that the deadline monotonic, if the rate and the, the period and the deadline are the same, it is actually the RMA, but the thing is that DMA is much more complex to understand and implement. So, can you please try to investigate. What is the what is the

complexity in implementing the deadline monotonic algorithm, what is the complexity? And please investigate, if there are any other reasons, why DMA is not as popular as rate monotonic algorithm?

Whenever you talk to a real time programmers, if you ask him that - what is the algorithm you are using? He will say, using rate monotonic algorithm. So, why this situation, please investigate, and submit me a report if you find something important or significant. Let us proceed.

Sir, what is the difference between earliest deadline versus DMA.

So, there is the question here - that how is earliest deadline first and deadline monotonic algorithm different?

It must be the dynamic be p 1, p 2, p 3 is used.

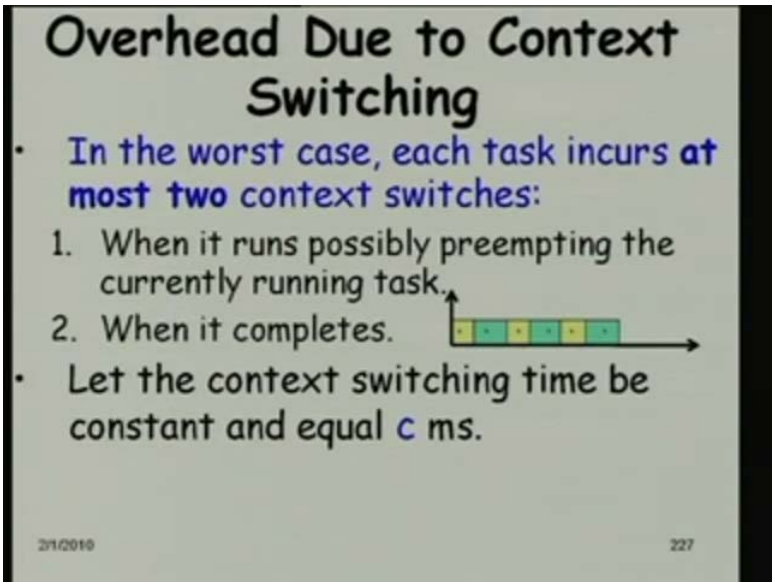
Yes yeah. So, earliest deadline first, as he says, that at each instant, at each scheduling point to find the nearest deadline, and that is taken up for execution, at each instant or each scheduling point, at each scheduling point, the scheduler determines which is the task having the nearest deadline and then takes it up. Whereas, in a deadline monotonic algorithm, you just find the relative deadline of all the tasks beforehand, not at every instant, and then, find out, assign priorities based on that. So, here the priority does not change in a deadline monotonic algorithm; whereas, e t f e t f depending on at which scheduling point you are talking of, different tasks might be considered at higher priority. Is that ok? Let us proceed from that point.

So, let us look at few other issues. One is that since the task execution times are very small, most of these tasks complete in few milliseconds and also their periods are also small. So, it should not ignore the context switching time. So far, for simplicity, we have been ignoring the context switching time. So, during the schedulability analysis, we should also consider the overhead due to context switching. You should take the pessimistic, because context switching also takes different times. It is not a constant time actually. So, it might take between a range of values, but we should take the pessimistic value.

Now, let us assume that the context switching time, at the worst-case, take some time. So, we need to consider that even though, so far we have been neglecting that for simplicity. Now, let us

see, when does the context switch occur? One is that when it preempts the currently running task. So, task - higher priority task - which has a reason, which has come in, will preempt the lower priority task, or if the CPU is idle, there will be no preemption; the task will start running. But for both these cases, there will be a context switch, does not matter whether it preempted or it did not preempt the CPU was idle, or both these cases, the context need to be loaded, is it not? The context switching has to occur.

(Refer Slide Time: 21:13)



So, if you consider that, in the worst case, each task incurs at most two context switches. See, here we will have to consider the worst-case scenario, not the average case or the best case; because, we want to guarantee that the tasks will be schedulable under the worst situation. Unlike in our normal, traditional operating systems, where we are more concerned about the average case behavior, here, we are concerned about the worst-case behavior and at the worst-case, each task incurs at most two context switches, when it preempts the currently running tasks and when it completes. But what if a task is… before it could complete, it is preempted again; I mean its preempted, and then, once it starts running again, it is preempted by some other task; again starts running and again preempted. So, is this assumption holding for this, at best.

For every run, it will have to do.

No, no just look at the question; I am saying that, irrespective of how many times a task has been preempted, at most we can consider that it has undergone two context switches. Every task has undergone two context switches. Is that assumption valid?

(( )) because as and when higher priority process come, then as (( )) the process will be preempted more than two number of times; two level of time.

No, suppose see we are trying to consider, how many times the preemption we need to consider for every task.

Sir, at least once, because it was…

No, not at least, at most.

At most twice, because it has not created, some other task that will have higher priority.

Yes exactly. Yes exactly.

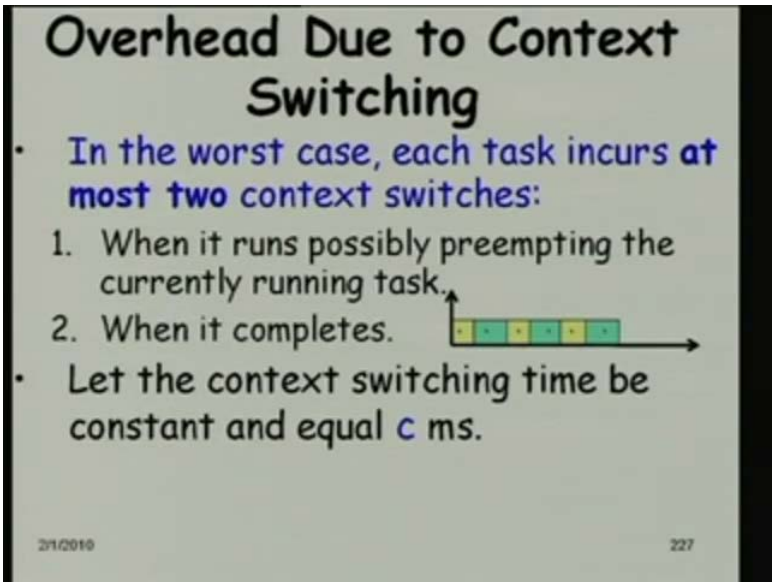
So that will complete fully.

So, let us see his answer, let us look at the diagram here.

See here. So, a higher priority task has come in here, and this has two contexts, which has taken care here right. Two context, one is… see this task started coming in here, at this point, there was a one context switch. Now, next time, this task has preempted it and this context which will be counted towards it, and which will be in our expression taken care, because it is a higher priority task and then it started running again right. So, the higher priority task has undergone two context switches, and these had undergone one context switch at this point, and now, after it completes, there is one more context switch.

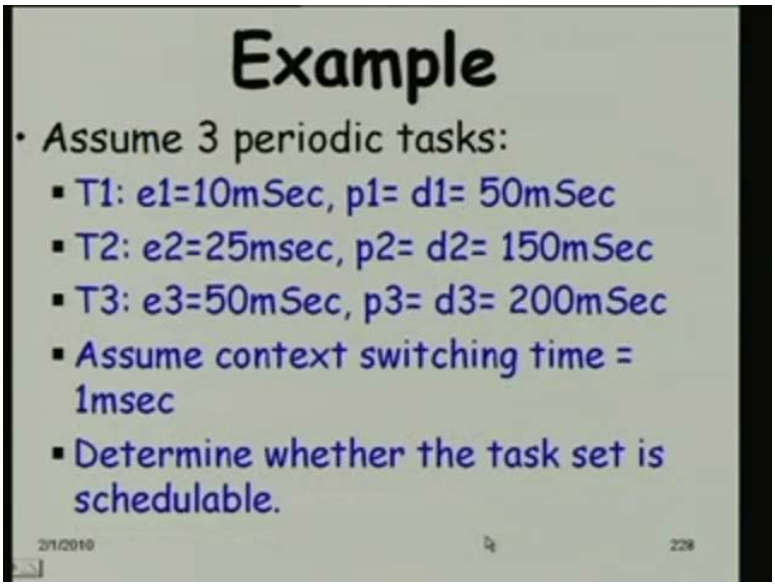
So, just think about it, that at most every task incurs two context switches. If we develop our expression by that, that every task incurs at most two context switches, we will be doing all right, because the higher priority task context switches will be taken care of. They will be added to the lower priority tasks, completion time. So, let us see the expression, possibly that will make more sense.

(Refer Slide Time: 21:13)



Let us, assume that the worst-case context switching time is equal to c millisecond. Now, if we consider, the overhead due to context switching in effect, we can say that the execution time of every task increases by two context switches, because at most two context switches are involved for every task. So, the execution time instead of e i, if we make it e i plus 2 into c, you should have taken the overhead due to context switching.

(Refer Slide Time: 25:11)



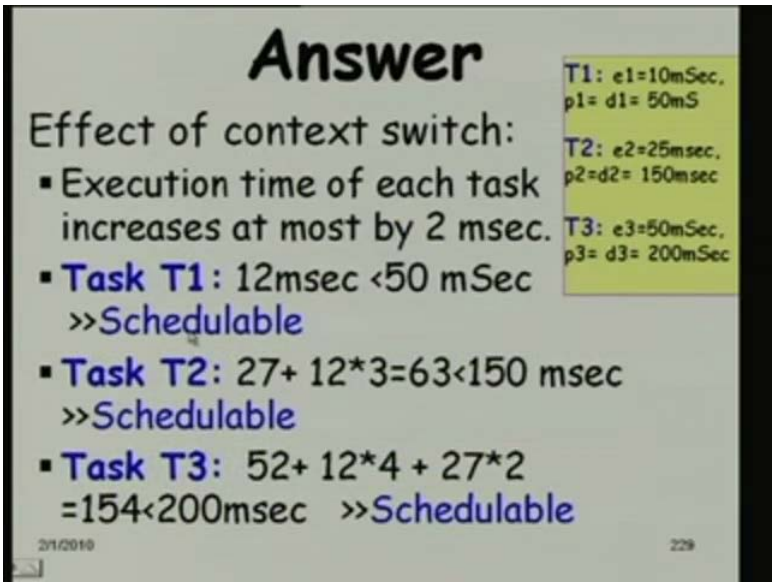
So, let us use that concept here. You have three periodic tasks, whose execution time is 10 millisecond, and the period equal to deadline is 50 millisecond; for T 2 it is 25 millisecond and 150 millisecond; and for T 3 it is 50 millisecond and 200 millisecond; and the context switching time is 1millisecond. See, 1 millisecond is very comparable to 10 millisecond, 25 millisecond, 50 millisecond. So, if we ignore it, the context switch time here, we may not be doing all right. We might arrive at a decision about schedulability, which can turn out to be wrong, because the context switching time is comparable to the task execution times.

Now, let us see how we need to determine the whether the task set is schedulable. So, what we need to do? Can you please tell, what we need to do?

Need to increase the execution time to 10 plus 2 seconds.

Exactly, for each of the task, we need to increase it by 2. So, this will become 12, this will become 27 and this will become 50. Now, when we apply the completion time theorem, we will see that for these to complete, the first task to complete is 12 less than 50; for the second task to complete it will be 27 plus 3 into 12. See here, the context switch times are taken care. The number of times it was interrupted, those context switches are taken care by the higher priority task; your factoring here those context switches. So, considering just 2 for this is all right. Similarly, for this task, we will factor in the context switches due to this T 2 and T 1. Now, let us see the solution.

(Refer Slide Time: 27:17)

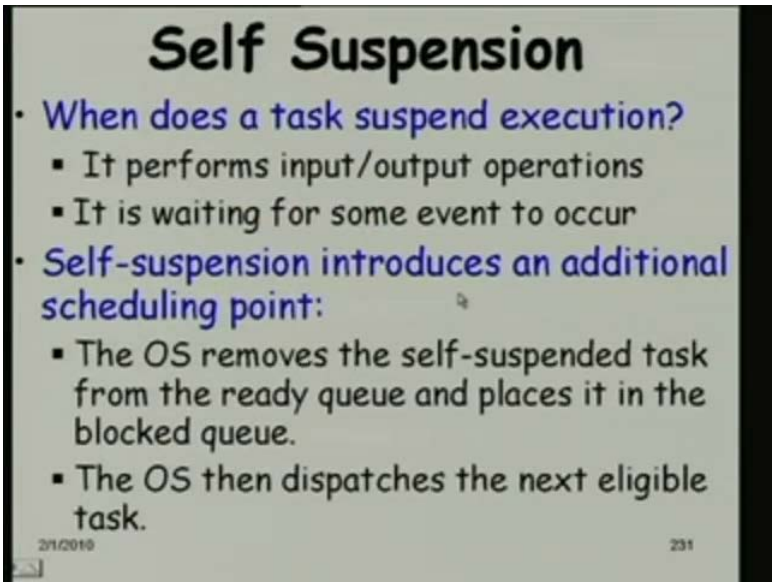


So, the execution times for each task becomes 12, 27 and 52, as we are saying and 12 is less than 50. So, the first task satisfies the completion time theorem; task T 2, 27 plus 12 into 3, this also satisfies; and the third task, also satisfies the completion time theorem. So, even after factoring in the context switch times, the task set is schedulable. Does that appear all right? Any difficulties here?

Should we always assume that context is trying to be constant?

Yeah see, we are taking the worst-case scenario. So, if we say that a task set is schedulable under all situations, the task sets will be schedulable, that is why we are taking the worst-case context switch time.

(Refer Slide Time: 28:26)



Let us proceed. Now, let us consider self suspension. This is another issue, which we have so far been ignoring. A task when it starts running, the execution time is given, we have been assuming that it runs to completion, but it might self suspend, even though it is the highest priority, still it might relinquish the CPU and suspend itself; why is that so? What do you think?

Waiting i o.

Yeah exactly, a task might self suspend waiting for i o or may be some events. So, these are two conditions, under which a task might self suspend. So, let us look at that, that the task when it performs input-output operations or it is waiting for some events to occur, itself suspends.

Now, what is the impact of self suspension on the schedulability? We need to analyze that, because if a task self suspends, and runs at later instant, it can affect the schedulability of the lower priority tasks, is it not? It will not affect the schedulability of higher priority task, because it has to anyway yield the CPU to higher priority task, but lower priority tasks might be affected. So, let us see what is the impact. Again, we will make a simplistic assumption here, we will restrict the number of self suspensions. So, we have... So far, restricted to considering only two scheduling points; is it not? What were the two scheduling points?

Minus z (( )).

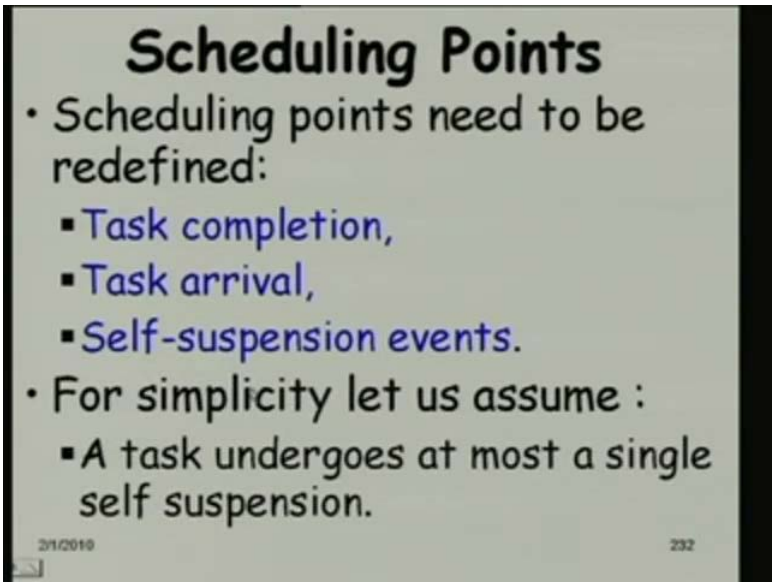
No, these are event driven schedulers. See, what you are talking of is cyclic schedulers.

(Audio not clear)

Yeah. So, in an event driven schedulers so for, we have been considering two scheduling points. One is when a task arrives, the scheduler needs to check whether it is a higher priority task than the currently running one, and whether it should be scheduled; and the second is when a task completes, it will check which is the highest priority now, and take that upper execution. So, two scheduling points, we have so far considered for event driven scheduling.

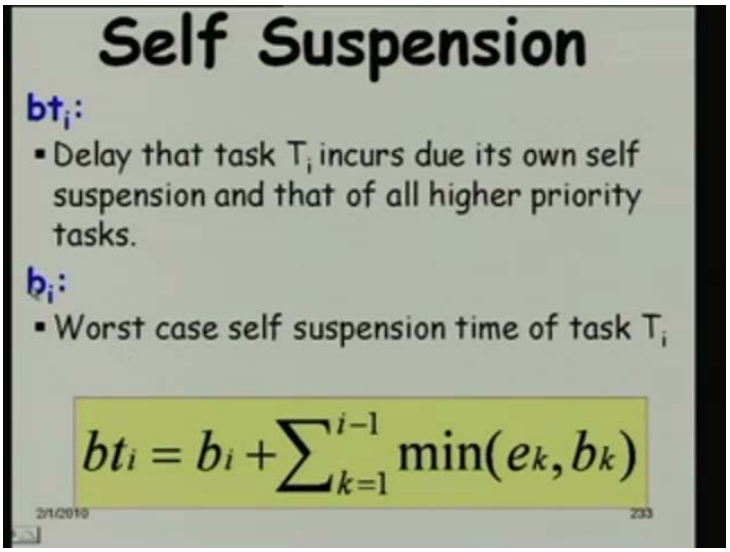
Now, we have to consider, a third scheduling point, which is at the self suspension. When a task self suspends itself, it is placed on the blocked queue, removed from the ready queue and placed in the blocked queue; from a traditional operating system, we know this much, and out of those tasks which are in the ready queue, it dispatches the next highest priority task.

(Refer Slide Time: 31:27)



So, in addition to task completion and task arrival events, we also need to consider the self suspension events. And let us assume that every task undergoes at most a single self suspension. We will have to restrict it to a single suspension or two suspension or something like that, because if it suspends arbitrary number of times, doing a schedulability analysis will become extremely difficult.

(Refer Slide Time: 32:11)



So, let us denote these two delays. One is the delay that the task T i, incurs by its own self suspension, and also, that of its all higher priority tasks; we will denote it by b T i; and b i is the worst case as self suspension time of the task T i.

So, b i is the is the self suspension time of the task T i and b t i is the self suspension, is the delay, that the task incurs due to its own self suspension, which should include b i, and also, the effect due to the self suspension of all higher priority tasks right. We are implicitly assuming here, that the tasks are arranged in order of their periods.

So, we have written this expression that b t i, the worst case delay that the task T i will incur is its own self suspension, which is obvious, that its completion will be delayed by the time itself suspends itself ,right. And now, just look at the second expression. We are saying that for the higher priority tasks, which is the minimum of the task's execution time, and the self suspension time. So, what about this one - the second term? Since we have written, just summation of b k; what do you think? Anybody would like to comment on this? This is the question clear. You see for the first term, in the delay due to self suspension which is b i is obvious, because if a task has suspended for some time, its completion time will be delayed by that much, but for the higher priority tasks when they self suspend, we are saying that we need to consider the maximum the execution time and the self suspension time sorry minimum of the… sorry.

Sir, the effect of the higher priority versus will only incur on this terms of this delay or to the lower, one they will not have. They will not… (Audio not clear)

Not clear, anybody would like to answer any other way? Yes, anybody like to comment on this - That why is it that we are considering the minimum of the self suspension time and execution time of the higher priority task? The answer is not so complicated.

See here, a lower priority task T i gets delayed due to a higher priority task, when after a higher priority task self suspends and it blocks the execution of the lower priority task right. Now, even if itself suspends for arbitrary number of point of time, the maximum over lap that can occur is e k or b k right. See, if it a self suspended, and by b k the task has got delayed. So, the maximum time on which it will overlap on the execution of the lower priority task is minimum of e k and b k.

See, even if it is clearly overlapping on the lower priority task, it will delayed by maximum e k right. Now, the delay is also restricted by b k, if b k is smaller then due to the self suspension, it will at best overlap for b k is it not? From the earlier case, where there was no self suspension and when the self suspension occurs for a higher priority task, the maximum change in the overlap will be b k, if b k is smaller or e k is the… If e k is smaller that is the maximum overlap that can occur. So, the maximum delay that a lower priority task will incur, due to self suspension, is its own self suspension and minimum of e k, b k for all higher priority task for single suspension sorry.

Is not clear, sir.

Not clear. Confusion in e k.

See, a lower priority task gets delayed due to self suspension, when there is a higher priority task, which has become after self suspension ready or something like that right. So, the higher priority task will be taken up and this will not be executed. Now, if let us say, let us assume the case when b k is less than e k right. So, what will be the maximum overlap that will occur with the lower priority task? b k. Now, let us say e k is less than b k. So, even if it has no delayed it by b k, but the maximum overlap that will occur is e k, is it not? Because it will complete by e k, it can prevent the lower priority task from executing by maximum e k, is it not? Even if it has self

suspended for large number of time, the maximum overlap due to this task will be e k, because after e k, it will anyway complete.

But, e k should be the execution time of the lower priority tasks, not the completion time of the… is the suspended

No, e k, no, e k is the higher priority task which is blocking a lower priority task T i. So, e k 1 when it is executing, T i will not be able to execute when t k is executing, T i will not be able to execute and it will complete in anyway by e k, right? e k is the execution time. So, the maximum it will, it can delay is by e k, but when b k is less than e k, it can at most delayed by b k right. Just think over it.

If b k is less, then it is obvious sir.

Yes.

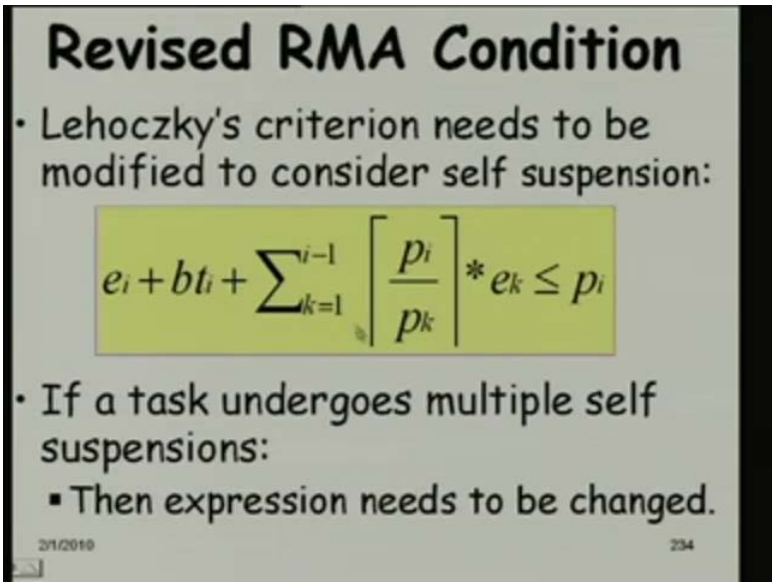
But, when e k is less, even then the task T i may not get executed in between the waiting of this other high priority.

No, once it self suspends, the CPU is relinquished; any task will start executing.

But, still the wait in delay time, you have to consider as b a, complete b a.

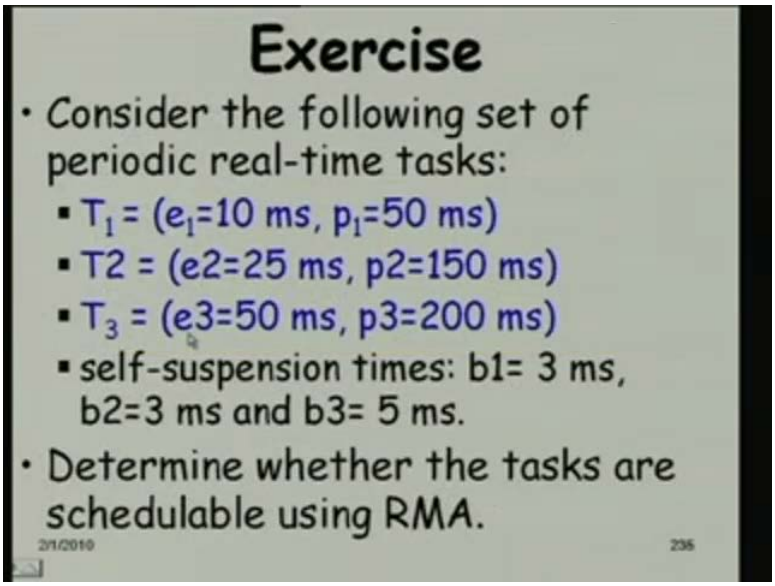
No, see b k it has the higher priority task self suspended itself, and that time the CPU is available. So, that is not a problem. The problem is that when it self suspends and then starts executing in phase or when the lower priority task is executing and it preempts it, is it not? So, that we are saying is that, it can at least prevent the lower priority task from executing by maximum e k time, because by e k time the t k will complete; think of it. So, the maximum delay due to self suspension is b t i which has two components. One is its own self suspension and other is minimum of e k, b k for every higher priority task, that is k equal to 1 to i minus 1.

(Refer Slide Time: 39:53)



Now, we need to change the Lehokzky's completion time theorem as follows. So, earlier we are just writing e i sigma k equal to 1 to i minus 1, ceiling of p i by p k into e k. Now, we need to add this term b t i, because this time will also be the task will be delayed right. And, of course, this is a single self suspension that we have considered. If we consider, two self suspension or so, or maximum n self suspensions, we need to change this expression here.

(Refer Slide Time: 40:37)



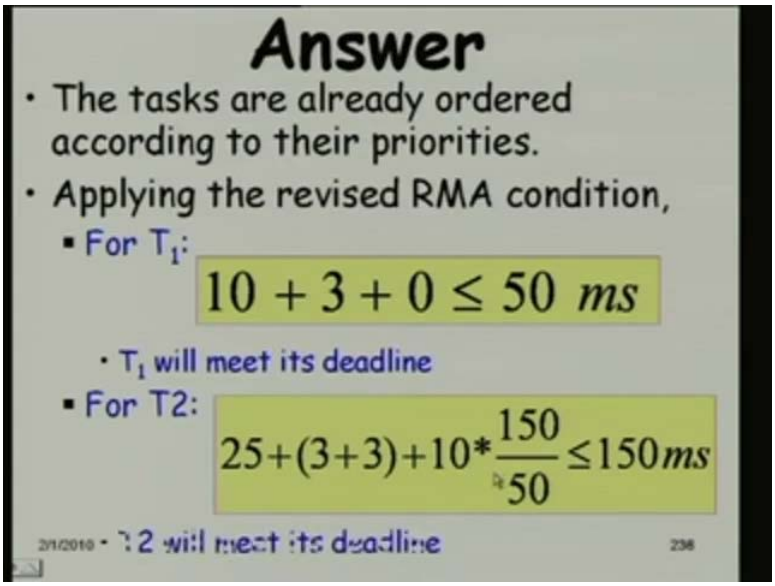
So, let us look at this exercise. We have three tasks, T 1, T 2, T 3; for simplicity, I mean without making it too non-trivial two tasks, we are just considering three tasks, but in a realistic scenario, you might have 5 or 10 tasks.

So, e 1 is 10 millisecond and p 1 is 50 millisecond, e 2 is 25 and 150 and e 3 is 50 and 200 millisecond; and the three tasks are different self suspension, maximum self suspension times; for T , it is 3 millisecond, for t 2 it is 3 millisecond and for t 3 it is 5 millisecond. So, will this task be schedulable. What do we have to do here?

Sir, we need b t for each and every task then add…, exactly.

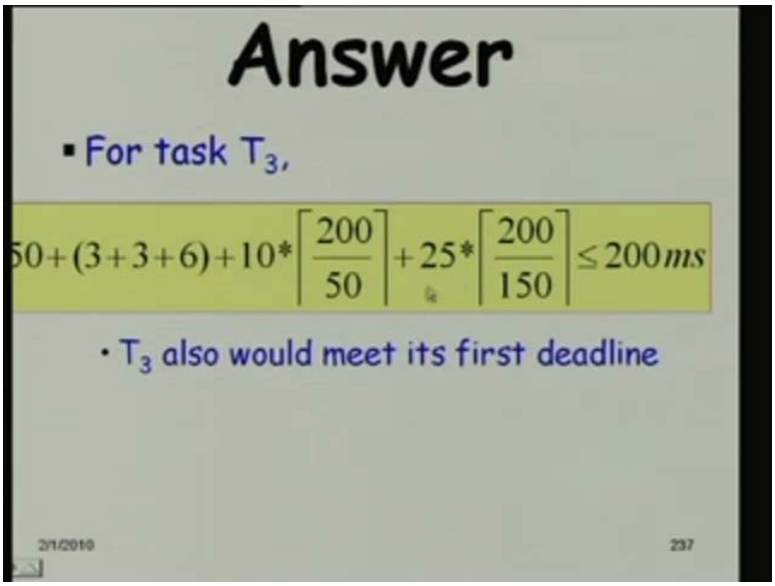
So, we need to compute the b t i, for every task T I, and then in the completion time theorem, we need to factor in that delay.

(Refer Slide Time: 41:49)



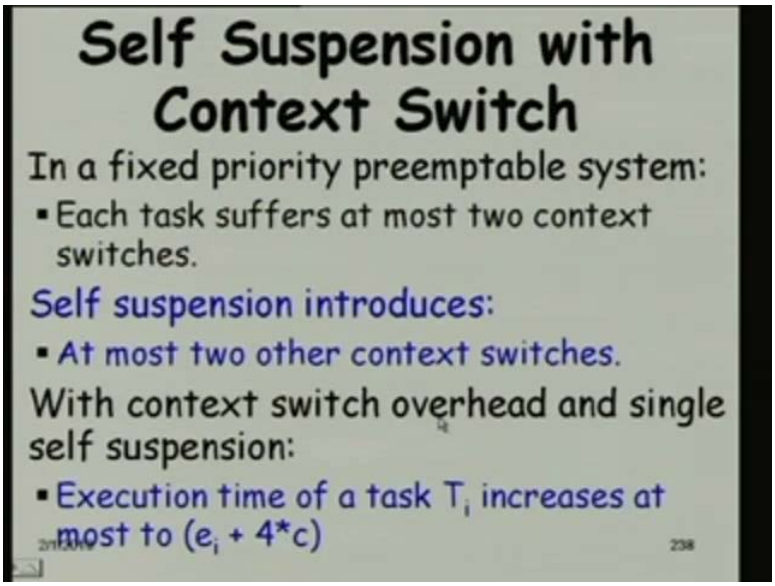
So, that is what is done here. For the first task, it is 3; b t i is 3 and the delay due to higher priority tasks is 0, because it is the highest priority task. So, execution time is 10 and its self suspension is 3, which satisfies is less than 50, its deadline. T 1 will meets its deadline, but T 2 we need to consider, the higher priority task which is 3 plus 3 is 6 and ceiling of 150 by 50 is 3. So, 3 into 10 is 30, 30, 55 plus 6 is 61 which is less than 150. So, T 2 will also meet its deadline.

(Refer Slide Time: 43:30)



For the task T 3, we need to add its execution time, and then the self suspension time or the execution time, whichever is lower for all higher priority tasks, and then, the execution time for the higher priority tasks during this duration. So, if you simplify this, that is 2 into 25 plus 4 into 10 plus 12 plus, that is 50, is it not? So, that is 50 plus 12 plus 40. So, 102 plus 50 is 152. So, for T 3 also it will meet the first deadline.

(Refer Slide Time: 43:47)



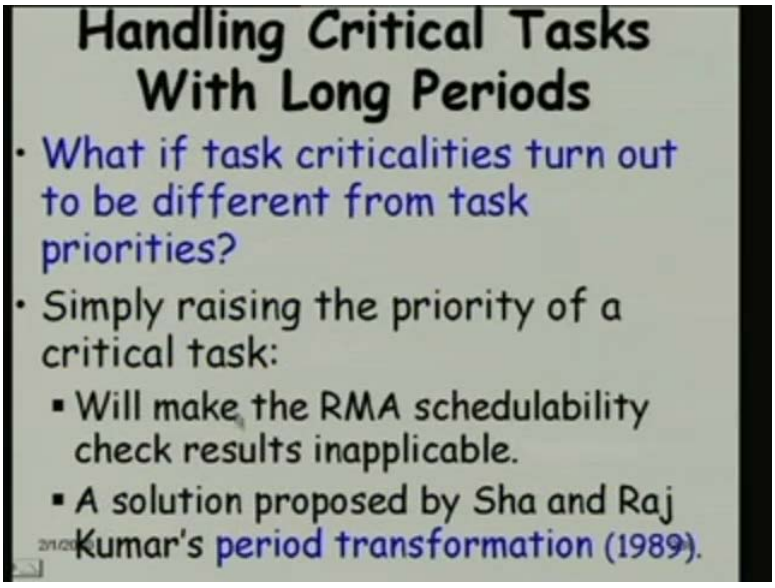
Now, let us see how do we take care of both self suspension and context switch? So far we have been assuming that the context switch is restricted to two. One when it preempts or it starts executing, and another it completes. And we said that, all higher priority tasks when the preempt it will be factored in and we do not have to consider for them. So, just two was enough. But, when you have additional scheduling point due to self suspension, we need to change that calculation, the maximum number of context switches that we need to take into account.

So, let us look into that. Since, we are assuming a single self suspension, the self suspension will introduce at most, two other context switches. So, we need to make it e i plus 4 into c.

Is suspension is an resumption?

Yes, self suspension and resumption.

(Refer Slide Time: 44:49)



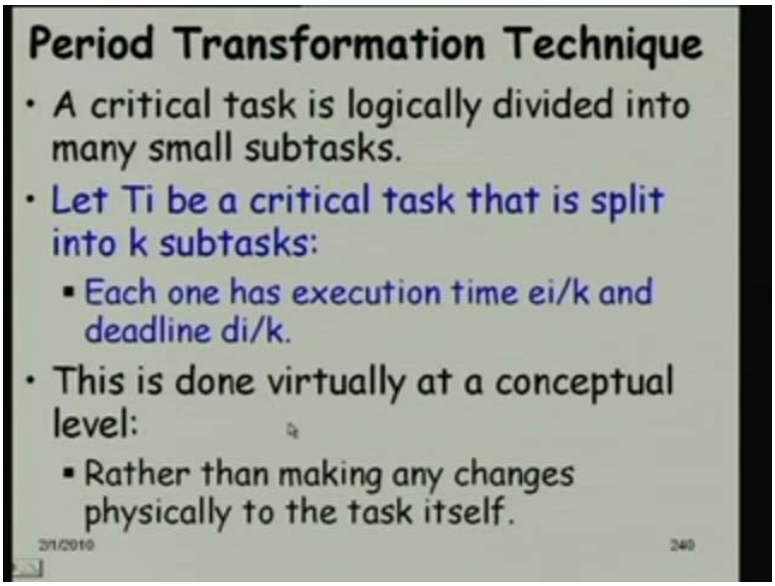
So, for taking care of both self suspensions, single or two time self suspension, and context switches, we can do it very easily, but now let us see another issue, in using the rate monotonic algorithm, in actual system development. You will have situations, when the task criticality is different from the task priority. See here, we have assigned priority, solely based on what is the rate at which the tasks are arising, right; the rate of the task - based it solely on that criterion, I have assigned the priority, and it might so happen that a task which whose period is slightly

larger, but that is a critical task; for example, the response needed for a robot to take deviation from a obstacle. Now, but that task should not fail, even when there is a overload.

A lower priority task, it can fail and still the system will work, but for critical tasks, in a over load situation also it should not fail. So, how do you take care, because here we have a contradiction, a conflicting situation here, where a lower criticality task has higher priority. One thing is that we can raise the priority of a critical task, even if we find that its period is high or the rate is low, still you can assign it a priority of 1, but if we do that, then we will be violating, the schedulability assumptions, in the schedulability criterion computation right.

So, the schedulability results will not be applicable. Now, what if we… So, what is the solution? How do we do that? A solution was proposed by Sha and Raj Kumar. The period transformation technique reported in 1989.

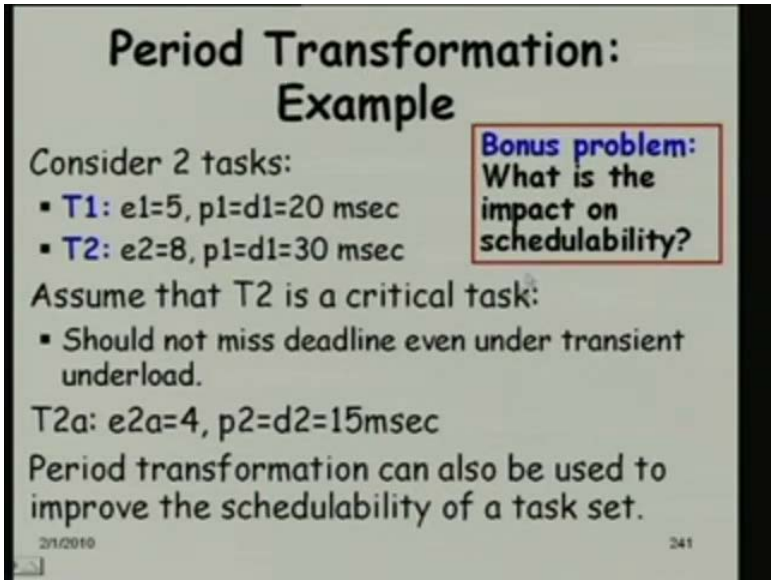
(Refer Slide Time: 47:20)



Now, let us see the essence of the result; very simple result actually; the paper is available on the internet. If you just give Sha and Raj Kumar period transformation technique, you will get the paper. So, here the task T i has execution time e i, and let us say, it needs to complete by p i , let us say. So, what we can do is, we can say that see, we will split it into k smaller tasks. So, instead of considering e i over a period p i, we can as well consider e i by k over p i k, that will take care

of it. Is it not? We just reduce the task e i, p i to e i by k and p i by k. So, it is in effect, its period has reduced by k times, and this is done at a conceptual level, we do not have to really split the task. In the schedulability computation, we just use this.

(Refer Slide Time: 48:41)

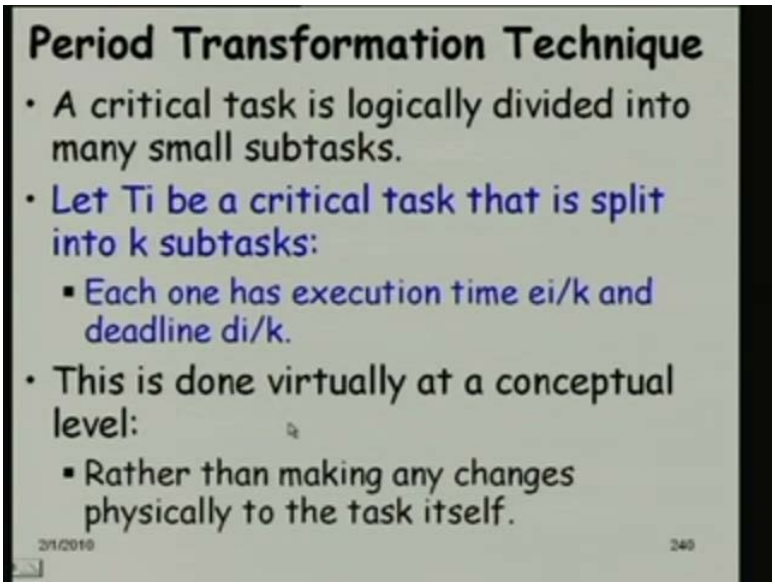


But one thing we must note here, that… that we will just talk little later. So, let us just look at an example of the application of the period transformation.

So, let us consider two tasks. e i, e 1 is equal to 5 millisecond and the period is 20 millisecond, and e 2 is 8 millisecond and the period is 30 millisecond, and then we find that T 2 is a critical task; whereas, T 1 is not as critical as T 2. So, if T 1 misses the deadline, it is not a severe failure; but T 2 should not miss its deadline, then the robot tumbles or something like that, collides and tumbles; so, this is a critical task.

So, what you have done here is that we have transformed T 2 into T 2 a, e 2 a is 4 and p 2 is 15. So, we have in effect, split the execution time and the period by half, and now, see that T 2 a is assigned the higher priority than T 1 and we can also use this in our schedulability analysis.

(Refer Slide Time: 50:10)



But, just one observation here, I think I forgot to written here, to write here. That see, even though we are splitting it on the into k smaller tasks, the actual deadline is on the last task, is it not? The last of these smaller tasks is actually critical. So, if we can use that in our schedulability analysis, possibly the schedulability, it will become more schedulability right. We can take that into for the tasks, for the k minus 1 tasks, the deadline, if that it is also missed, it is not problem. I mean missed by a small factor, it is not a problem, but for the last task - sub task - the deadline is missed then the task will fail right. Just remember that.

But, is there any impact on schedulability at all, is it that a task set which was schedulable earlier by splitting it, it will become non-schedulable, and is it also, that we can improve schedulability of a task set by splitting the task into smaller periods. So, we need to analyze two situations: one is that an already a schedulable task set, after we split a critical task into smaller parts, it becomes schedulable and second is can we improve the schedulability of a task set, which was unschedulable earlier, and by splitting it, we have improved it schedulability, it has now become schedulable. So, both ways we have to consider; please investigate this.

As we are saying that Sha Raj Kumar's paper is available on the net for everyone to look at it, 1989 paper, we can consult that or any other paper or any other literature that you wish, and

please submit this result, that what is the effect of period transformation technique on tasks schedulability?

So far, we have been posing large number of bonus problem, please attempt that seriously, because we will evaluate all of them, and whoever is sincere, trying to put some effort and do it, search the literature, Google search and think about it originally, and submit results, will be rewarded. So, let us break now, we will meet for the next lecture.