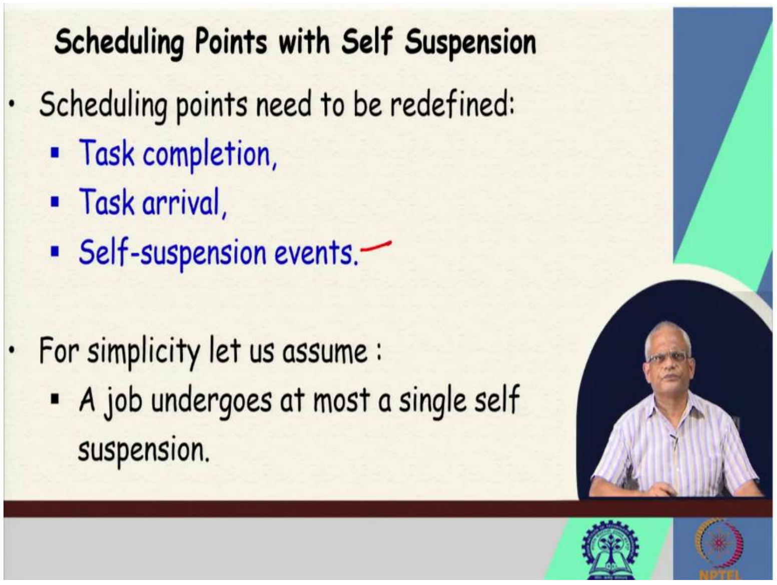
Real Time Systems
Professor. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture No. 21
RMS Generalizations

Welcome to this lecture. In the last few lectures, we have been discussing about the rate monotonic scheduler. As we have been saying that rate monotonic scheduling is very popular in real time systems. If you are programming a real time system it is very likely that he will use a rate monotonic scheduler unless you are developing a very simple embedded application which does not even have scope for large number of tasks and so on then you are going to use the cyclic scheduler.

Otherwise, it is very likely that you will use the rate monotonic scheduler or a variant of that. We have been discussing some issues with rate monotonic scheduling. For example, we discussed about the deadline monotonic scheduling, the deadline monotonic analysis and the deadline monotonic scheduler is appropriate when the task deadline and period are different. It is the optimal scheduler in that situation. And we have been discussing few other issues like self-suspension and context. Now, let us proceed from that point.

(Refer Slide Time: 1:48)



Scheduling Points with Self Suspension

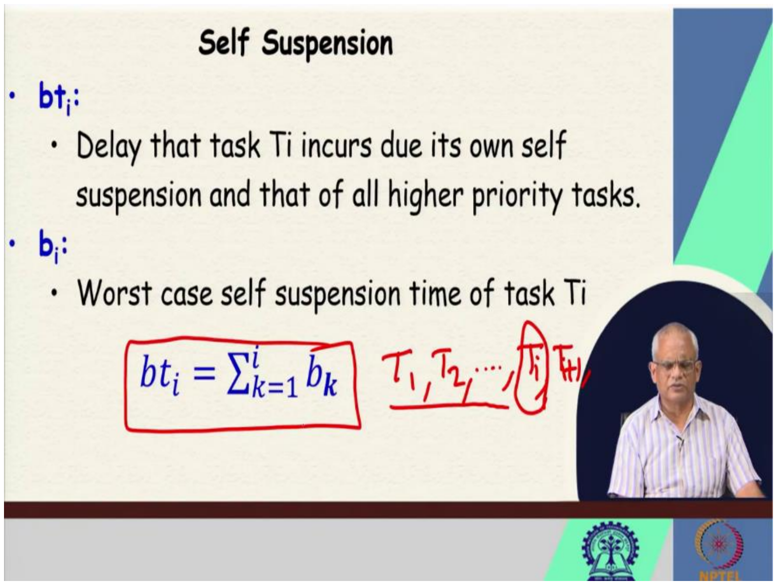
- Scheduling points need to be redefined:
  - Task completion,
  - Task arrival,
  - Self-suspension events.
- For simplicity let us assume :
  - A job undergoes at most a single self suspension.

In the last lecture towards the end were discussing about self-suspension of tasks. Tasks do self-suspend, because of a variety of reasons, maybe they are waiting for an event. Maybe the resources are not available. Maybe they are doing a input or output and then they self-suspend and then other tasks, lower priority tasks feature waiting to execute, they start executing and the tasks self suspends.

So, the scheduling points, those are the points where the scheduler becomes active, starts running to select the next task in the pure rate monotonic scheduling, it is the task arrival and the task completion events. But with self-suspension, we have additional scheduling points. So, not only task completion, task arrival, but also the self-suspension events. When a task self-suspends, the scheduler needs to become active start running and look for which task to run at that point.

And for simplicity, let us assume that job that is a task instance undergoes at most a single self-suspension. You might say that this is too restrictive, because job may undergo multiple self-suspensions, once maybe to wait for event another time to read input and so on. Yes, those cases can be considered. If we understand a single self-suspension case, we can easily generalize our result to multiple self-suspensions.

(Refer Slide time: 4:16)



Self Suspension

- bt_i:
  - Delay that task Ti incurs due its own self suspension and that of all higher priority tasks.
- b_i:
  - Worst case self suspension time of task Ti

$$\boxed{bt_i = \sum_{k=1}^{i} b_k} \quad \underline{T_1, T_2, \ldots,} \boxed{T_i}\, T_{i+1},$$

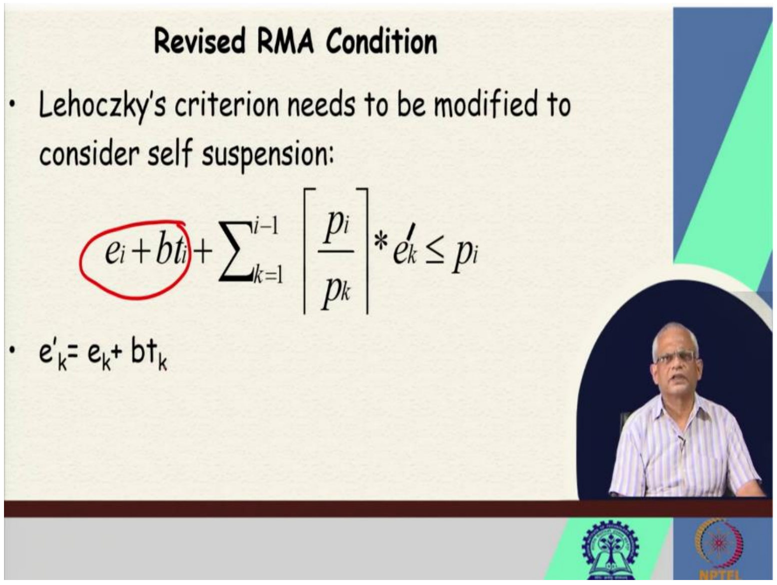
Now, to analyze the impact of self-suspension and schedulability, let us introduce these notations bt_i is the delay a task incurs due to own self suspension and the self-suspension of the higher priority tasks. This is in addition to the other delay that the task undergoes this delay bt_i is when we relax that there are no self-suspension and to allow our task to self-suspend. And b_i is the worst case self-suspension time of T_i.

Now, we can give b_i, bt_i that is the total delay on account of self-suspension by task T_i is the submission of the self-suspension time of not only this task, but also all higher priority tasks.

So, assuming that T_1 is the highest priority T_2, the tasks are arranged in order of their priority and T_i, T_{i+1} is a lower priority task.

So, the task T_i the delay incurred is not only due to its own self suspension, but also the self-suspensions of all other tasks. And that is what we are expressing here. bt_i is the submission of self-suspension of not only the present task, but also all its higher priority tasks.

(Refer Slide Time: 6:17)



Revised RMA Condition

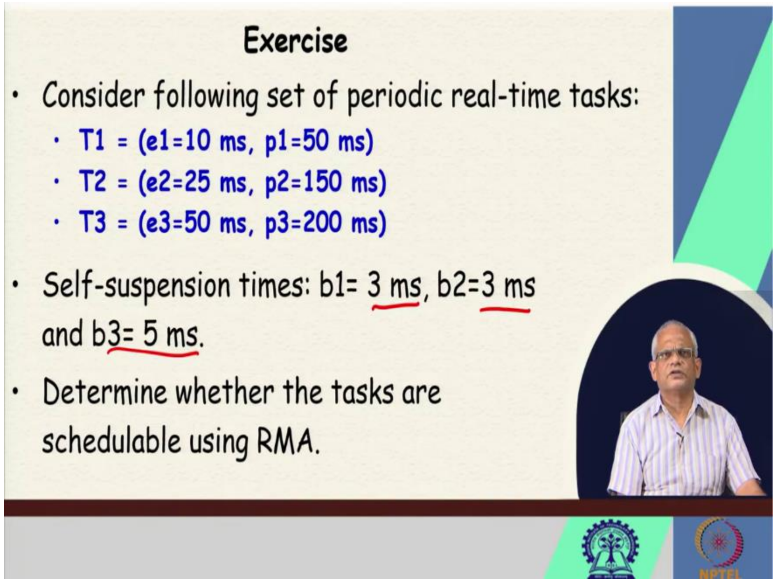
- Lehoczky's criterion needs to be modified to consider self suspension:

$$e_i + bt_i + \sum_{k=1}^{i-1} \left\lceil \frac{p_i}{p_k} \right\rceil * e'_k \le p_i$$

- $e'_k = e_k + bt_k$

And having defined that term, let us define the Liu Lehoczky's criterion that is the completion time theorem in presence of self-suspension, the execution time of the current task becomes $e_i + bt_i$ due to self-suspension and also the task undergoes delay, according to the completion time theorem is $\lceil p_i / p_k \rceil$ * execution time of task t_k, but we will use e'_k where $e'_k = e_k + bt_k$.

(Refer Slide Time: 7:15)



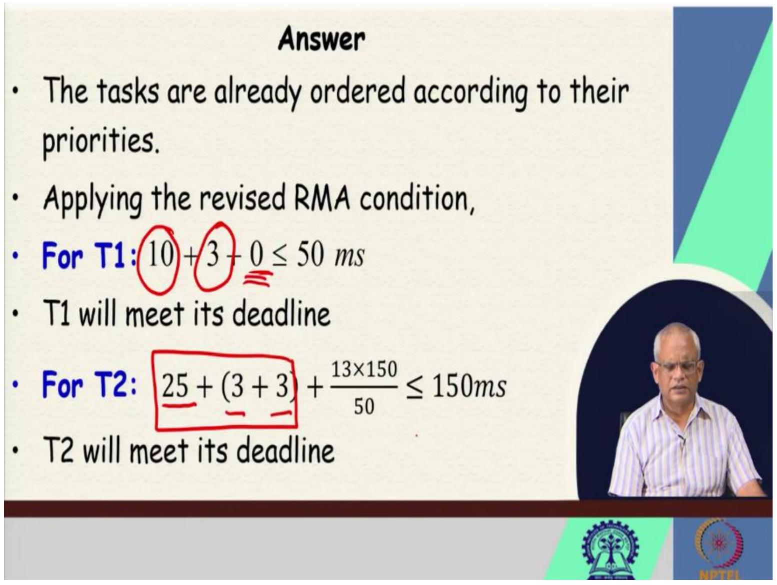
Exercise

- Consider following set of periodic real-time tasks:
  - T1 = (e1=10 ms, p1=50 ms)
  - T2 = (e2=25 ms, p2=150 ms)
  - T3 = (e3=50 ms, p3=200 ms)
- Self-suspension times: b1= 3 ms, b2=3 ms and b3= 5 ms.
- Determine whether the tasks are schedulable using RMA.

Now, let us do a exercise let us say we have 3 tasks T1, T2, T3 and execution time are 10, 25 and 50 and their periods is 50, 150 and 200. Now, we know the worst case self-suspension sometimes of T1 is 3 millisecond T2 is also 3 millisecond and for T3 it is 5 milliseconds and we want to find out the schedulability result and the worst case response time for the tasks T1 T2 T3.

So, we would like to use the completion time theorem to find out whether the task set T1, T2, and T3 is schedulable in a rate monotonic scheduler, when we consider the self-suspension times of the 3 tasks and also we want to find the worst case response times.

(Refer Slide time: 8:35)



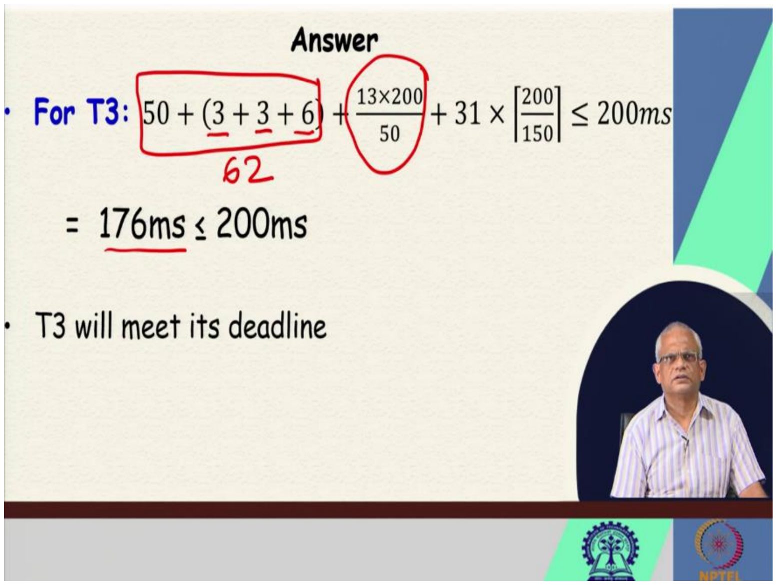
Answer

- The tasks are already ordered according to their priorities.
- Applying the revised RMA condition,
- For T1: $10 + 3 + 0 \le 50\ ms$
- T1 will meet its deadline
- For T2: $\boxed{25 + (3 + 3)} + \frac{13 \times 150}{50} \le 150ms$
- T2 will meet its deadline

Now, for T1, 10 is the execution time 3 is the self-suspension time and T1 is the highest priority task. So, it does not incur any delay account on account of its own higher priority task because it itself is the highest priority task and 10 + 3 < 50, the worst case response time per T1 is 13.

So whenever there is a T1 task instance it will complete by 13 milliseconds and the deadline is 50 milliseconds it will complete well before its deadline. Now let us consider T2; for T2, 25 is the execution time its own self suspension time is 3 and its higher priority task is 3. So the education time for T2 considering self-suspension is 31. And now it is higher priority task considering the self-suspension time it is 13 so (13 X 150) / 50 < 150. So, T2 is schedulable.

(Refer Slide Time: 10:22)



Answer

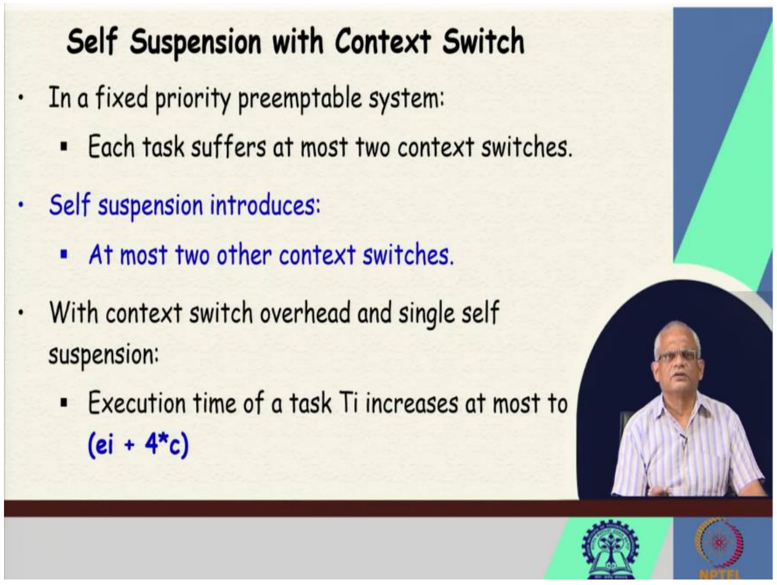
- **For T3:** $\underbrace{\boxed{50 + (3 + 3 + 6)}}_{62} + \frac{13 \times 200}{50} + 31 \times \left\lceil \frac{200}{150} \right\rceil \le 200ms$

  $= \underline{176ms} \le 200ms$

- T3 will meet its deadline

Now, what about T3? For T3, the highest priority task has 3 and the second highest is 3 and the third task T3 has 6 as the self-suspension. So, its execution time becomes 62 and for the highest priority task, the delay at most will be (13 X 200) / 50. And for the second highest priority task it will be 31 X ⌈ 200 / 150 ⌉ and it happens to be 176. So, the tasks are scheduled.

(Refer Slide Time: 11:16)



Self Suspension with Context Switch

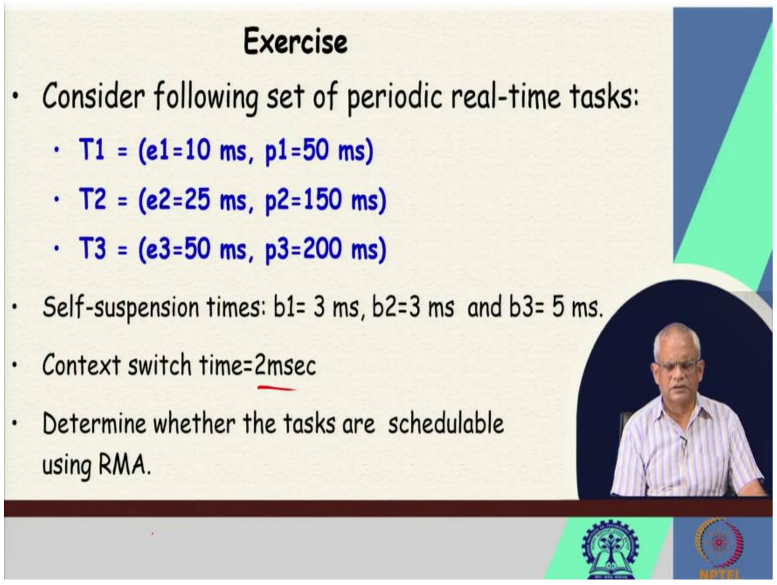
- In a fixed priority preemptable system:
  - Each task suffers at most two context switches.
- Self suspension introduces:
  - At most two other context switches.
- With context switch overhead and single self suspension:
  - Execution time of a task Ti increases at most to (ei + 4*c)

So, with self-suspension, the task set was schedulable. Now, let us consider the context switch times. Because if the task execution time is comparable to the context switch times. We need to consider the context switch time we had seen that without self-suspension in a fixed priority preemptable system; each task suffers at most 2 context switches.

Now, self-suspension introduces at most 2 other context switches because the task self suspends and then it resumes. So, there are 2 other selves context switches, the task self suspends once. Now, the total number of context switches for a task will be 4*c. So, we can consider the effect of context switch by adding 4c to ei.

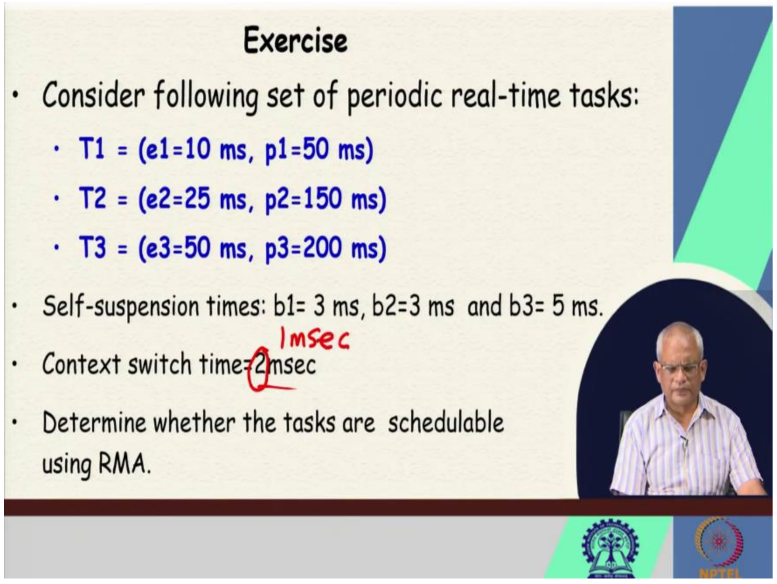
So, every tasks execution time as if it is increased by 4*c. So, that is how we can take the context switch times with the self-suspension, the impact of context switch time with self-suspension on the schedulability of a task set.

(Refer Slide Time: 13:02)



Exercise

- Consider following set of periodic real-time tasks:
  - T1 = (e1=10 ms, p1=50 ms)
  - T2 = (e2=25 ms, p2=150 ms)
  - T3 = (e3=50 ms, p3=200 ms)
- Self-suspension times: b1= 3 ms, b2=3 ms and b3= 5 ms.
- Context switch time=2msec
- Determine whether the tasks are schedulable using RMA.



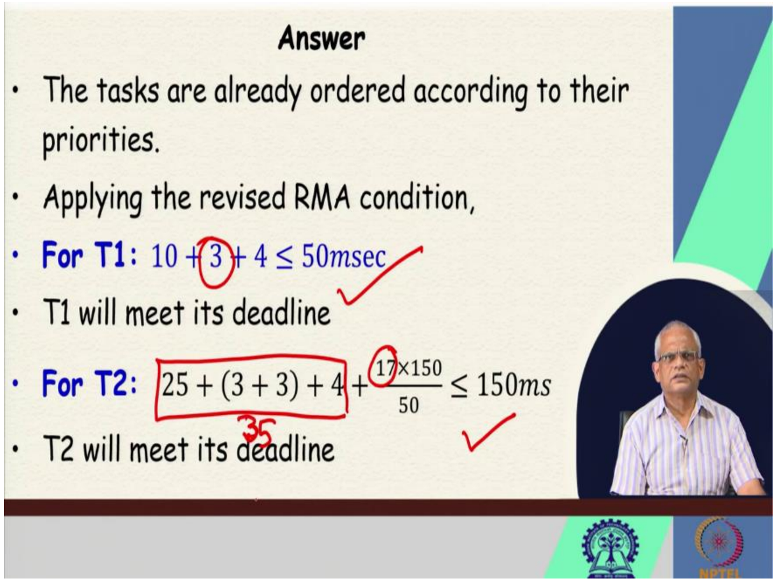
Exercise

- Consider following set of periodic real-time tasks:
  - T1 = (e1=10 ms, p1=50 ms)
  - T2 = (e2=25 ms, p2=150 ms)
  - T3 = (e3=50 ms, p3=200 ms)
- Self-suspension times: b1= 3 ms, b2=3 ms and b3= 5 ms.
- Context switch time=2msec
- Determine whether the tasks are schedulable using RMA.

1msec

Now, let us consider the same task set which was schedulable under self-suspension. Now, let us consider the same self-suspension times but also consider context switch time of 2 milliseconds. Now, we want to check whether the task set is schedulable? For the sorry, this is 1 millisecond. So, the problem this is 1 millisecond.

(Refer Slide Time 13:50)



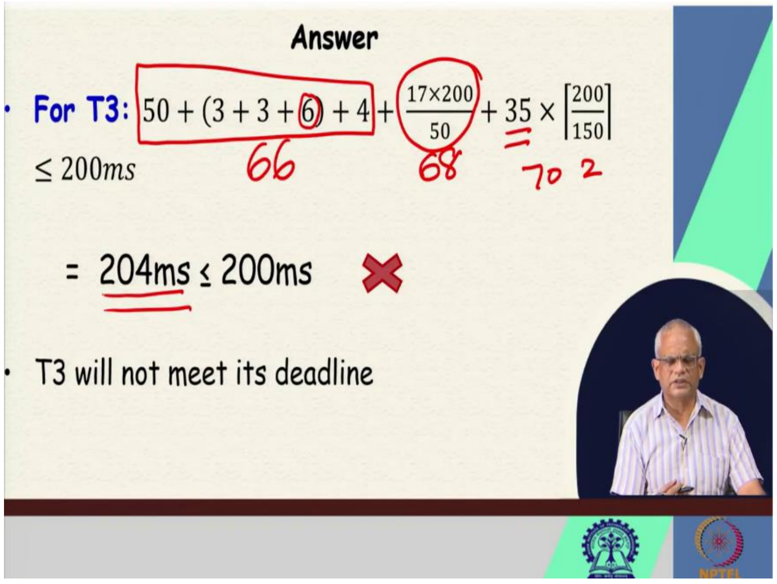
Answer

- The tasks are already ordered according to their priorities.
- Applying the revised RMA condition,
- **For T1:** $10 + 3 + 4 \leq 50msec$ ✓
- T1 will meet its deadline
- **For T2:** $\underbrace{\boxed{25 + (3 + 3) + 4}}_{35} + \frac{17 \times 150}{50} \leq 150ms$ ✓
- T2 will meet its deadline

So for T1 the self-suspension time is 3 and we need to consider 4 context switch times which is 4*1 and therefore it is 13 + 4 =17. Now for the T2, the self-suspension times of its own and higher priority is 6 and context switch time is 4. So the effective execution time of T2 is 25 +6 + 4 =35, but it also incurs delay due to its higher priority task whose execution time is 10 and 3 is its own self suspension time and 4 is the context switch time.

So, we have 17 here as its new execution time in 250 / 50 and still, if you compute it becomes 51 + 35 = 86. So, still it is less than 150 milliseconds it is schedulable. T1 is schedulable T2 is schedulable.

(Refer Slide time: 15:22)



Answer

- **For T3:** $\underbrace{\boxed{50 + (3 + 3 + 6) + 4}}_{66} + \underbrace{\frac{17 \times 200}{50}}_{68} + \underbrace{35}_{70} \times \underbrace{\left\lceil \frac{200}{150} \right\rceil}_{2} \leq 200ms$

  $= \underline{204ms} \leq 200ms$ ✗

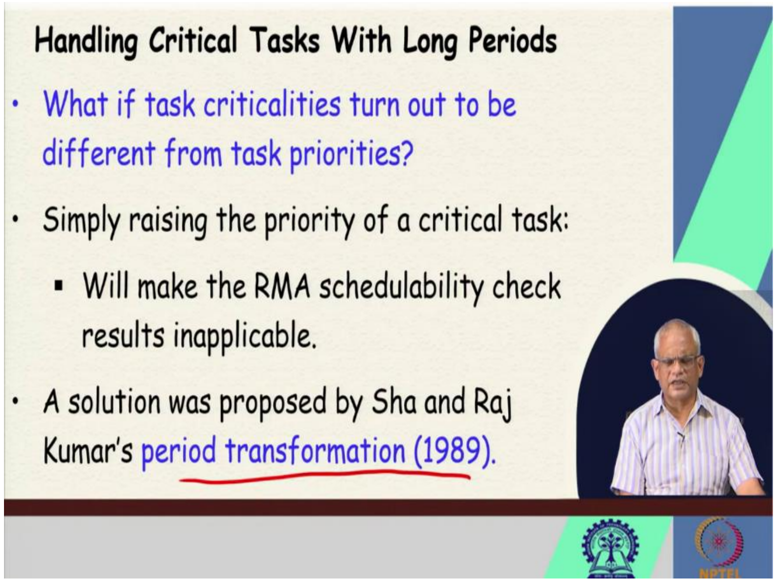
- T3 will not meet its deadline

Now, let us try to look at T3, For T3, the self-suspension time is 3 + 3 + 6, 6 is its own self - suspension time and higher priorities are 3 + 3 and context switch time is 4. So, the execution time, the effective execution time of T3 is 50 + 16 = 66. Now, the delay due to the highest priority task, is 17 is its effective execution time in the 200 / 50 = 68.

And for the second priority task, the effective execution time is 35. And ⌈ 200 / 150 ⌉ = 2 so 2*35 = 70. So, 70 + 66 + 68 = 204 > 200 milliseconds. So, the condition does not satisfy the worst case execution time is greater than deadline. So, it might miss deadline, sometimes it is just exceeding 200 millisecond.

So, T3 will not meet all its deadlines, when we consider self-suspension and context switch even though if we ignore these 2 the task set is schedulable but when we consider self-suspension alone, it is schedulable. But when we consider context switches, the task set becomes unscheduled level T3 will keep on missing its deadline.

(Refer Slide Time: 17:21)



Handling Critical Tasks With Long Periods

- What if task criticalities turn out to be different from task priorities?
- Simply raising the priority of a critical task:
  - Will make the RMA schedulability check results inapplicable.
- A solution was proposed by Sha and Raj Kumar's period transformation (1989).

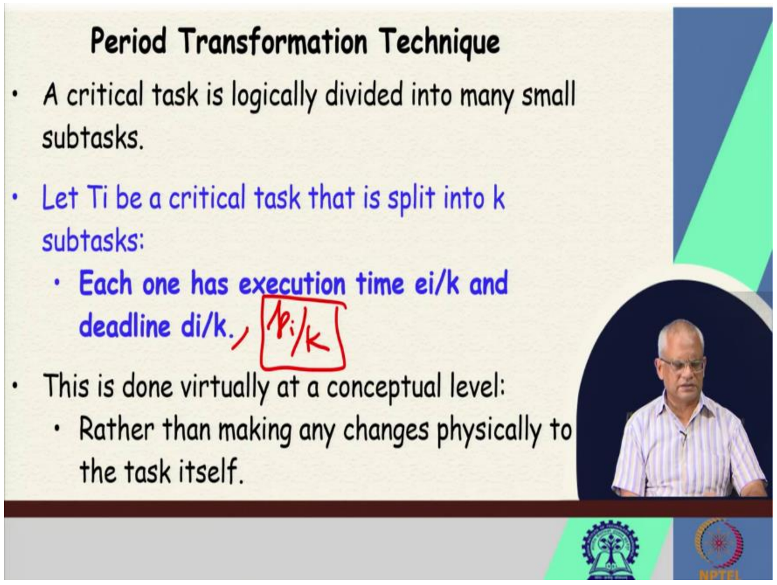
Now, we have one more issue to consider, how we handle critical tasks with long periods because the rate monotonic scheduler, for this we need to assign priorities of tasks based on their periods. So, it may so happen that a low priority task has a very small period and that gets a higher priority than a critical task.

It can so happen that the critical task is the lowest priority and the non-critical tasks have got all higher high priority values. And in that case, we know there is a danger that even if the task set as it is, is schedulable. But if one of the low priority tasks here that is non-critical task gets delayed some reason, then the critical task may miss its deadline. So, how do we handle this situation?

The order in which task priorities have been assigned, is different from the criticalities of the task, the most critical task has got a very low priority just because it had a long period. And we cannot just simply assign a high priority to this. That will violate the basic assumption of the rate monotonic scheduler that the priorities have been assigned according to the task periods.

And many of our analysis results might fail that some task which is schedulable with a readjusted priority, we just artificially inflated the priority of the critical task and then the task set may start missing its deadline. How do you handle this situation? The solution was given by Shah and Raj Kumar, in their period transformation technique in their paper in 1989. Now let us see what is this period transformation technique?

(Refer Slide Time: 19:50)



Period Transformation Technique

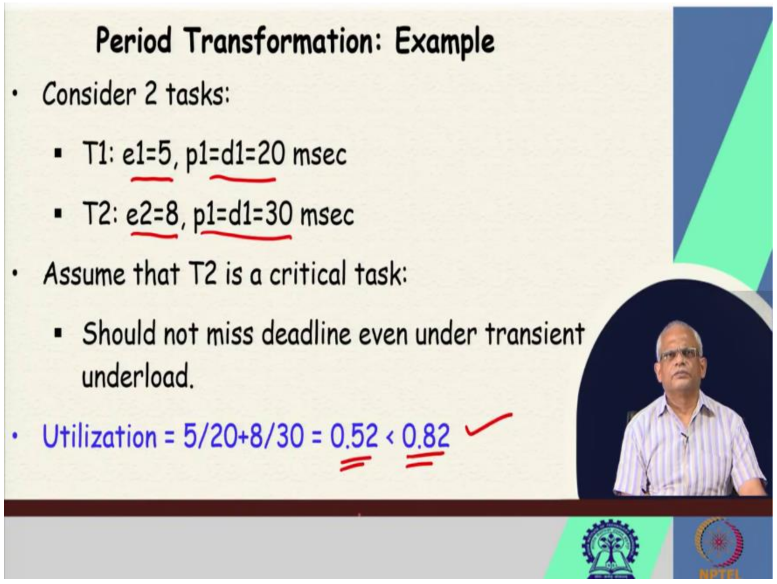
- A critical task is logically divided into many small subtasks.
- Let Ti be a critical task that is split into k subtasks:
  - Each one has execution time ei/k and deadline di/k, $\boxed{p_i/k}$
- This is done virtually at a conceptual level:
  - Rather than making any changes physically to the task itself.

So here if we know that there is a critical task Ti whose period pi is long, then we split the task into k equal parts with the execution time of ei / k and deadline of di / k and period of pi / k. We do not really split; we do not change the program. This is only for our analysis, that we change the critical task to consist of k sub tasks.

This is to check whether the task set is schedulable, and so on and if we decrease its period by pi / k. So, if k is 4, we reduce the period by 4 times and now the task will get high priority. And if we are not happy with the high priority pi / 4, we might make it pi / 8, let us say pi / 10.

And then we see that whether it is got sufficiently high priority. Now let us see what is the impact of the schedulability, when we do this kind of adjustment is a virtual adjustment. Just to handle critical tasks, no change to the program occurs is a virtual concept just to help with our analysis.

(Refer Slide Time: 21:47)



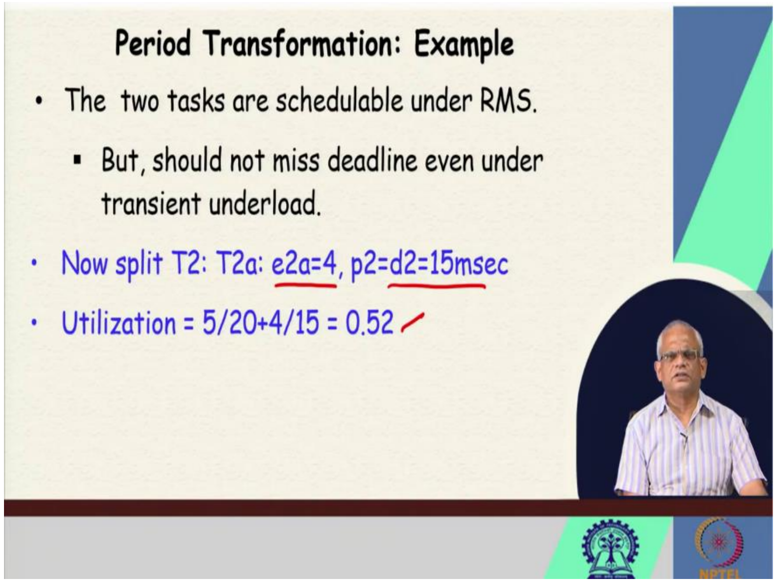
Period Transformation: Example

- Consider 2 tasks:
  - T1: e1=5, p1=d1=20 msec
  - T2: e2=8, p1=d1=30 msec
- Assume that T2 is a critical task:
  - Should not miss deadline even under transient underload.
- Utilization = 5/20+8/30 = 0.52 < 0.82 ✓

We will just consider a very simple situation, we have only 2 tasks, T1 is execution time is 5, and its period and deadline are 20. And have a task T2 whose execution time is 8, but the period and deadline are 30. But it turns out that T2 is more critical than T1. This is not a good situation, because T2 has a lower priority than T1. And if T1 gets delayed due to some reason, then T2 will miss its deadline.

And non-critical tasks getting delayed will make a critical task miss its deadline. And we do not want this situation to happen. So, we will have to raise the priority of T2. And we will use the Shah and Raj Kumar period transformation result. So first, let us consider the utilization. The utilization due to the 2 tasks is 5 / 20 + 8 / 30 = 0.52. And the Liu Leyland bound is 0.82. So, 0.52 < 0.82, and therefore the task set is schedulable. And now let us apply the period transformation technique.

(Refer Slide time: 23:28)



Period Transformation: Example

- The two tasks are schedulable under RMS.
  - But, should not miss deadline even under transient underload.
- Now split T2: T2a: e2a=4, p2=d2=15msec
- Utilization = 5/20+4/15 = 0.52

We will split the task T2 into execution time of 4 and period of 15. So, there is we just reduced its period and in each period it does less computation. Now, T2 becomes the highest priority task. So, the utilization remains the same. **a**nd it is schedulable there appears to be no change on the schedulability but remember that as far as the Liu Leyland criterion is concerned there is no impact of changing the priority splitting the task T2, but if we use the Liu Lehoczky's criterion, the completion time criterion.

Now T2 will cause delay to T1. Earlier T1 was the highest priority it was not getting delayed on account of T2. Now T2 can cause delay to 1 and that may make T1 miss its deadline and also the impact of self-suspension and the impact of context switches. So, there we can find a difference when you use the video transformation technique to raise the priority of the critical task, and the important thing to notice here is that we do not violate the rate monotonic scheduling principles.

The basic assumption that the task priority is proportional to its period is not violated. And all our analysis we can do based on that and our results will hold. But, unless we do that, we just arbitrarily raise the priority of a task without changing anything else, and then our analysis results will be void. And even if we say that the task set is schedulable, it may not be.

We are at the end of this lecture. There are few other issues with the rate monotonic scheduler other than what we have seen so far. So, we will discuss those in the next lecture. We will stop here. Thank you.