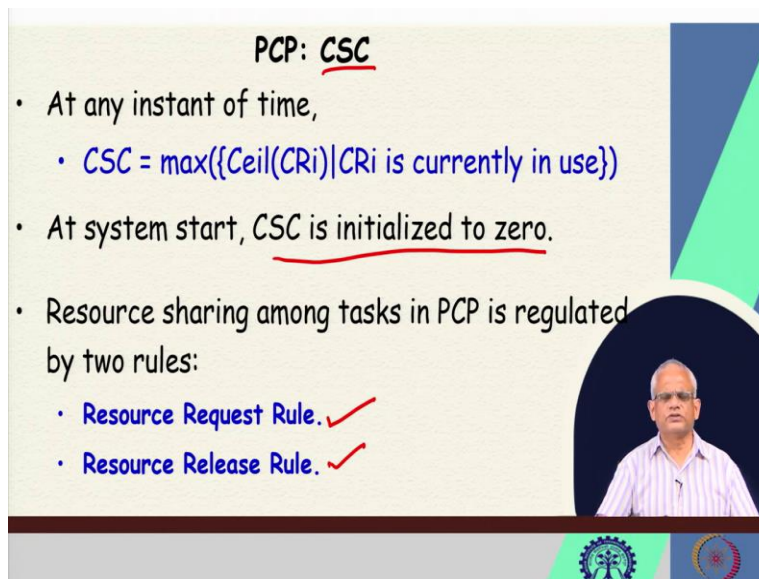


Real Time System
Professor Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Lecture 30
Working of Priority Ceiling Protocol

Welcome to this lecture, we are discussing in the last lecture about the priority ceiling protocol. The priority ceiling protocol is a sophisticated protocol used in large sophisticated applications. It is bit more complex than the basic priority inheritance protocol and the highest locker protocol, in the last class trying to understand the priority ceiling protocol.

In this lecture, we will first discuss the nitty-gritty of the priority ceiling protocol and then we will find the maximum blocking times that the tasks may suffer, maximum inversion times that the tasks can suffer under the ceiling protocol and then we will compute the schedulability of a task, the impact of the schedulability when the ceiling protocol is used and based on this we can also easily compute the impact of the schedulability of the basic priority inheritance scheme and the highest locker scheme. Now, let us proceed.

(Refer Slide Time: 01:43)



PCP: CSC

- At any instant of time,
 - $CSC = \max(\{\text{Ceil}(CR_i) | CR_i \text{ is currently in use}\})$
- At system start, CSC is initialized to zero.
- Resource sharing among tasks in PCP is regulated by two rules:
 - Resource Request Rule. ✓
 - Resource Release Rule. ✓

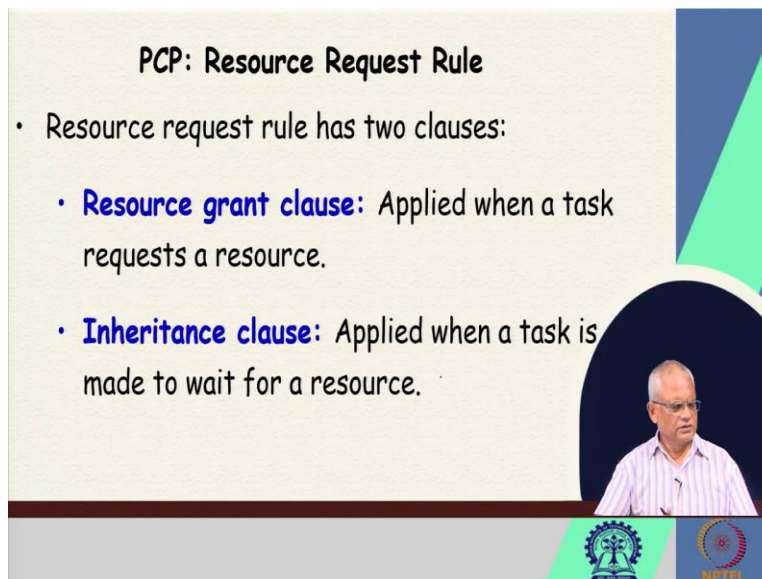
Just to recollect in the priority ceiling protocol, we need a system environment variable called as the current system ceiling CSC and anytime during the execution, the current system ceiling has the value of the ceiling of all the resources that are currently in use. If some new resources

acquired by a task, then the ceiling value, the CSC is examined and it is updated if the new resource which has acquired has a higher ceiling value.

And similarly, if a task releases a resource, then its impact on the CSC is examined if the ceiling value of the task is equal to the system ceiling, then the system ceiling value is adjusted to be the highest ceiling of all remaining resources that are being used. And at system start, the system ceiling is set to zero assuming that there is no such priority as zero priority.

So, it would be set to zero to indicate that it is a system initialized and there is no resources that are being used and the ceiling value is used by the protocol, when tasks, request for resource and tasks, release resources. So, both of these times the ceiling protocol has a set of rules. When a task requests for a resource, the resource request rule is applied and when a tasks releases a rule, the resource release rule is applied.

(Refer Slide Time: 04:00)



PCP: Resource Request Rule

- Resource request rule has two clauses:
 - **Resource grant clause:** Applied when a task requests a resource.
 - **Inheritance clause:** Applied when a task is made to wait for a resource.

Now, let us look at the resource request rule. That is the rule that is applied when a task requests for protocol for resource under this protocol. The resource request rule has 2 clauses one is the resource grant clause whenever a task a request for a resource, the resource grant clause is applied and when the grant clause is applied, it may be determined based on this grant clause whether the resource can be granted or not granted.

So, if the resource is granted, then only the ceiling value will be changed. So, your current system ceiling and so on. On the other hand, if the resource is denied, to a task requesting for a resource, then an inheritance clause is applied to the presently executing task.

(Refer Slide Time: 05:14)

PCP: Resource Grant Clause

- Unless a task holds a resource that set the CSC (current system ceiling):
 - It is not allowed to lock a resource unless its priority is greater than CSC.

The diagram shows a resource R1 with a ceiling of 2 (CSC = ceil(R1) = 2). Task T5 is using R1. Task T3 is requesting R3, but its priority (3) is less than the current system ceiling (2), so the request is denied. Task T1 is requesting R2, and its priority (1) is greater than the current system ceiling (2), so the request is granted. The diagram also shows task T3's priority (3) and task T1's priority (1) relative to the CSC (2).

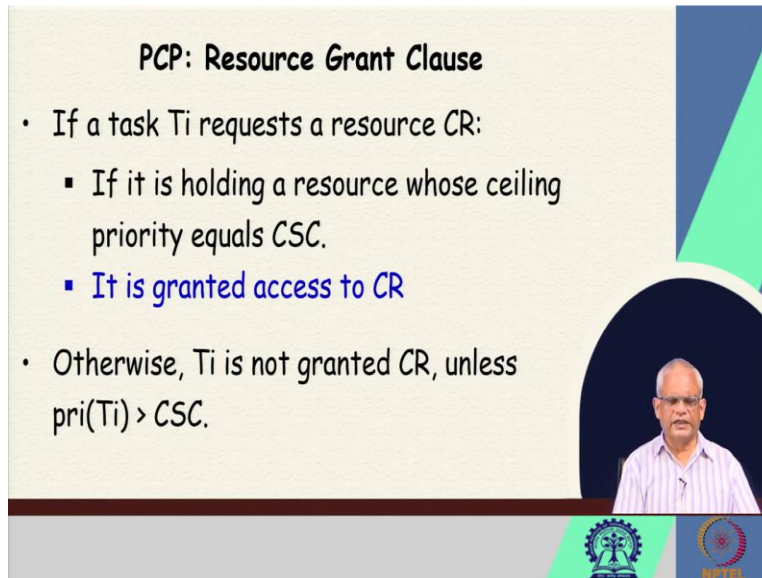
Now, let us look at these 2 clauses of the resource request rule with some examples, so that we can better understand this. Under the resource request rule, the first clause is the resource grant clause. So, here a resource requested by a task is granted to the task, if that task had set the current system ceiling. So, that means, if there is a task that is using a resource.

So, let me just say that this is the resource and there is a task which is using the resource. Let us, say some T_5 is using resource R_1 . Then the current system ceiling if it is set by T_5 , so which is current system ceiling, if it is equal to the ceiling of R_1 . Then if T_5 requests resource R_2 under the resource grant clause, it is granted the resource. But let us say if a task T_3 needs another resource, let us say R_3 . If T_3 requests R_3 then this will be denied.

Because T_3 's priority is less than the current system ceiling, R_1 might be used by let us say T_1 and therefore ceiling of R_1 is equal to let us say 1 and T_3 's priority, let us say 3 and 3 will be compared with 1 and since $3 < 1$, T_3 will be denied using R_3 even though R_3 is free. And of course, let us say the ceiling is 2 here and if a task 1 with priority 1 requests some resource that will be granted that resource.

So, that is what it says, it can lock a resource if its priority is greater than CSC. So, when a request for a resource comes, if it is a task, which is using the highest ceiling value resource with highest ceiling value, then it will be granted the resource. Otherwise, if its priority is greater than the CSC value, then it will be granted the resource otherwise it will just block. It will be denied the resource.

(Refer Slide Time: 08:10)



PCP: Resource Grant Clause

- If a task T_i requests a resource CR :
 - If it is holding a resource whose ceiling priority equals CSC .
 - It is granted access to CR
- Otherwise, T_i is not granted CR , unless $pri(T_i) > CSC$.

The slide features a video inset of a man with glasses and a mustache, wearing a light-colored striped shirt, speaking. The background of the slide is light beige with a blue and green geometric design on the right side. At the bottom, there are logos for a university and NPTEL.

But if it is denied the resource so if it is not granted, it is not granted the resource and under the situation if it is not the task, which is holding the maximal ceiling resources maximum ceiling value or its priority is greater than current system ceiling. Otherwise it will be denied the resource.

(Refer Slide Time: 08:39)

PCP: Resource Grant Clause

- If T_i is granted access to the resource CR_j ,
- Then, if $CSC < \text{Ceil}(CR_j)$, then CSC is set to $\text{Ceil}(CR_j)$.

The diagram illustrates the Resource Grant Clause. It shows a box labeled R_1 with $C=5$ above it and T_7 below it with an arrow pointing up. To the right is a box labeled R_2 with $C=3$ above it. To the left, $CSC = 5$ is written and crossed out with a red line, and 3 is written below it. A red arrow points from the 3 to the R_2 box. The diagram is overlaid on a video feed of a man in a striped shirt.

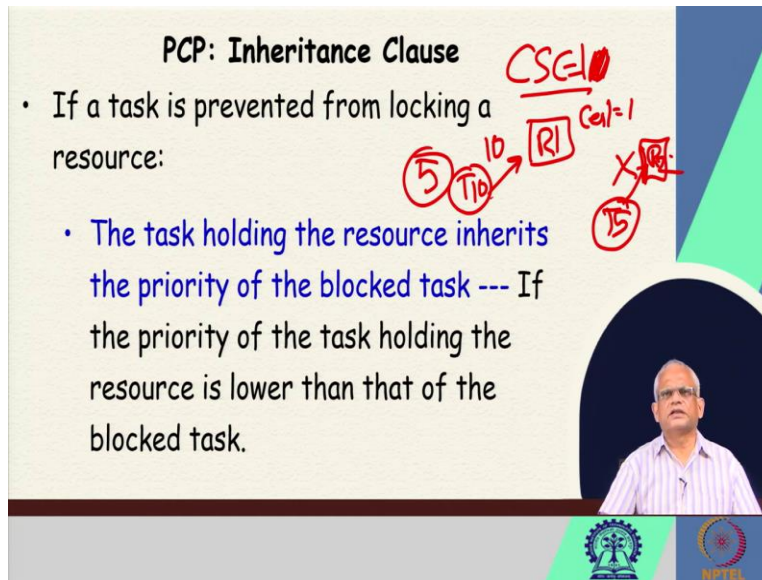
If it is denied the resource and if it is granted the resource let us say, if it is granted, a task is granted a resource R_1 the ceiling value is let us say 5 and it was being used by a task T_7 and T_7 requested another resource R_2 whose ceiling value is 3. So, when T_7 was executing using R_1 the current system ceiling is equal to 5.

Now, when T_7 requested R_2 it will be granted because this is the task which is set to the CSC and now it has acquired R_2 whose ceiling value is 3. So, immediately, thus, current system ceiling will be made equal to 3. So, that is the rule that is what we are saying here. When T_i is granted access to the resource, if the current system ceiling is less than the new resource which has been granted, then CSC is made equal to the ceiling of CR_j .

(Refer Slide Time: 10:00)

PCP: Inheritance Clause

- If a task is prevented from locking a resource:
- The task holding the resource inherits the priority of the blocked task --- If the priority of the task holding the resource is lower than that of the blocked task.

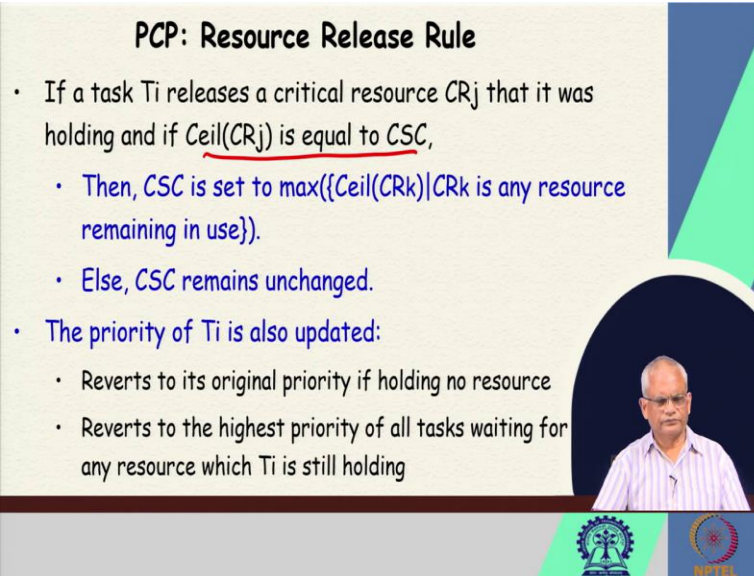


If a task requesting a resource is denied the resource, then the inheritance clause comes into picture. If the task is granted the resource we do not apply the inheritance clause. The task is prevented from locking a resource then the inheritance clause is applied here the task holding the resource inherits the priority of the blocked task if the priority of the task holding the resource is lower than the priority of the blocked task.

So, here, let us say the current system ceiling is 10 is 1 the current system ceiling is let us say 1 because a resource R1 whose ceiling is 1 is being held by a task T₁₀ whose priority is 10. So, when T₁₀ acquired R1 the current system ceiling will be set to 1 because the ceiling priority of R1 is 1. Now, let us say there is another task T₅ would start executing T₅ can execute because T₁₀'s priority has not changed, it still executes it priority 10 and 5 is a higher priority task and it starts executing and after some time it requests R5 or let us say R2.

Now, T₅'s priority is checked against the current system ceiling. So, 5 is less than 1. So, it is not a higher priority task than the current system ceiling and also it is not the task which is set the ceiling value the current system ceiling. So, T₅ will be denied access to R2, even though R2 was free. But as soon as it is denied, the inheritance clause will be applied and T₁₀ will inherit the priority T₅. So, T₁₀'s priority will become 5. So, that is the inheritance clause here.

(Refer Slide Time: 12:32)



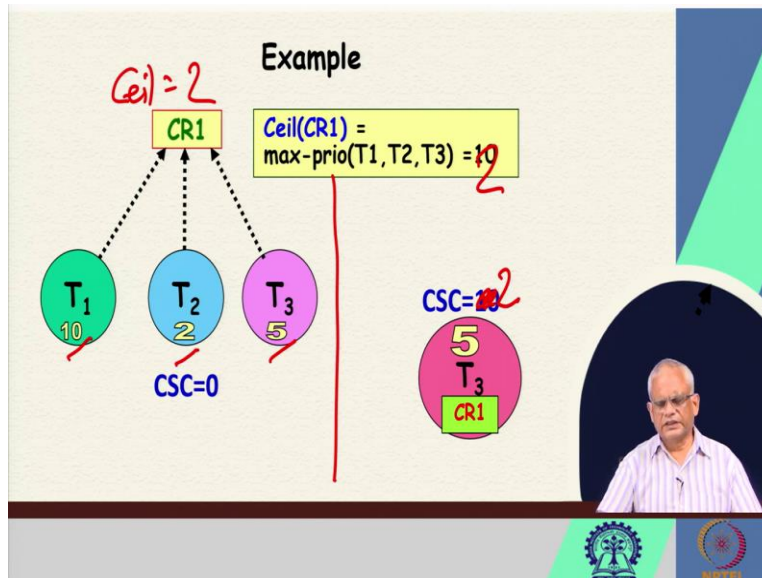
PCP: Resource Release Rule

- If a task T_i releases a critical resource CR_j that it was holding and if $Ceil(CR_j)$ is equal to CSC ,
 - Then, CSC is set to $\max(\{Ceil(CR_k) | CR_k \text{ is any resource remaining in use}\})$.
 - Else, CSC remains unchanged.
- The priority of T_i is also updated:
 - Reverts to its original priority if holding no resource
 - Reverts to the highest priority of all tasks waiting for any resource which T_i is still holding

We will look at another example to illustrate this. And when a task releases a resource the resource release rule is applied. Here the resource which was released by a task if that is the ceiling priority of that is equal to the current system ceiling then the current system ceiling is set to the maximum of the remaining resources under use very intuitive.

Otherwise, if the resource that was released is not the ceiling of that resource is not equal to CSC , CSC is higher than the ceiling of the resource being released then CSC remains unchanged and priority of the task is also updated. It might have inherited the priority of some task waiting for that resource. So, in that case, it would revert to its original priority. But if it is still holding other resources for which there are also tasks waiting, then it would have the highest priority of the all the tasks waiting for the resources that are being held by T_i .

(Refer Slide Time: 14:00)

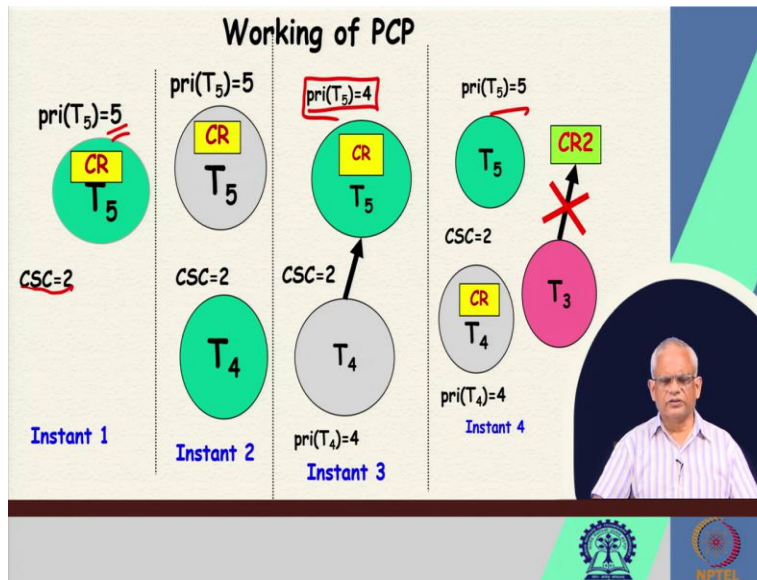


Now, let us take one example. Let us, say from a static analysis of the code it determine that the resource CR1 will be used some time or other by 3 tasks T₁, T₂, T₃, T₁'s priority is 10, T₂'s priority is 2 and T₃'s priority is 5 and the smaller the priority value higher the priority. So, T₂ is the highest priority task. Now, what will be the ceiling priority of the resource CR1? The ceiling priority of the resource CR1 is the maximum of the priority of all these tasks.

So, the maximum priority of T₁, T₂, T₃, which is equal to 2. So, the CR1, the ceiling will be become 2 and to start with the current system ceiling will be set to 0. Now, let us say there is a task, the task T₃ is waiting and its priority is 5. The T₃ is executing, and its priority is 5 and at this time, it acquired the resource CR1. So, what will be the implication of this?

It requested CR1 and since the current system ceiling is 0, indicating that no task is using any resource, so T₃ will be granted access to CR1, its own priority will remain unchanged. But then the current system ceiling = 2 and there will be no other change.

(Refer Slide Time: 16:33)



Now, let us explain the working of the ceiling protocol with slightly different in more complex situation. Let us, say we have a task T₅ using a critical resource 5 a critical resource and the priority of T₅ is 5 that is at the instance in the CSC is 2 and at instant 2 T₄ starts executing T₅ even though it has used the acquired the critical resource, its own priority has not changed. It is executing at its own priority 5, it is only the system ceiling current system ceiling is equal to the ceiling priority of CR and ceiling priority of CR is 2 therefore CSC is 2.

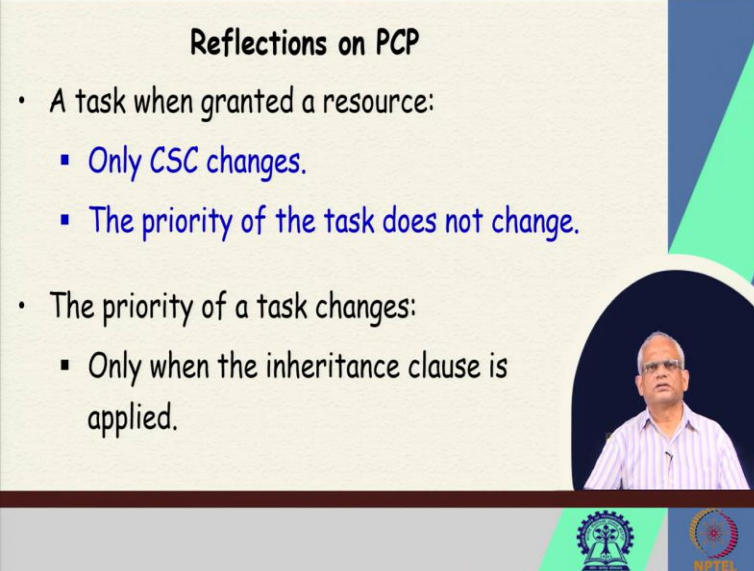
And now T₄ is the higher priority task than T₅ it starts executing and after some time, it needs the resource CR and as soon as it needs the resource CR the T₅'s priority becomes 4 it has inherited the priority T₄, T₄ was denied access to the resource and the priority of T₅ increases to 4 and it inherits the priority of 4.

Now, let us say after some time T₅ releases the resource so it gets back its own priority 5 and T₄ gets the resource CR the current system ceiling remains unchanged because that is the ceiling priority of CR and the priority of T₄, it just executes at its own priority 4. But let us say a higher priority tasks T₃ starts executing because that is higher than T₄ it can preempt T₄ before T₃ starts executing. And let us say after some time it requests for CR2, that is a different resource.

It is holding CR now it is requesting for CR2 but it will be denied access to CR2 because T₃'s priority is less than the current system ceiling so that clause will be applied. There is resource

grant clause and T_3 will be denied access to CR2 even though CR2 is free. So, that is the working of the priority ceiling protocol.

(Refer Slide Time: 19:35)



Reflections on PCP

- A task when granted a resource:
 - Only CSC changes.
 - The priority of the task does not change.
- The priority of a task changes:
 - Only when the inheritance clause is applied.

The slide features a video inset of a speaker in the bottom right corner. The background is light beige with a blue and green geometric design on the right side. Logos for IIT Bombay and NPTEL are visible at the bottom.

When a task is granted a resource only the CSC changes, the priority of the task does not change. This is different from the highest locker protocol. In the highest locker protocol as soon as a task is granted a resource, the tasks priority becomes equal to the ceiling priority of the resource. Here does nothing happens, the task continues to execute its own priority and only some system variable CSC's value changes.

And if you can remember that in the highest locker protocol, the inheritance related inversions were occurring, because the task was getting very high priority value as soon as it acquired a resource, but here its priority is not changing is executing at its own priority and therefore, such inheritance inversions will not occur here. The priority of the task does not change, the priority of a task changes only when there is another higher priority task waiting for this resource and then the inheritance clause is applied.

(Refer Slide Time: 21:05)

PCP: An Analysis

- Prevents deadlocks. ✓
- Prevents chain blocking. ✓
- Prevents unbounded priority inversion.
- Limits inheritance-related inversion. ✓

The slide features a video inset of a man in a striped shirt speaking. The background is light beige with a blue and green geometric design on the right. Logos for IITM and IITR are visible at the bottom.

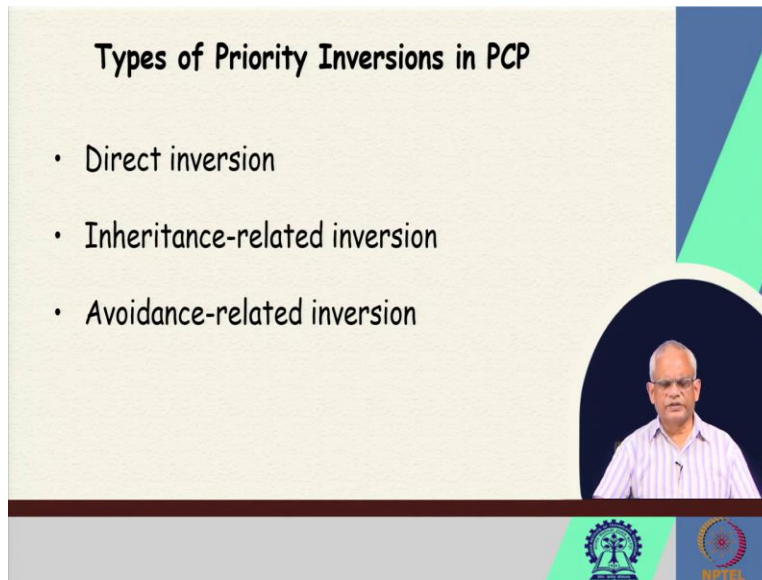
Now, let us analyse the priority ceiling protocol. We will see that it prevents deadlock, it prevents chain blocking, it prevents unbounded priority inversion and it limits the inheritance related inversion. Inheritance related inversion does not become 0 because here also there is inheritance clause and the tasks priority may become higher when another task with a resource requirement is waiting, which has been denied access to a resource.

So, as you can see, this is the superior protocol compared to the basic inheritance scheme priority inheritance scheme and also the highest locker protocol it prevents deadlock prevents chain blocking prevents unbounded priority inversion and the inheritance related inversion is brought down to very low values compared to the highest locker protocol when inheritance related inversion is very high.

(Refer Slide Time: 22:19)

Types of Priority Inversions in PCP

- Direct inversion
- Inheritance-related inversion
- Avoidance-related inversion

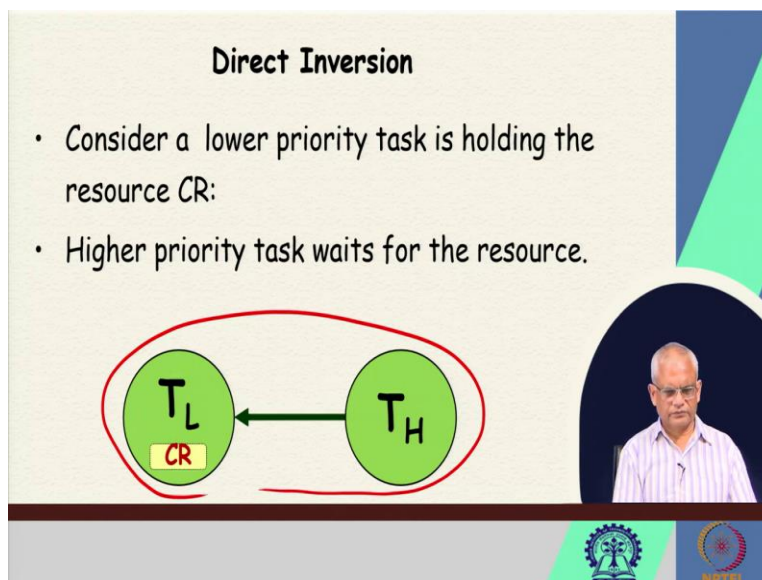


Now, let us understand the inversions that can occur in PCP and we will use this in our analysis schedulability analysis that will do the total blocking time or priority inversion time for a task and that we will use for computing its schedulability. Now, the types of inversions that can occur in the ceiling protocol, one is the direct inversion inheritance related inversion and avoidance related inversion. So, there are 3 types of inversion that can occur in the ceiling protocol.

(Refer Slide Time: 23:05)

Direct Inversion

- Consider a lower priority task is holding the resource CR:
- Higher priority task waits for the resource.



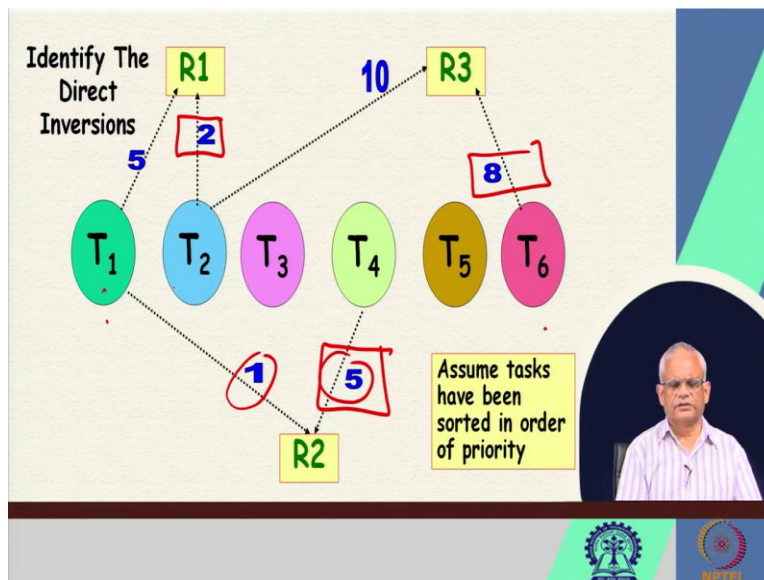
First, let us look at the direct inversion. This is very simple. A low priority task is holding resource and high priority task requests for that resource it has to wait. So, this is the situation for

direct inversion. A high priority task is blocking for a resource which is being held by a low priority task. So, in highest locker protocol, this situation does not occur, direct inversion does not occur, because as soon as TL acquires a resource its priority becomes very high.

And then the high priority task cannot execute, cannot preempt TL because its priority has become equal to its own priority cannot preempt that and therefore direct inversion does not occur in the highest locker protocol. But here it can occur because TL even after occurring CR, its priority does not change. So, TH can preempt TL start executing and after some time it needs CR and TH's priority will be compared to the CSC.

And since CSC will be at least equal to that TH. It will be denied access to the resource CR and therefore it will block and priority of TL will be increased by the resource inheritance clause. But we say that whenever the situation occurs, that a low priority task is holding a resource then the high priority task is waiting for that resource, we call this as the direct inversion.

(Refer Slide Time: 25:12)



Now, let us look at this task resource graph which you have obtained from a static analysis of the code there are 6 tasks T₁ is the highest priority task and T₆ is the lowest priority task and we have named the tasks such that they are in the descending order of their priority T₁ is the highest priority and T₆ is the lowest priority and their resource requirement is shown by shaded line a dotted line and the time for which the need the resource is shown on the dotted line.

Now, R2 is needed by T₁ and T₄, R1 is needed by T₁ T₂ and R3 is needed by T₂ T₆. Now, here, which tasks can undergo direct inversion and for how long? So, T₁ can undergo direct inversion by T₄ on account of R2. So, T₄ might be holding R2 and then T₁ starts executing needs R2 and it gets blocked. So, what is the duration for which T₁ can undergo direct inversion that is 5 because T₄ can hold it for 5 units and therefore, T₁ can undergo direct inversion for 5 units T₁ can also undergo direct inversion by T₂ on account of R1 for 2 units.

But, can T₂ undergo inversion due to R1? No, T₂ does not go even though T₁ is holding R1, T₂ does not undergo inversion because T₂ cannot execute when T₁ is executing and also a task undergoes inversion only by the lower priority tasks. If a higher priority task is executing, it is not called inversion.

Now, what about T₂ and T₆? T₂ undergoes direct inversion due to T₆ for duration of 8 units because T₆ might use R3 for 8 units. So, T₂ can undergo inversion on account of T₆ for 8 units. So, there are 3 tasks which can under 2 tasks T₁ can undergo inversion for 2 units and 5 units T₂ can undergo for 8 units so two tasks T₁ and T₂ can undergo inversion.

(Refer Slide Time: 28:20)

Inheritance-Related Inversion

- When a low priority task is holding a resource and a high priority task is waiting for resource:
 - The priority of the low priority task is raised.
 - An intermediate priority task not needing that resource:
 - Suffers inheritance-related inversion.

The slide includes a diagram with three circles representing tasks: T_L (Low priority), T_H (High priority), and T_M (Intermediate priority). T_L is circled in red and has a checkmark, indicating it is the task holding the resource. T_H is also circled in red, indicating it is the high priority task waiting for the resource. T_M is circled in blue, indicating it is the intermediate priority task that suffers inheritance-related inversion. A red arrow points from T_M towards T_L, suggesting the priority inheritance.

Now, the other type of inversion is called as inheritance related inversion here if a task is holding a resource a low priority task is holding a resource and the high priority task is waiting then the low priority task will get the priority of the high priority task and let us say some intermediate

priority tasks they will they cannot execute because the priority of TL is become equal to that of the high priority task.

So, the inheritance clause by that the priority of TL is raised to that of TH and therefore the task T_i which is higher than L but lower than H this will undergo inheritance related inversion. So, again we will do a bit of analysis for inheritance related inversion, like given a task graph, what is the inheritance related inversion that a task may suffer. We are at the end of this lecture. We will stop here and we will continue in the next lecture. Thank you.