Real-Time Systems

Prof. Dr. Rajib Mall

Department of Computer Science and Engineering

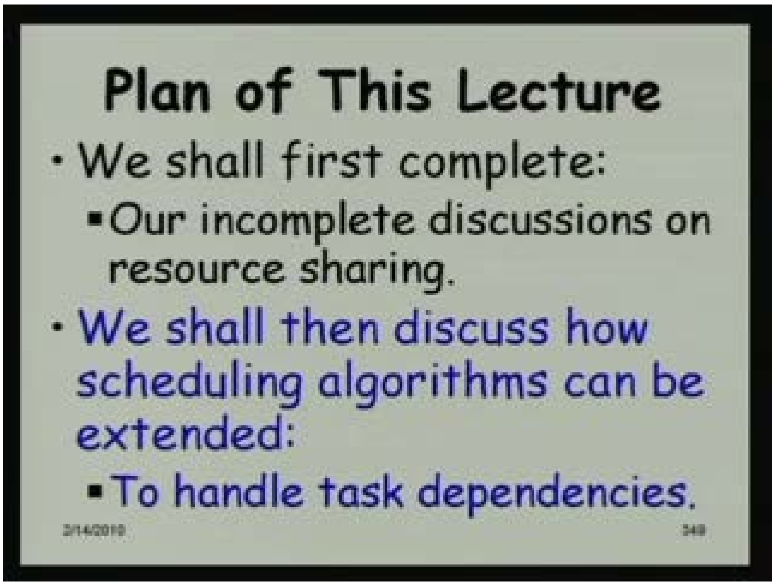
Indian Institute of Technology, Kharagpur

Lecture No. # 16

Handling Task Dependencies

Good morning. Let us get started from what we were doing last time.

(Refer Slide Time: 00:28)



Plan of This Lecture

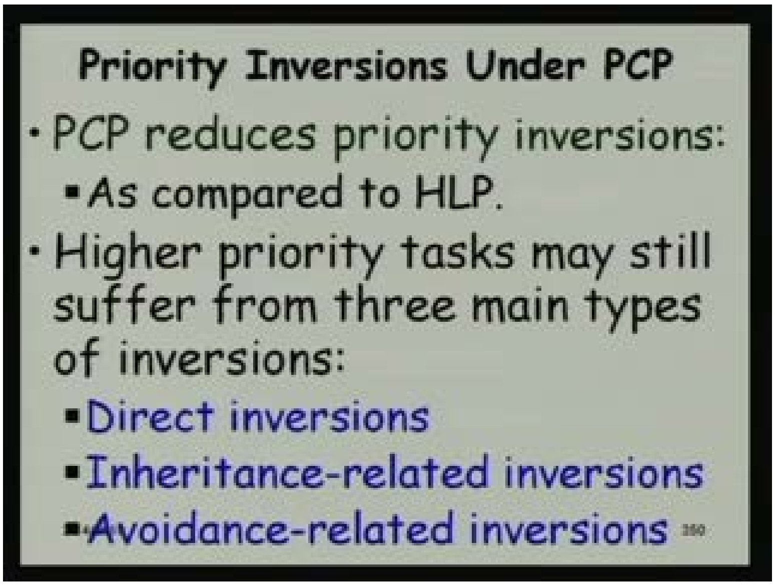
- We shall first complete:
  - Our incomplete discussions on resource sharing.
- We shall then discuss how scheduling algorithms can be extended:
  - To handle task dependencies.

2/14/2010 349

Today, we will first complete some parts of the discussion that was incomplete last time. And then, we will discuss about how the different types of scheduling algorithms that we had discussed; how they can be extended to handle task dependencies. Actually, if you remember, we had said that first we are going to discuss very simple situations, where the tasks are independent uniprocessor; and then, we are slowly bringing in the reality, where we are going to discuss first about handling the dependencies among tasks. Then, we will see what if we use multiprocessors and distributed systems, because nowadays, as you know, multiprocessors and distributed systems are becoming the norm; uniprocessors are slowly becoming obsolete. With that motivation, let us proceed today.

First, we will complete the discussions that we could not complete last time; and from then, we will go to handling task dependencies.

(Refer Slide Time: 01:36)



Priority Inversions Under PCP

- PCP reduces priority inversions:
  - As compared to HLP.
- Higher priority tasks may still suffer from three main types of inversions:
  - Direct inversions
  - Inheritance-related inversions
  - Avoidance-related inversions

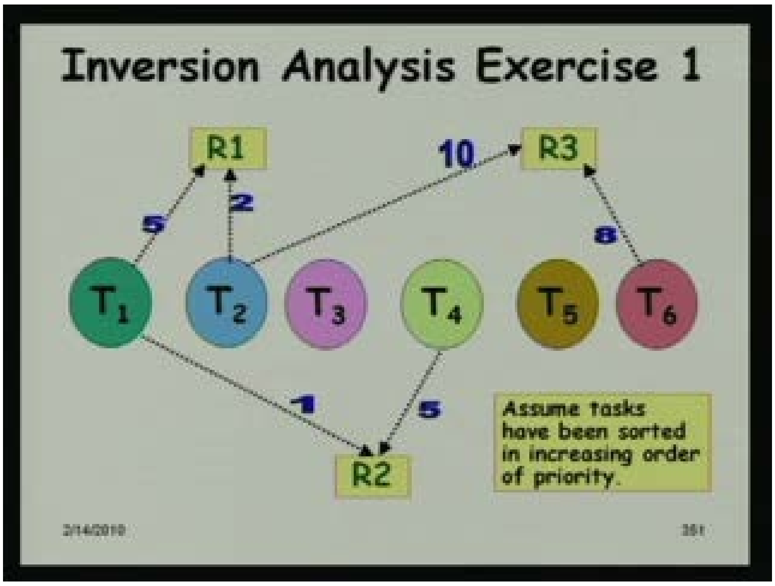
350

Towards the end of last class, we were discussing about priority inversions under the priority ceiling protocol – the different types of inversions that can occur. If you recall what we were discussing; we were discussing that, priority ceiling protocol reduces the total duration for inversions. The highest local protocol – we had remarked that it is not used, because there are too much of inversions in this; and, tasks can miss their deadlines. But, even though the PCP – ceiling protocol, it reduces the priority inversions; but, we had seen that, they still suffer from three main types of inversions. What are they? Anybody remembers? What are the three types of inversions the tasks can suffer in the priority ceiling protocol?

[Not audible] (Refer Slide Time: 02:41)

Looks like you remember. One is the direct inversion; inheritance-related inversion; and then, the avoidance-related inversion. And, we are trying to compute these inversions for an arbitrary situation. So, that was what we were doing last time. We had done some examples and also had given you one exercise to do. Now, let us do one or two more exercises, so that we understand the issues here in priority ceiling protocol, because this is the one actually used in the commercial operating systems; and, when you implement an application using a real time operating system, this is the one which you would be using.

(Refer Slide Time: 03:33)



Inversion Analysis Exercise 1

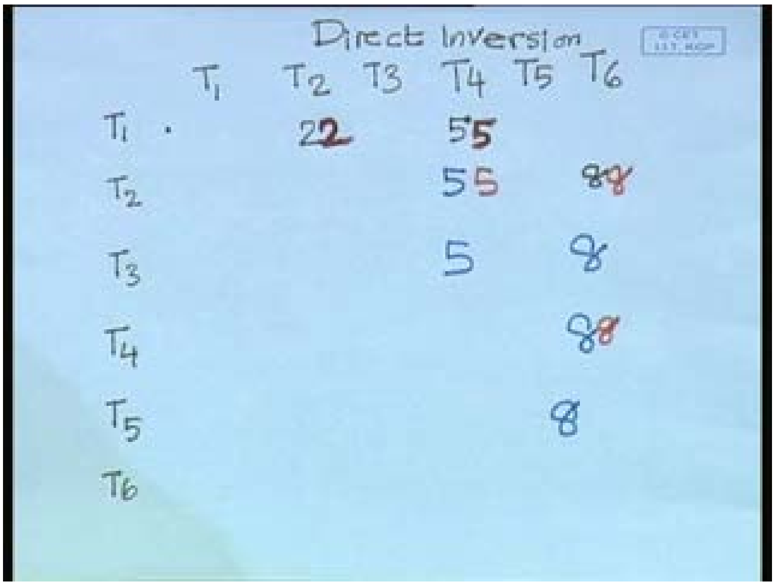
Assume tasks have been sorted in increasing order of priority.

2/14/2010

351

So, let us look at this situation. Six tasks are there; and, three resources; and, this is their sharing – the resource requirement and the duration for which they need the resource. And then, if you remember, we are preparing three types of tables: one was for the direct inversion; and the other for inheritance-related inversion; and, one for the avoidance-related inversion. What about this one? The direct inversion for T 1 – it suffers direct inversion due to which tasks? I can write the table also. Maybe it is a good idea to write the table, so that everybody understands that.

(Refer Slide Time: 04:38)



Direct Inversion

	T1	T2	T3	T4	T5	T6
T1	·	22		55		
T2				55		88
T3				5		8
T4						88
T5						8
T6						

We have six tasks: T 1, T 2, T 3, T 4, T 5, T 6; T 1, T 2, T 3, T 4, T 5, T 6. Now, how much direct inversion does T 1 suffer on account of any other task? Let us first identify that by looking at the diagram; let me just also give the title of the table as the direct inversion table. Please tell me how much inversion does T 1 suffer from other tasks?

(( )) T 2.

It suffers from T 2; is it? For what duration? 2. As I can see, T 1 suffers… It can suffer inversion due to T 2 for 2 milliseconds. And, any other task?

T 4

T 4. So, T 4 how much? What is the duration?

[Not audible] (Refer Slide Time: 05:50)

Let me just write 5 milliseconds. Now, let us look at the diagram again. Please look at what are the inversions that T 2 suffers on account of the other tasks; and then, we will populate this table. Let us wait for the diagram to come up. The diagram has come up here (Refer Slide Time: 06:15). T 1 suffers inversion due to R 1 and R 2. And, the tasks involved are T 2 and T 4. T 2 causes inversion for 2 milliseconds; and, T 4 for 5; that is what you had said. What about T 2?

[Not audible] (Refer Slide Time: 06:34)

T 2 due to T 6 for 8. And, let me just write down. T 2 for T 6 for 8 milliseconds. Now, let us look at the diagram again. Let us identify the inversions due to T 3. Will T 3 suffer any direct inversion? No. What about T 4?

No.

No. What about T 5?

No.

No. T 6? No. So, only T 1 and T 2 suffer direct inversions. So, those two only we have populated. And, the rest we do not write anything; those are 0. Is everybody comfortable with this?

Now, let us try to find out the inheritance-related inversion (Refer Slide Time: 07:25). I will use the same table, maybe with a different pen. We will just write down near that. Let us find out the inheritance-related inversion. Does T 1 suffer any inheritance-related inversion? What is the definition of inheritance-related inversion? Anybody would like to answer? What do you mean by inheritance-relate… How does that occur? How does inheritance-related inversion occur?

A lower priority task gets the…

A lower priority task has the resource. And then?

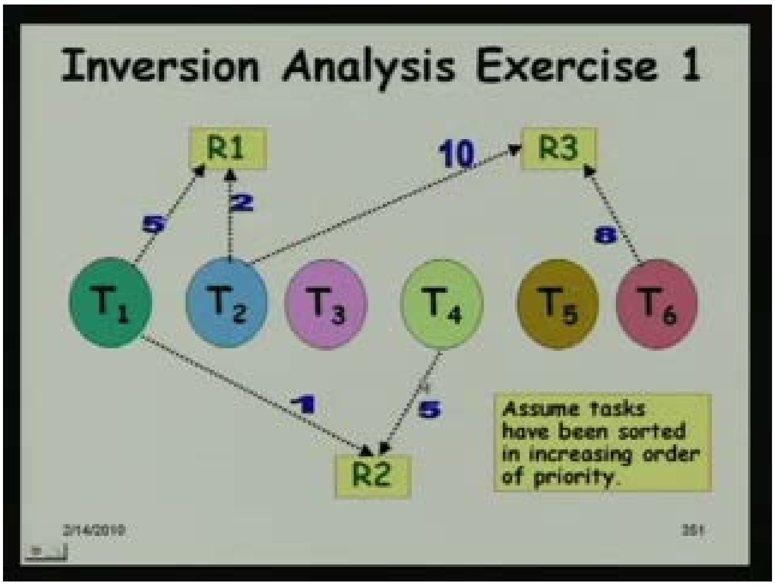
Intermediate task will block the (( ))

Another task is waiting for the resource; is it? And then, the priority of the task holding the resource goes up by the inheritance clause of the priority ceiling protocol.

But, the intermediate tasks do not require the resource (( ))

Yeah. No, that is inheritance. So, that is the inheritance-related inversion; the intermediate tasks are getting clubbed. Here for T 1 to get blocked one task with at least as much priority as T 1 should be requested the resource. And, T 1 itself cannot wait for itself. It cannot undergo inversion. So, T 1 cannot undergo inheritance-related inversion. Yes or no? So, T 1 does not undergo any inheritance-related inversion. What about T 2?

(Refer Slide Time: 09:14)



Inversion Analysis Exercise 1

Assume tasks have been sorted in increasing order of priority.

2/14/2010

351

Look at the diagram on the screen. Will T 2 undergo an inheritance-related inversion?

Yes (( ))

Yes.

[Noise - Not audible]

So, it will undergo inheritance-related inversion due to T 4. So, T 4 is holding the resource; T 1 is waiting; and, T 2 can undergo an inheritance inversion for 5 milliseconds. And, any other cases it can undergo inversion? No. What about T 3?

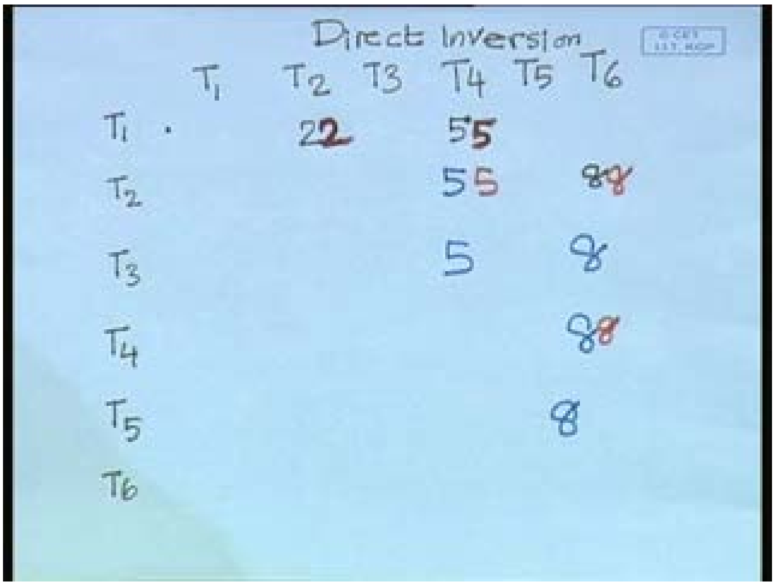
[Not audible] (Refer Slide Time: 09:54)

T 3 will undergo inversion due to T 4 and T 6. And, what about T 4?

T 6.

T 4 due to T 6. And, what about T 5? T 5 also due to T 6.

(Refer Slide Time: 04:38)



Direct Inversion

	T1	T2	T3	T4	T5	T6
T1	·	22		55		
T2				55		88
T3				5		8
T4						88
T5						8
T6						

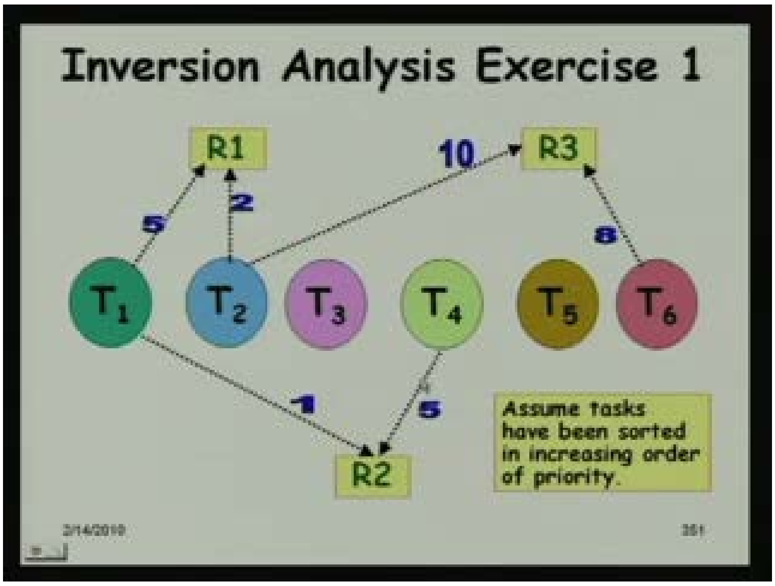
Let me just write that down. You said that T 1 does not undergo any inversion; and, T 2 undergoes inversion on account of T 4. And, that is for 5 (Refer Slide Time: 10:28). And, T 3 undergoes inversion on account of T 4 and T 6 for 5 and 8. And, T 4 undergoes an inversion due to T 6 for 8. And, T 5 – it can undergo inheritance inversion on account of T 6 for 8. Does everybody agree with these figures?

Now, let us look at the avoidance inversion. Please observe the diagram again – the task figure. And, let us try to find out the avoidance inversion. Let the… It has come up (Refer Slide Time: 11:13). Will T 1 undergo any avoidance inversion? Please think over. What is the avoidance inversion? How does avoidance inversion occur? Anybody would like to answer how does avoidance inversion occur? Anybody remembers how does avoidance inversion occur?

Sir, the task is not granted immediately access to resource (( ))

This occurs due to the ceiling value. There is a system ceiling; remember? There is a current system ceiling. And, whenever a task acquires a resource, the system ceiling is set equal to the ceiling of that resource. And, whenever a task requests another resource, the task's priority is checked against the current system ceiling. Only if its priority is greater than the current system ceiling or it is the task, which has set the ceiling, which last acquired the resource due to which the ceiling was set. Under these conditions, the resource is granted. If the tasks priority is less than the system ceiling and it is not the task, which last set the ceiling, resource will be refused even if the resource is available. And, we had seen that this is to prevent deadlocks.

(Refer Slide Time: 09:14)



Inversion Analysis Exercise 1

Assume tasks have been sorted in increasing order of priority.

2/14/2010

351

Let us look at the figure again – the task figure; and, try to find out (Refer Slide Time: 12:54) the avoidance inversions. Will T 1 undergo avoidance inversion?

Yes.

How will T 1 undergo avoidance inversion? Just tell me the situation. Some of you are saying yes. Let us hear how T 1 will undergo avoidance inversion.

Sir, suppose T 6 is accessing the resource R 3.

T 6 is accessing. So, R 3 is ceiling value, will be set; system ceiling; no problem; this will be lower than T 1. That is not the situation. How will T 1 undergo avoidance inversion?

T 2 tries to acquire (( )) (Refer Slide Time: 13:44)

Since no one is answering, let me just refresh your memory. See when T 4 acquires R 2, the ceiling will be set, is equal to the priority of T 1, because that is the ceiling of R 2. And, now, let us say T 1 is requesting for R 1. It would not be granted; otherwise, there can be deadlock situation. Just listen to the situation. T 4 is already holding R 2. And, that time, T 1 requests R 1; it will be refused.

Sir, when T 4 is accessing R 2, the ceiling value is set (( )) that of T 1.

T 1, yes.

(( )) system ceiling is set by task T 1.

No, it is set by T 4. On account of T 4 holding the resource, the ceiling has been set. So, T 4 will be allowed to access another resource, not T 1.

Not T 1.

Not T 1. But, what about T 1 requesting R 2? Will that be avoidance inversion? No, it would not be; it is a direct inversion. So, T 1 requesting R 1; for T 4, T 1 can undergo avoidance inversion for how much duration?

5.

5. Will it undergo avoidance inversion due to any other task? Yes, due to T 2. Due to T 2, because just look at the situation; T 2 is already holding R 1 and T 1 might be requesting R 2. And, in that situation, it will undergo an inversion for 2 milliseconds. This is avoidance inversion. What about T 2? Will it undergo avoidance inversion?

[Not audible] (Refer Slide Time: 15:45)

It can undergo avoidance inversion on account of T 4, because T 4 might be holding the resource R 2; and, T 2's request for R 1 will be rejected or it has to wait, because the ceiling value is set to 1. So, the duration will be 5. What about any other…

T 6 (( ))

Yeah, on account of T 6. So, T 6 might be holding R 3 and T 2 might be requesting R 1. And, in that case, it will undergo for 8 milliseconds. What about T 3? Will it undergo avoidance inversion? No, it does not need any resource. So, it cannot undergo an avoidance inversion, because it will not request for any resource. Only when the request for resource is denied due to the ceiling value, it will undergo inversion.

What about T 4? Will it (Refer slide Time: 16:58) undergo inversion due to T 1?

No.

No, because T 1 is already higher priority and T 4 does not need any other resource. But, what about…

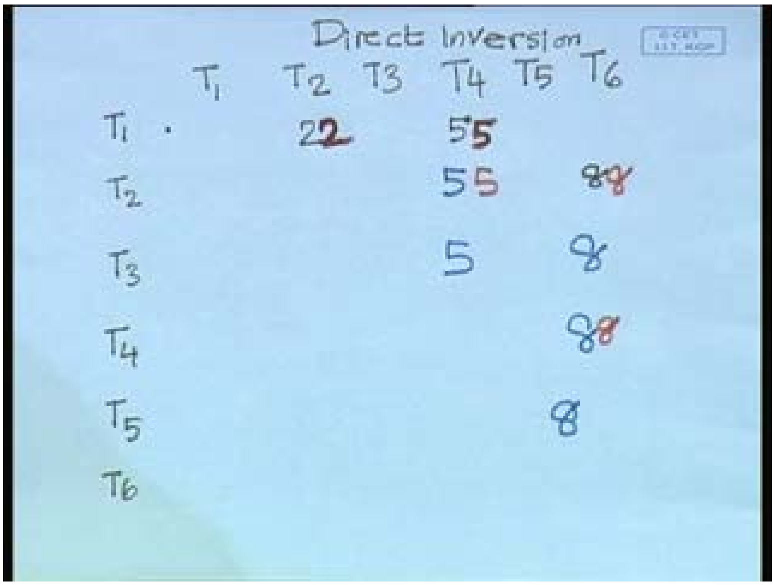
[Not audible]

No; but, what about for other tasks? It would not undergo for T 1. But, will T 4 undergo inversion due to any other task?

(( )) T 6.

Yes, it will undergo avoidance inversion due to T 6, because as T 6 acquires R 3, the system ceiling will be set to that of T 2 to be equal to 2; and, T 4 – if it requests, R 2 will be denied and it has to wait for 8 milliseconds maximum. What about T 5? It will not undergo inversion, because it does not need resource. What about T 6? No, T 6 is the lowest priority. A lowest priority task does not undergo any inversion.

(Refer Slide Time: 04:38)

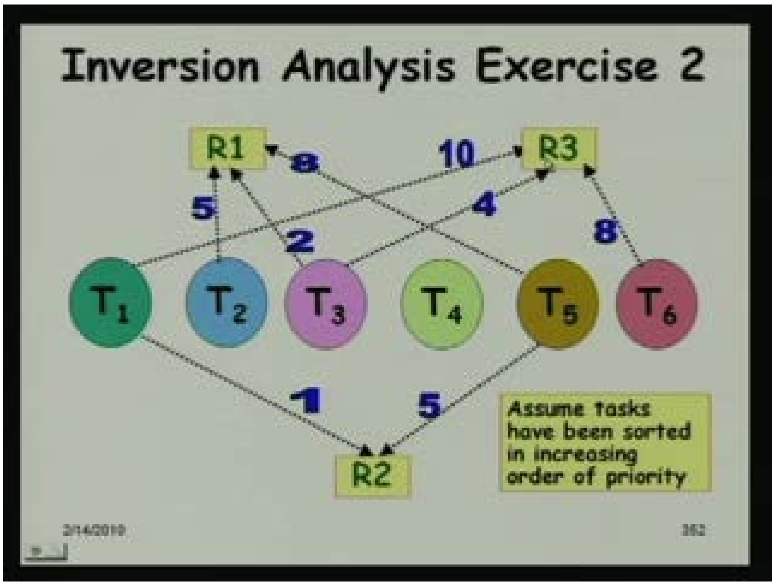


Direct Inversion

	T_1	T_2	T_3	T_4	T_5	T_6
T_1		22		55		
T_2				55		88
T_3				5		8
T_4						88
T_5						8
T_6						

Let me just write down the observations. You said the (Refer Slide Time: 18:11) T 1 undergoes inversion on account of T 4 and T 2; on account of T 4 for 5; and, on account of T 2 for 2. Let me use a different color; I am just using the blue. The blue is for inheritance inversion and the red is for avoidance inversion. And, T 2 undergoes avoidance inversion on account of T 4 and T 6. So, T 4 – it will undergo for 5; and, T 6 for 8. And, T 3 does not undergo any avoidance inversion; does not need any resource. And, T 4 will undergo on account of T 6 for 8. And, T 5 does not undergo any avoidance inversion. So, now, a task we had seen can undergo at most one inversion. And, T 1 – the maximum duration we just look through the row and found that 5 is the maximum duration for which we can undergo inversion T 2 for 8, T 3 for 8, T 4 for 8, T 5 for 8, T 6 is 0.

(Refer Slide Time: 19:41)



Inversion Analysis Exercise 2

Assume tasks have been sorted in increasing order of priority

2/14/2010

352

Let us proceed from here. Is this clear, how do we calculate the inversions? Now, let us look at the… I think there is one more exercise here much more complicated. That one with huge resource requirements; large number of resource requirement for different tasks. But, if you have understood how you computed for the previous exercise, you will be able to do it. Would you like to work it out here or do you think that it is OK. Anybody would have difficulty working out, then we can discuss; otherwise, we can just keep and give you as an exercise. What do you think? Should we try it out here?

Let me not write it at least; let me just ask you, because that is quick. What about T 1? Will it undergo direct inversion? What about direct inversion for T 1?

[Not audible] (Refer Slide Time: 20:33)

T 1 would not undergo direction inversion due to T 3. See T 1 – direct inversion; they are sharing the same resource. So, due to T 5 and T 6, T 1 can undergo direct inversion. What about…

[Not audible]

Direct inversion – T 1 can undergo direct inversion due to T 3 for 4 and on account of T 5 for 5 and on account of T 6 for 8. What about T 2? On account of T 3 and T 5. What about T 3? On account of T 6; on account of T 2, it would not undergo inversion, because this is a higher priority task.

[Not audible]

T 3 will undergo inversion due to T 6 and also due to T 5 on account of R 1 sharing. So, due to T 5 and T 6. T 4 will not undergo any direct inversion. It does not need any resource. What about T 5? It cannot undergo any direct inversion. It is not sharing resource with any lower priority task. Similar is T 6.

Now, what about inheritance inversion? Will T 1 undergo inheritance-related inversion? No, it is the highest priority and we cannot have a situation; another task is waiting; another task with higher priority waiting. What about T 2?

[Not audible] (Refer Slide Time: 22:18)

It will undergo due to T 5 and also T 3 and also T 6. What about T 3? Will it undergo inheritance inversion?

T 5 (( ))

Due to T 5, yes; and also… No. Will it undergo T 6? Yes, T 1 is sharing with T 6. So, on account of both T 5 and T 6 can undergo inheritance inversion. What about T 4? It will undergo due to both T 5 and T 6. What about T 5? Will it undergo inheritance inversion? Yes, due to T 6.

And, let us look at the avoidance inversion (Refer Slide Time: 23:13). Will T 1 undergo avoidance inversion? Yes or no?

Yes sir.

Yes, it will undergo inversion. So, which tasks?

Sir, T 6 (( ))

Due to… it can have inversion on account of T 3, on account of T 5 and on account of T 6. So, once they acquire a resource, it cannot even… Once T 3 acquires R 3, and T 1 request for R 2 will be rejected. Similarly, for T 5 acquires R 2, T 1 request for R 3 will be rejected. Similar is with T 6.

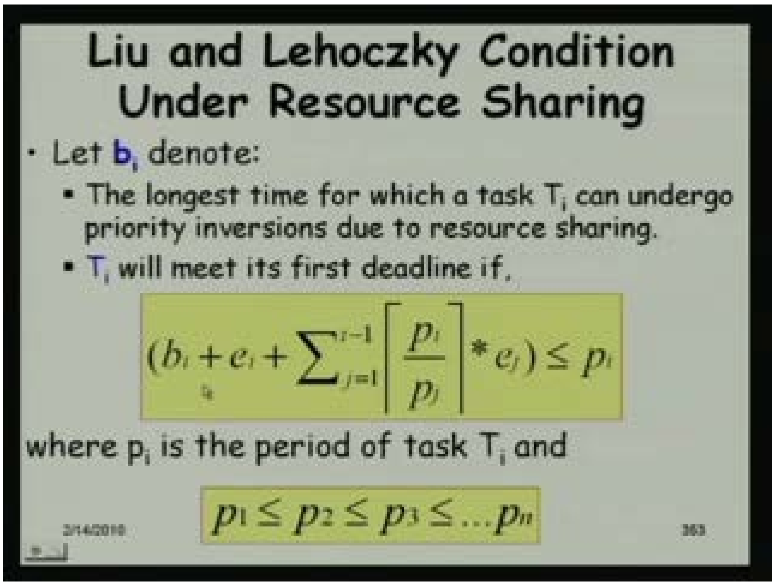
Now, what about T 3? Will it undergo avoidance inversion? On account of which tasks?

[Not audible] (Refer Slide Time: 24:16)

On account of T 5, we would always have to look for the lower priority tasks. On account of T 5 acquiring R 2 or T 5 acquiring R 1 – any of these cases, it will undergo an inversion; and also, for T 6 acquiring R 3.

Let us proceed from here. Looks like you are comfortable determining the different types of inversions under the priority ceiling protocol.

(Refer Slide Time: 24:54)



Liu and Lehoczky Condition Under Resource Sharing

- Let b_i denote:
  - The longest time for which a task T_i can undergo priority inversions due to resource sharing.
  - T_i will meet its first deadline if,

$$(b_i + e_i + \sum_{j=1}^{i-1} \left\lceil \frac{p_i}{p_j} \right\rceil * e_j) \le p_i$$

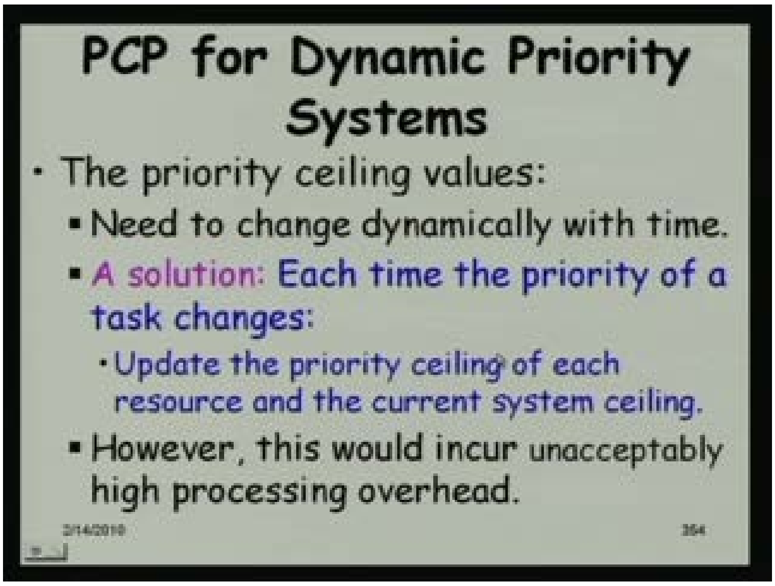
where p_i is the period of task T_i and

$$p_1 \le p_2 \le p_3 \le \ldots p_n$$

2/14/2010 353

Now, once we have found out the longest duration for which a task can undergo priority inversion, that is, the highest entry in the corresponding row of the task, that is, b i, we will have to modify the Liu and Lehoczky condition. We add b i also here, because that is the maximum inversion it can undergo. And then, if still the task meets its deadline, then the task will be schedulable. If b i is too much, then it may not be schedulable.

(Refer Slide Time: 25:42)



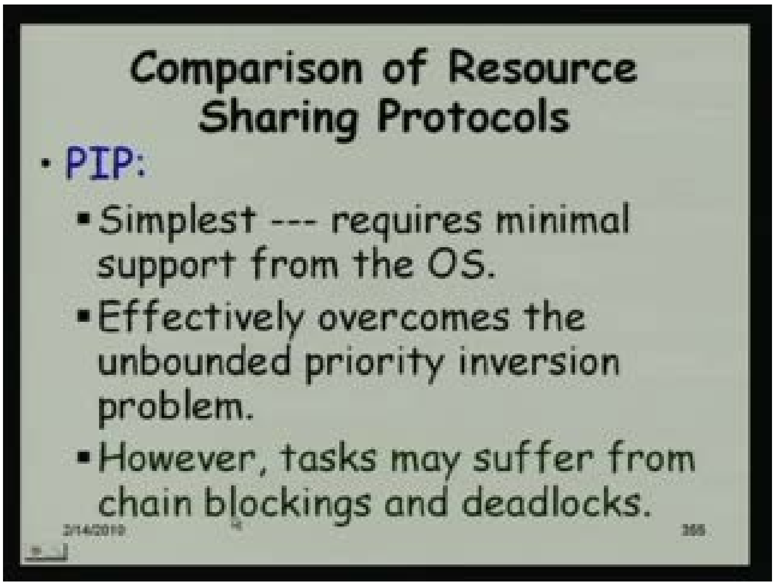
PCP for Dynamic Priority Systems

- The priority ceiling values:
  - Need to change dynamically with time.
  - A solution: Each time the priority of a task changes:
    - Update the priority ceiling of each resource and the current system ceiling.
  - However, this would incur unacceptably high processing overhead.

2/14/2010 354

Now, you have seen that how priority ceiling protocol can be extended or the analysis of the priority ceiling protocol can be done to check whether a set of tasks will meet their deadline or miss their deadline based on their resource sharing requirement. But, what about a dynamic priority system like EDF. Can PCP – a protocol such as a priority ceiling protocol be used for dynamic priority system? See if we have to use a protocol like priority ceiling protocol, we will have to compute the priority ceiling value for every resource. And, we know that, the virtual priority of task keeps on changing and we have to update the ceiling for each resource, and also, the current system ceiling. And, if we try to do that, it will incur unacceptably high processing overhead and we cannot use resource sharing under dynamic priority system; it is extremely difficult. And, this is one of the main reasons why EDF is not used. We had seen some examples, where the rate monotonic algorithm is used. Later, as we proceed, we will see more and more examples, where the rate monotonic algorithm is used for task scheduling even though it is less proficient than EDF.

(Refer Slide Time: 27:19)



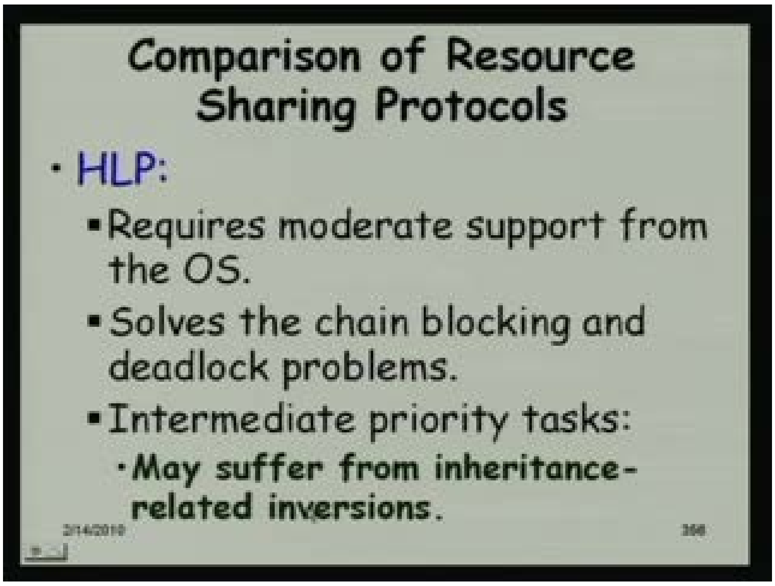
Comparison of Resource Sharing Protocols

- PIP:
  - Simplest --- requires minimal support from the OS.
  - Effectively overcomes the unbounded priority inversion problem.
  - However, tasks may suffer from chain blockings and deadlocks.

2/14/2010 355

Now, let us just compare the different resource sharing protocols. Recollect that, we had said that, the basic inheritance scheme; the priority inheritance protocol is simplest, requires minimal support from the OS; only we need to reset the priority value of the task when another task is waiting for the resource. This is the only requirement. And, it could overcome the unbounded priority inversion problem. But, tasks can suffer from chain blocking and deadlocks. These are severe problems.

(Refer Slide Time: 28:01)



Comparison of Resource Sharing Protocols

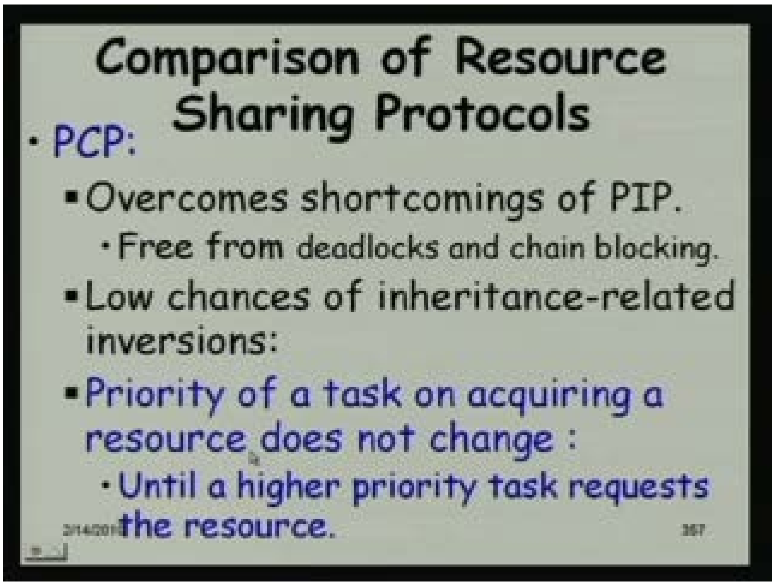
- HLP:
  - Requires moderate support from the OS.
  - Solves the chain blocking and deadlock problems.
  - Intermediate priority tasks:
    - **May suffer from inheritance-related inversions.**

2/14/2010

398

And, we had discussed about the highest local protocol – requires moderate support, where we set ceiling for resources; keep track of all ceiling of all resources. And then, as the task acquires a resource, its priority is changed to that of the ceiling priority. And then, after it releases the resource, the priority is reset. It solves the chain blocking and deadlock problems. But, the intermediate priority tasks can suffer from inheritance-related inversions. And, since the task is raised to a high value, the tasks suffer from inheritance-related inversions frequently. And this can lead to deadline misses, because there are frequent inversions.

(Refer Slide Time: 28:53)



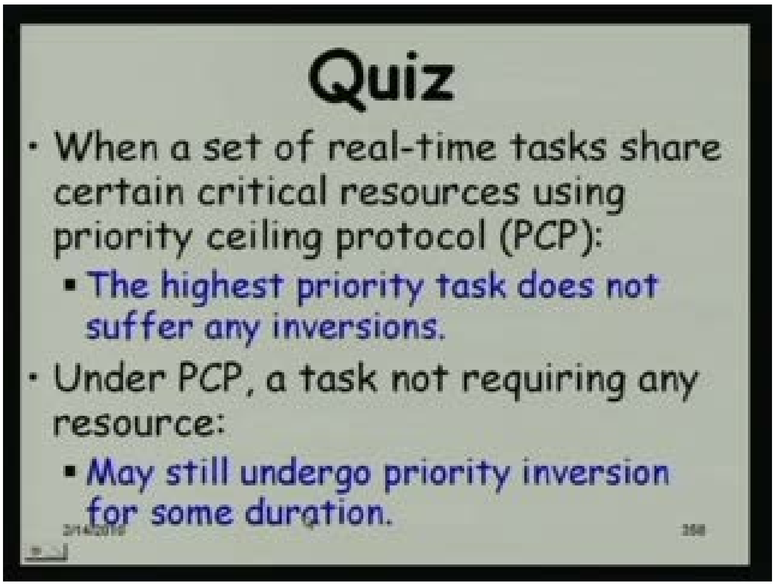
Comparison of Resource Sharing Protocols

- PCP:
  - Overcomes shortcomings of PIP.
    - Free from deadlocks and chain blocking.
  - Low chances of inheritance-related inversions:
  - Priority of a task on acquiring a resource does not change :
    - Until a higher priority task requests the resource.

2/14/2010 357

And, we had seen an improvement over the HLP form of priority ceiling protocol – overcomes the short comings of priority inheritance protocol. It is free from deadlocks and chain blocking. And also, it incurs lower inheritance-related inversion as compared to the priority ceiling protocol. The main reason for why it incurs low inheritance inversion is that the priority of a task does not change when it acquires the resource. So, intermediate priority tasks would not get affected on less of course; a higher priority task is waiting for the resource.

(Refer Slide Time: 29:36)



Quiz

- When a set of real-time tasks share certain critical resources using priority ceiling protocol (PCP):
  - The highest priority task does not suffer any inversions.
- Under PCP, a task not requiring any resource:
  - May still undergo priority inversion for some duration.

2/14/2010 358

Now, let me just ask few simple questions just to check whether you are following what we are doing. Now, just listen to this statement – when a set of real time tasks share certain critical resources using the priority ceiling protocol, the highest priority task does not suffer any inversions. Do you agree with this? Yes or no?

It will suffer direct inversion (( ))

It suffers direct inversion. But, any other types of inversion?

Avoidance.

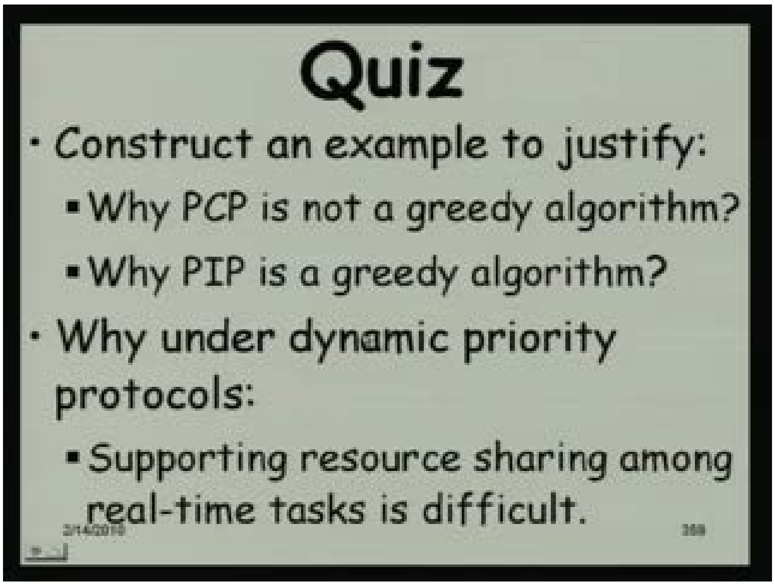
It can also suffer from avoidance inversion. It may not suffer from inheritance inversions.

Now, let us look at this statement – under priority ceiling protocol, a task not requiring any resource may still undergo priority inversion for some duration. Do you think this is a correct statement?

Yes, (( ))

It can still undergo inheritance-related inversion. What about avoidance inversion? No, it would not; only inheritance-related inversion it can still undergo.

(Refer Slide Time: 30:48)



Quiz

- Construct an example to justify:
  - Why PCP is not a greedy algorithm?
  - Why PIP is a greedy algorithm?
- Why under dynamic priority protocols:
  - Supporting resource sharing among real-time tasks is difficult.

2/14/2010 358

Now, this is just a thought question. Why is it that the priority ceiling protocol is not a greedy algorithm; whereas, inheritance protocol is called as a greedy algorithm? What do we think? Can you give an example or something why ceiling protocol is not a greedy algorithm?

Sir, the problems we worked out (( ))

Yes.

The situations were avoided (( )) the avoidance-related inversions occurring. Sir, from there we can argue that it is not a greedy algorithm (( )) because…

Anybody has any other idea? See he gave an answer; I will just tell that answer. But, what about any other answers? Anybody would like to give any other answer?

Actually, we are not increasing the priority of the task (( ))

No. See this is a question of whether it is a greedy or not greedy algorithm; it has nothing to do with priority. It is the way the algorithm operates. Can we call it as a greedy algorithm? If we cannot call it as a greedy algorithm, just give an example to justify, why it is not greedy. Not very hard to think actually. See a task will be blocked even though the resource that it is requiring is available. No one is using that resource, but still the task will be blocked. If it was a greedy algorithm, the resource is available and its priority is OK, etcetera, then just give it the resource. But, here the task is having higher priority than the current one that is running and the resource is available; but, still the task will be blocked. That is due to the avoidance related; overcome the avoidance inversion like he was answering. So, it is not a greedy algorithm, because when the resource is available, it still does not grant. So, just think about it.

Now, why under dynamic priority protocol, supporting resource sharing among real time tasks is difficult? What problems would occur if you are using a dynamic priority protocol such as an EDF – earliest deadline first; and then, we want the tasks to share critical resources? What kind of problem will occur?

Inheritance-related…

Sir, only the (( )) change and then we do not know. Suppose we are assigned a lower priority…

Exactly. There will be too many inversions, because implementing a protocol to minimize the inversion is very difficult, because the ceiling value itself changes dynamically. The task priorities change and the ceiling values change and we cannot think of a protocol; nobody has thought of a protocol, which can be used to reduce the inversions. So, it is unusable when the tasks have to share critical resources.

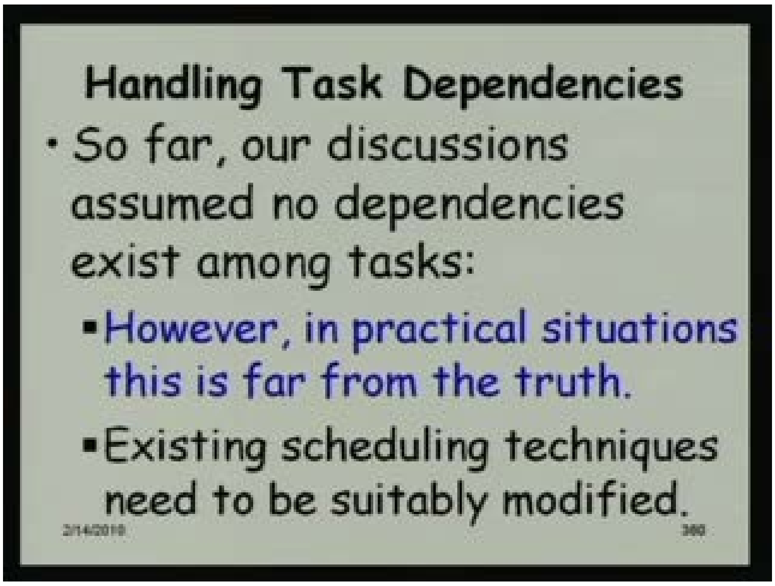
Now, let us proceed from that point.

It has been dynamic (( )) Is not there any sharing at all?

It is very difficult. See if the system is loaded, if a task rarely executes, then we can think that see that task may not cause inversions to other tasks. Now, let us say the task runs frequently and it is also using critical resources. It can cause inversions to other tasks under EDF. And, if the deadline is close, it is milliseconds or something rather than minutes, then it will become unusable. The tasks will undergo inversions for long duration and there can be deadline misses. And, if the consequence of deadline misses are not acceptable, then which is true for most hard real time systems, then we cannot use it.

Let us proceed from that point.

(Refer Slide Time: 35:07)



Handling Task Dependencies

- So far, our discussions assumed no dependencies exist among tasks:
  - However, in practical situations this is far from the truth.
  - Existing scheduling techniques need to be suitably modified.

2/14/2010

360

Now, let us try to bring in this complicacy. So far, we assumed that the tasks are independent. Which is far from the truth in most practical situations? In practical situations, tasks do have dependencies; unless something completes, another task cannot start. And, we need to modify the existing algorithms; fortunately, not very difficult; it is not too much of a complication. Let us see how to handle that.

(Refer Slide Time: 35:42)

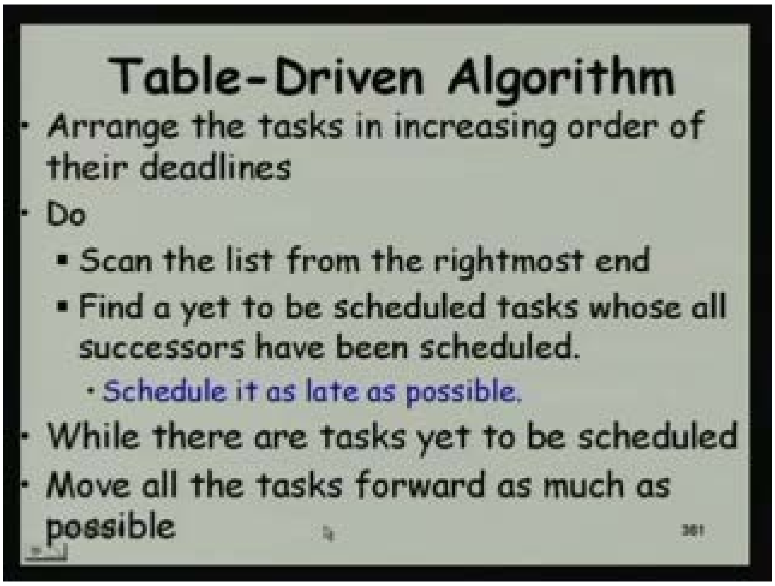


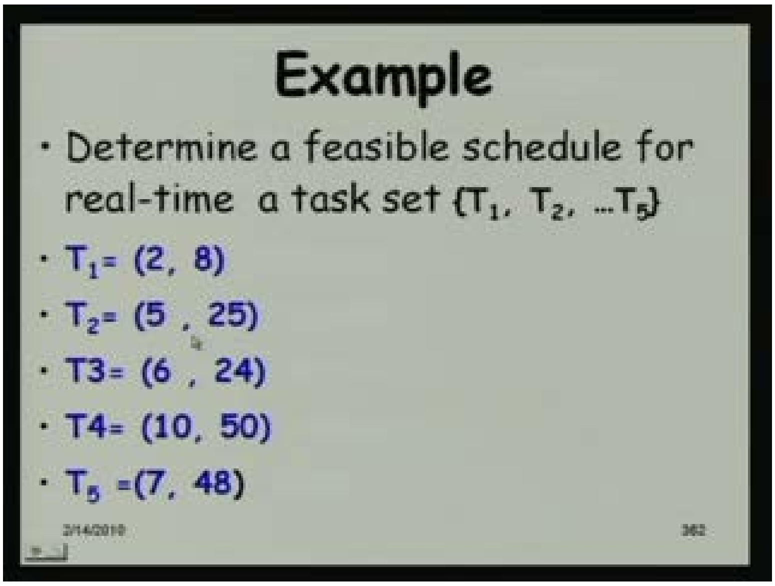
Table-Driven Algorithm

- Arrange the tasks in increasing order of their deadlines
- Do
  - Scan the list from the rightmost end
  - Find a yet to be scheduled tasks whose all successors have been scheduled.
    - Schedule it as late as possible.
- While there are tasks yet to be scheduled
- Move all the tasks forward as much as possible

361

First, let us look at our table-driven algorithm, because we had looked at the table-driven once as the clock driven algorithm as an example; and then, we had looked at the eventdriven algorithms: the RMA and EDF. So, first, let us look at the table-driven algorithm; what to do here; and then, we will look at the RMA and EDF. In table-driven algorithm, is a simple algorithm to extend the table-driven algorithm to handle task dependencies? We arrange the tasks in increasing order of their deadlines. After having done that, we scan the tasks from the right-most end and find a yet to be scheduled task, whose successors have been scheduled. And then, we schedule it as late as possible such that the task does not miss its deadline. We continue doing this while there are tasks yet to be scheduled. So, that is, the list is exhausted. And then, after finding this schedule, we move all the tasks forward as much as possible. We will just see that with the help of an example.

(Refer Slide Time: 36:53)



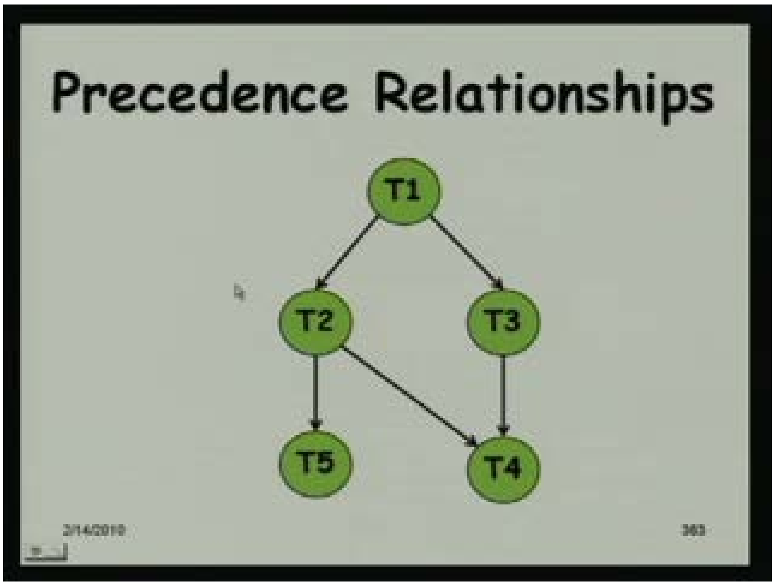
Example

- Determine a feasible schedule for real-time a task set {T_1, T_2, …T_5}
- T_1= (2, 8)
- T_2= (5 , 25)
- T3= (6 , 24)
- T4= (10, 50)
- T_5 =(7, 48)

2/14/2010 362

This is the example. We have five tasks here. And, this is the execution time T; for T 1, it is 2 milliseconds and the deadline is 8; T 2 – it is 5 and 25.

(Refer Slide Time: 37:13)



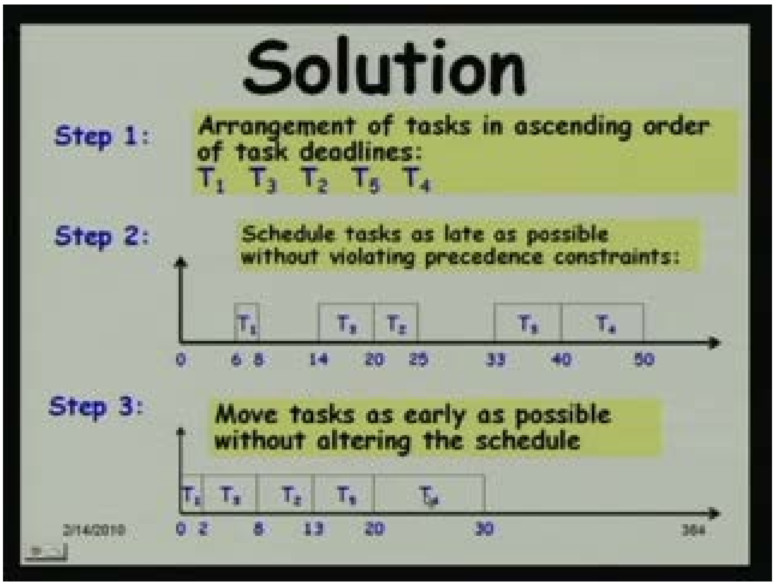
Precedence Relationships

2/14/2010

363

And, this is the precedence relationships between the different tasks. Now, let us see how we apply the algorithm.

(Refer Slide Time: 37:24)



Solution

Step 1: Arrangement of tasks in ascending order of task deadlines:
T_1 T_3 T_2 T_5 T_4

Step 2: Schedule tasks as late as possible without violating precedence constraints:

Step 3: Move tasks as early as possible without altering the schedule

2/14/2010

364

First, we arrange the tasks in ascending order of their deadlines. If we observe that task characteristics, we will see that T 4 has a longer deadline than T 5. So, this is the order that is worked out here T 1; and, T 2 has a longer deadline than T 3. So, T 1, T 3, T 2, T 5, T 4 if we arrange them in terms of their tasks deadlines. Now, here we have taken T 4; and, 50 is its deadline. So, we have scheduled it here. It runs for 10 milliseconds. Its execution time is up to 40. And, T 5 can run from 40 to 33; T 2’s deadline is 25. So, it can run from 25 to 20; 5 milliseconds is its execution time. And then, T 3 can be taken up at 20 and it can take 6 milliseconds from up to 14 to 20. And, T 1’s deadline is 8 and it takes 2 milliseconds to run. So, it can start at 6 in the worst case. These are the worst case start times for the tasks. And, still none of the tasks miss their deadlines; that is what we have worked out here.

And, the step 3 is move all these forward. This has T 1; we move it to 0 to 2; and then, from 2 to 8, T 3 has come; and 8 to 13, T 2 has come. So, we moved everything forward. And, this is the final schedule for the table-driven scheduler.

Here there is no (( ))

If you remember… See the question somebody is asking is that what about the period of the tasks? Are not we consider that? See if you remember what we were discussing in the table-driven scheduling, is that we compute the schedule for one major cycle and then keep on repeating that. So, this is for one major cycle we have computed the schedule. And then, we will keep on repeating this.

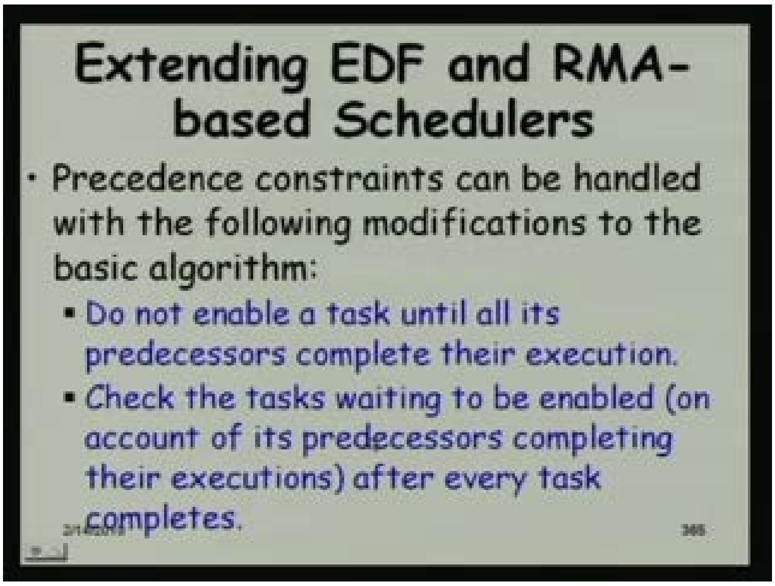
But we are changing the phase on that (( )) When we are moving it forward…

No, see here… Let me just again clarify. See he is talking about phase. But, see here we did not specify any phase. So, the tasks are assumed to arise at 0 milliseconds. So, as long as it is not scheduled before 0, then we are doing all right. If any task is having a nonzero phasing, then of course, we cannot move it before that. So, that is the complication we will have to consider when tasks have nonzero phasing. But, here we have implicitly assumed, tasks have 0 phasing.

All the periods (Refer Slide Time: 40:34) (( )) all the tasks arrive at the same time.

No, since somebody else is also asking the question, let me just revisit that problem that we are showing. See here (Refer Slide Time: 40:49) in many cases, where these task dependencies exist like this tree we have shown here, T 4 depends on the results of T 2 and T 3. So, first, T 1 needs to be executed; then T 2 and T 3; then, possibly T 5 or T 4 or T 4 and T 5. That is the order that is permissible. But, these tasks – they occur together. So, they keep on repeating. Know, unless T 1 completes, T 2 cannot complete. So, this way they will keep on repeating. So, the period for these all of them is same; maybe 200 for each one. So, every 200 milliseconds, T 1 will occur once; T 2 will occur once; T 3 will occur once. So, all of them have same period. And of course, if you say that, what if they have different periods? Then, it is not very meaningful, because see T 1 let us say occurred two times and T 3 one time only, then T 3 should succeed which one? That question comes. So, whenever there are precedence relationships, the tasks occur together the same period and that is equal to the LCM that we have assumed. And, we just computed for one such major cycle.

(Refer Slide Time: 42:23)



Extending EDF and RMA-based Schedulers

- Precedence constraints can be handled with the following modifications to the basic algorithm:
  - Do not enable a task until all its predecessors complete their execution.
  - Check the tasks waiting to be enabled (on account of its predecessors completing their executions) after every task completes.

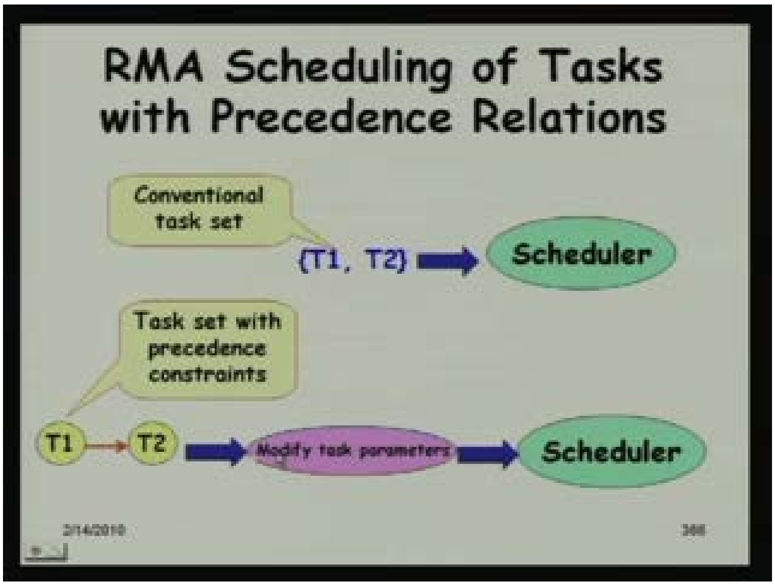
365

Now, let us see… We saw that, for cyclic schedulers, it is not very difficult to handle task dependences; simple algorithm. Just found out their worst case completion times. And, based on that, we said that what is the latest by which it should start; and then, for each of them, we did that and then move them forward; simple thing. But, now, let us see how do we extend the EDF and RMA to handle task dependencies. Precedence constraints can be handled with… Again, simple modifications. The modification is as follows.

We will take some example and illustrate. We will have to modify the task parameters actually. We should not enable a task. So far, we are saying that a task is enabled; can start executing any time after its phase. If it has a nonzero phase, we cannot start execution before its phase. So, after the phase, the task becomes ready; it is executed.

But, what you are saying here is that, even though the task has some phase, we cannot really make the task ready. We will keep the task pending; we will not make it ready; we will not enable it until all its predecessors complete execution. Simple thing, because see it is needing results from the predecessors; or, in other words, let me just say that, the task cannot start until its predecessors cannot complete. So, we will change its ready time, which may be more than phase. So, keep it pending; we do not bring it to the ready queue until its predecessors are complete. And then, also, we need to do another thing. The scheduler needs to check the tasks waiting after every tasks completes, because after a task completes, it might so happen that the task that is waiting for this to complete. The duration for which it is complete might have been over. So, this is the simple modification that we have to do.

(Refer Slide Time: 44:52)



RMA Scheduling of Tasks with Precedence Relations

Conventional task set

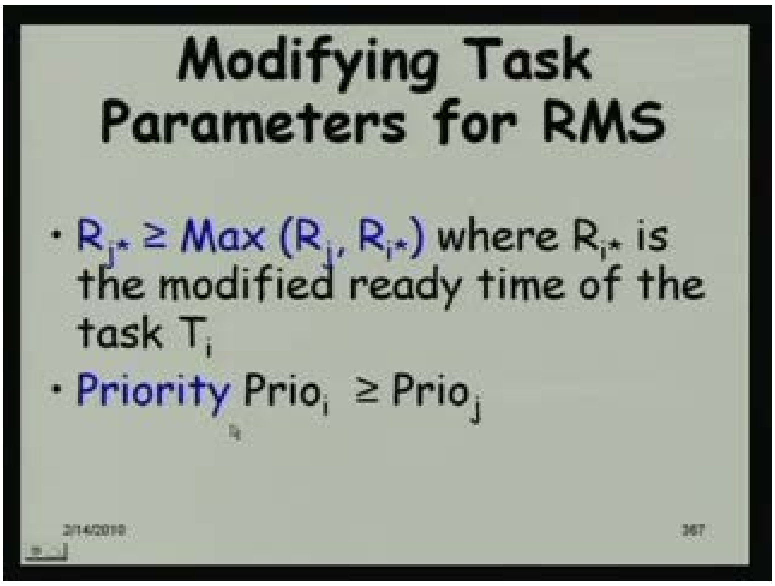
{T1, T2} → Scheduler

Task set with precedence constraints

T1 → T2 → Modify task parameters → Scheduler

Now, let us look at that with some examples. First, we will look at RMA. So far, we had independent tasks: T 1, T 2, etcetera. They are given to the RMA scheduler. But now, we will have precedence relations between tasks. We will modify the task parameters and then we will give it to the scheduler.

(Refer Slide Time: 45:18)



Modifying Task Parameters for RMS

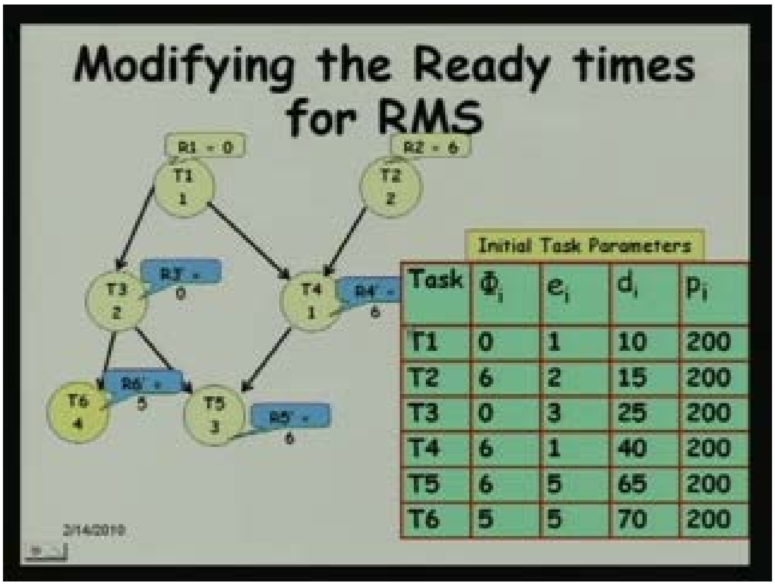
- $R_{j^*} \geq$ Max (R_j, R_{i^*}) where R_{i^*} is the modified ready time of the task T_i
- Priority $Prio_i \geq Prio_j$

2/14/2010

367

The modification that we will do for the rate monotonic algorithm or the rate monotonic scheduler is that the ready time of a task j… Ready time of a task j after modification – it is written star. Ready time of a task j should be greater than equal to the maximum of the ready time of this task, that is, the phase of this and that for all its predecessors. R i star is all its predecessors. And then, we will also change their priorities.

(Refer Slide Time: 46:02)



Modifying the Ready times for RMS

Initial Task Parameters

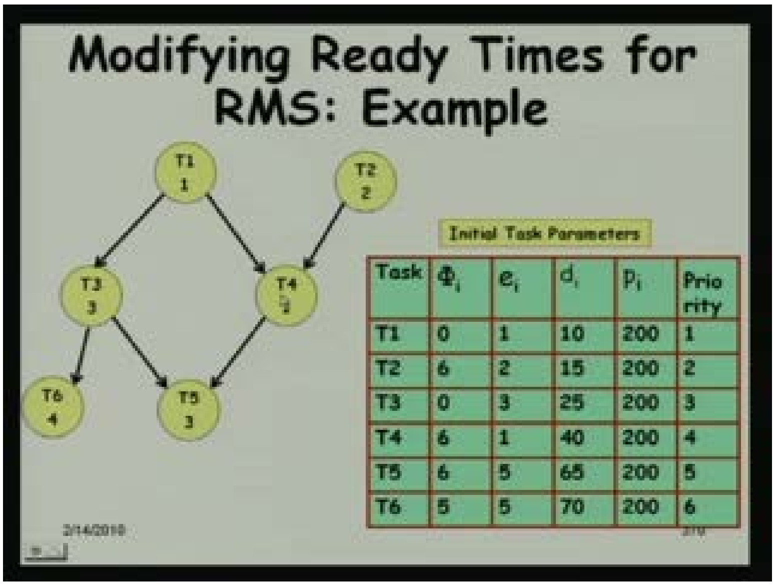
Task	Φ_i	e_i	d_i	p_i
T1	0	1	10	200
T2	6	2	15	200
T3	0	3	25	200
T4	6	1	40	200
T5	6	5	65	200
T6	5	5	70	200

2/14/2010

Let us see how we do that with the help of an example. Let us assume this kind of a task precedence. The T 1, T 2 take 1 and 2 milliseconds here to execute. And, T 2 has a phase of 6 milliseconds. T 3 requires 3 milliseconds to execute. T 4 takes 1 milliseconds to execute; and, T 6 takes 5 milliseconds to execute; and, T 5 takes 5 milliseconds to execute. And, T 6 has a phase of 5. Only these two tasks are nonzero phase. Now, their deadlines are also given. And, see all their periods are the same – 200. So, T 1 has a deadline 10; T 2 – 15; T 3 – 25; T 4 – 40; T 5 – 65 and T 6 – 70. So, these are the phases for these tasks we have written here. T 6 has 5. And, these are 0, 6; R 2 is 6.

Now, if we find the maximum of R 1 and R 3. So, that is 0. So, this is equal to 0 (Refer Slide Time: 47:36). Now, for R 4, it will be maximum of this, this and this – these three, which is equal to 6. So, we make it equal to 6. Now, what about T 6? It will be having a phase of 5. So, it will be maximum of 0 and 5. See we are modifying the ready times. We are doing for this one. So, this will be maximum of 6 and 0, which is equal to 6. And, for T 6, it will be maximum of 0 and 5, which is 5. So, we will modify their ready times; simple thing to do, so that the tasks precedence are honored. But, just modifying their ready times are not enough, because even if the task becomes ready slightly late, but what if T 1 priority is becoming less? Because they are all having equals priorities here. See they all have equal periods. So, we need to give different priorities to them, so that they enable the predecessor to complete before the successor can be taken execution. So, the predecessors would have higher priority.

(Refer Slide Time: 49:08)



Modifying Ready Times for RMS: Example

Initial Task Parameters

Task	Φ_i	e_i	d_i	p_i	Priority
T1	0	1	10	200	1
T2	6	2	15	200	2
T3	0	3	25	200	3
T4	6	1	40	200	4
T5	6	5	65	200	5
T6	5	5	70	200	6

2/14/2010

So, that is what we will do here. We will just keep on assigning T 1 is 1, 2, 3, 4, 5, 6.

We will take deadlines (( ))

No. If we take not really deadline, we are just looking at their precedence, because unless a predecessor does not complete, the successor if it starts, then it will block it if it is a higher priority.

For tasks, same level will (( ))

Just because they are all same period, their priority would have all been same. And, with same priority, this will not work; tasks can miss their deadlines. So, what we are trying to do is assign them priority, so that a predecessor will not get blocked on account of a successor; or, in other words, this will enable them to follow this sequence of execution. The sequence in which their… There in this task graph, we are just giving them the same priority. But, possibly what we could have done, see between T 3 and T 4, they are at the same level in the tree. So, we have the flexibility how to distinguish between these two (Refer Slide Time: 50:36) in the priority.

Can they be given the same priority?

Same priority – then, we will have to check if both of them will meet their deadline in the worst situation.

(( )) because its deadline is much…

Exactly. So, what we see is, whichever tasks deadline is lower, we give that higher priority. If we give them equal priority, then we have to check whether both of them will complete by that time.

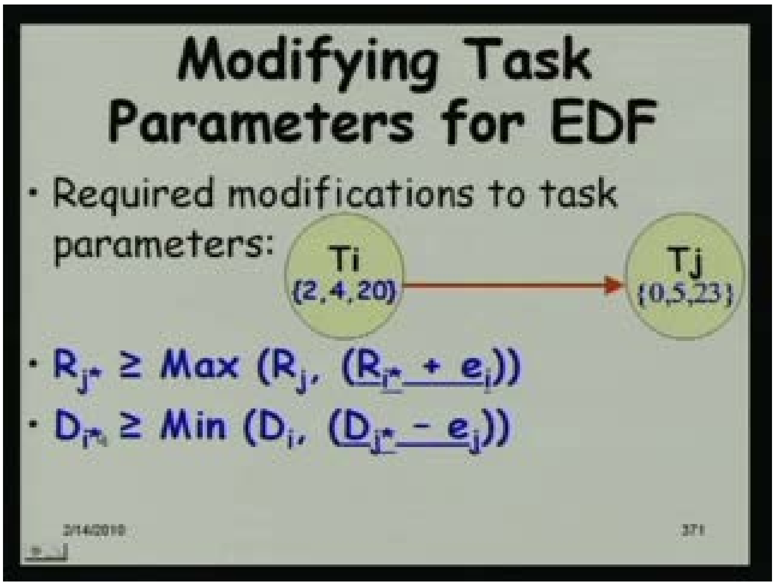
In case of… Since it is a modified RMS, we do not consider for that deadline as a (( ))

No, see the priorities are all in a rate monotonic algorithm given based on the frequency or the period. But, what we are saying is that, here they should have all got the same priority, but we are trying to give them in different levels of priority, because we want the predecessor to complete before the successor can start, because what if the successor starts off and the predecessor has not completed and some predecessor? Then, that will get blocked. And then, this chain will be violated. Simple thing; just think about it.

But since they are (( )) has the same level in the precedence graph, they can be given the same (( ))

No, that argument is not valid. See what he is saying is that, whether tasks in the same precedence level can be given the same priority? No, that argument is not valid, because we have to also consider the deadline. What if T 4 keeps on executing and T 3 cannot complete before T 4 completes? T 3 might miss its deadline. So, what we see is that, between T 3 and T 4, who has the nearest deadline, and give it the higher priority.

(Refer Slide Time: 52:31)



Modifying Task Parameters for EDF

- Required modifications to task parameters:
- $R_{j*} \geq \text{Max}\ (R_j,\ (\underline{R_{i*}\ +\ e_i}))$
- $D_{i*} \geq \text{Min}\ (D_i,\ (\underline{D_{j*}\ -\ e_j}))$

2/14/2010 371

Ti
{2,4,20}
Tj
{0,5,23}

Now, let us see how to modify EDF. Similar thing we will have to do here; only slightly complicated; slightly more complicated here. Here for the ready time of a task, we look at the ready time of that same task and also all its predecessors ready time plus the execution time. And, we will also have to change the deadlines of the tasks, because the scheduling is done based on the deadlines. And, this is the way we have to compute the deadline. And, we will just take some example in the next class and see how a task set with precedence relations can be scheduled under EDF successfully by modifying their ready time and their deadlines.

We will stop here and meet next class to carry on from this point. Thank you.