Real Time Operating System

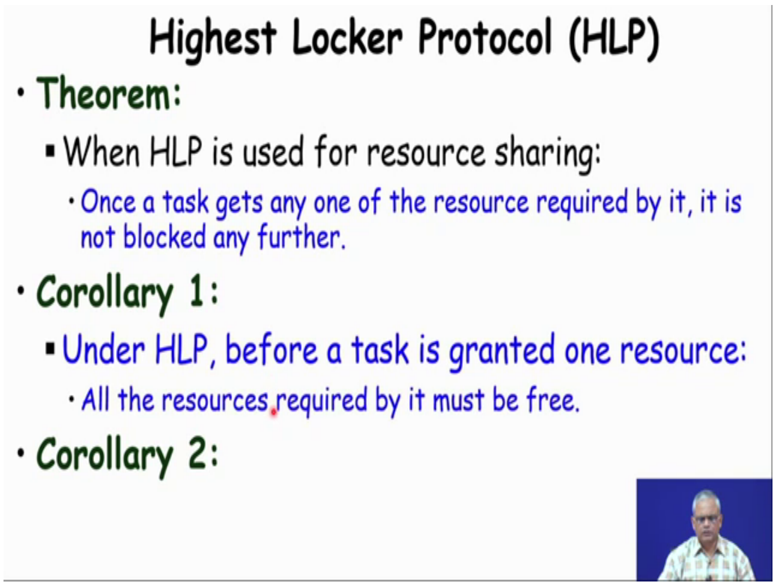
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 16
Priority Ceiling Protocol

Welcome back to this lecture. In the last lecture we were discussing about how real time task can share resources; initially we had looked at the priority inheritance protocol which is very simple scheme, where a lower priority task which has acquired a resource inherits the priority of a higher priority task waiting for the resource, but then the basic priority inheritance scheme had two major problems. One is that it does nothing to avoid deadlocks and the second is the chain blocking problem, where a higher priority task needing multiple resources undergoes inversions for each of the resources. We looked at improvement over the basic priority inheritance scheme known as the highest locker protocol.

n the highest locker protocol ceiling priorities are assigned to resources depending on what is the highest priority task that may use that resources. And once a task acquires the resource its priority is increased to be equal to the ceiling value, and therefore, other tasks which do not need the resource, but are of higher priority they get blocked, and that is called as inheritance blocking. And this is a major problem with a highest locker protocol, and in today's lecture will quickly complete our discussion and the highest locker protocol and will discuss the priority ceiling protocol.

(Refer Slide Time: 02:14)



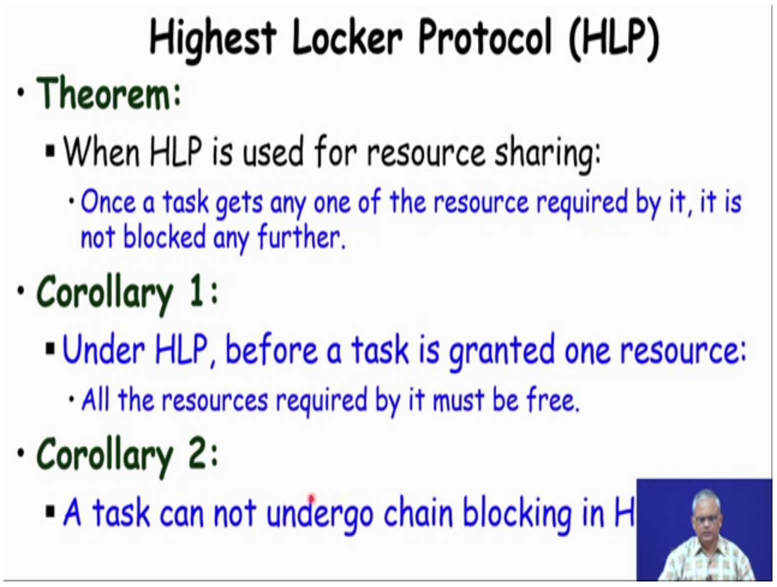
Highest Locker Protocol (HLP)

- Theorem:
  - When HLP is used for resource sharing:
    - Once a task gets any one of the resource required by it, it is not blocked any further.
- Corollary 1:
  - Under HLP, before a task is granted one resource:
    - All the resources required by it must be free.
- Corollary 2:

in the highest locker protocol let us look at some of the properties of this protocol, when a task gets one of its resource it is not blocked any further. What it means is that under the highest locker protocol chain blocking cannot occur why is that how can we argue? A formal proof is given in the text book, but then in formally we can understand that when a task acquires a resource its priority increases. To be equal to the highest priority that may use the resource and therefore, if any of the resource that is used by a task is acquired already acquired by another task, then that task would have higher priority and the current task could not be allowed to run. Let me repeat again that let us a task needs two resources r 1 and r 2 the pr ceiling value associated with those two resources must be equal to or greater than the task that we are considering. And let us assume that one of the tasks is locked by a one of resources is locked by another task, or both the resources are locked by a other task and then.

Let us assume that this task requires the first resource, and then it blocks until the first resource is freed, but then the task holding the second resource if priority does not drop below the task waiting for the resource, because the resource ceiling value must be greater than the task that is requesting the resource. Therefore, it is easy to see that under highest locker protocol chain blocking cannot occur and therefore, before a task is granted one resource all resources that it requests must be free.

(Refer Slide Time: 04:54)

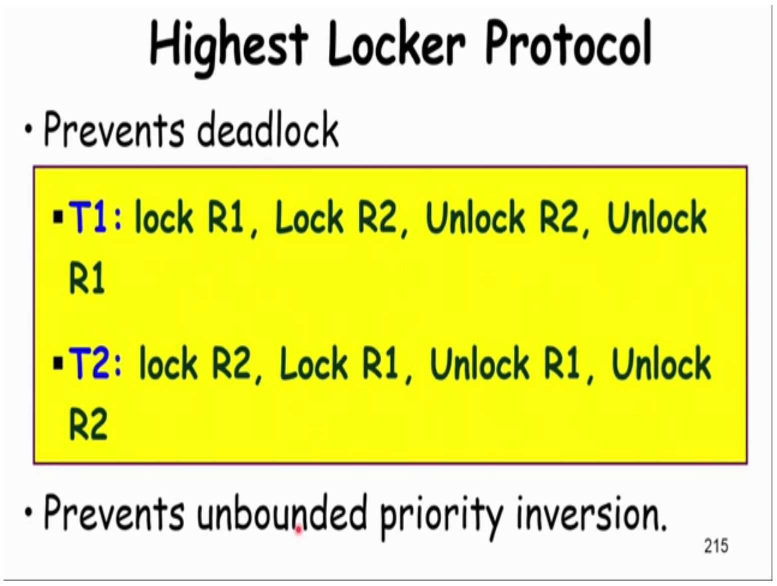


Highest Locker Protocol (HLP)

- Theorem:
  - When HLP is used for resource sharing:
    - Once a task gets any one of the resource required by it, it is not blocked any further.
- Corollary 1:
  - Under HLP, before a task is granted one resource:
    - All the resources required by it must be free.
- Corollary 2:
  - A task can not undergo chain blocking in H

And it cannot undergo chain blocking, in the highest locker protocol.

(Refer Slide Time: 05:02).



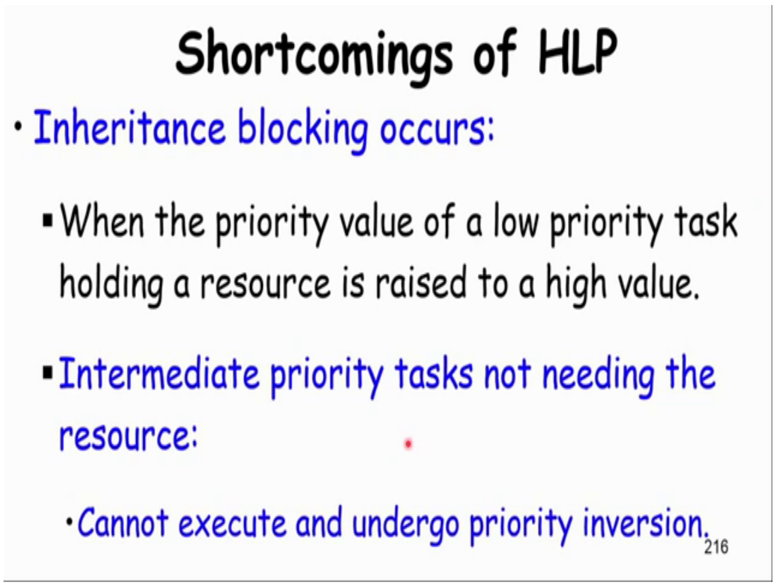
Highest Locker Protocol

- Prevents deadlock
  - T1: lock R1, Lock R2, Unlock R2, Unlock R1
  - T2: lock R2, Lock R1, Unlock R1, Unlock R2
- Prevents unbounded priority inversion.

215

Now let us look a its behavior with respect to deadlock. Let us assume that the task T 1 and T 2 are the similar sequence of events as occurred in the case of simple priority inheritance scheme where we showed that deadlock occurs. Here let us assume a similar set of instructions let us assume T 2 is the lower priority it locks R 2, but once it locks R two the priority of T 2 gets increased at least to equal to T 1 and therefore, T 1 cannot become ready cannot get CPU to lock R 1 and therefore, T 2 priorities increased atleast to that of T 1 and it locks R 2, R 1 then unlocks R 1, R 2 etcetera. So, deadlock cannot occur you can take any other example, and see that deadlock cannot occur because T 2s priority is made equal to the ceiling priority of the resource it locks and if the other tasks needs the resource its priority will be as much as the priority of the other task and also it prevents unbounded priority inversion because the tasks priority as soon as it acquires the resource increased to high value, which is the ceiling of the resource.

(Refer Slide Time: 06:47)



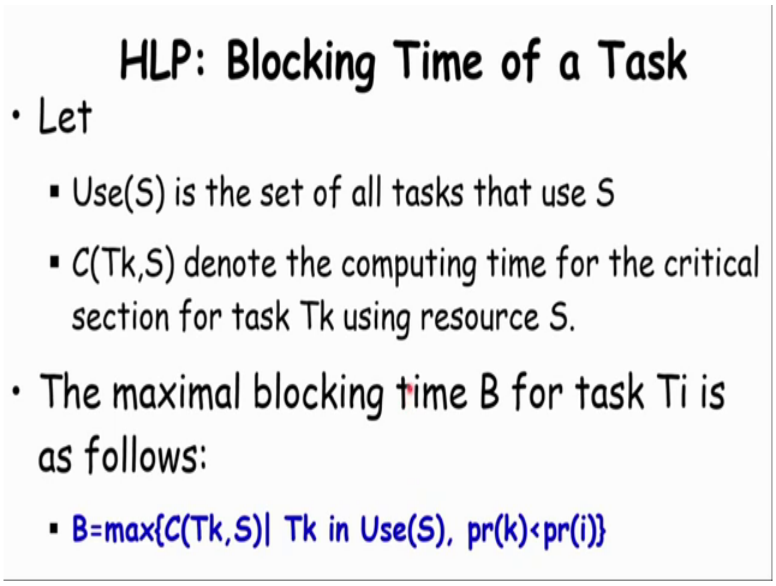
Shortcomings of HLP

- Inheritance blocking occurs:
  - When the priority value of a low priority task holding a resource is raised to a high value.
  - Intermediate priority tasks not needing the resource:
    - Cannot execute and undergo priority inversion.

216

But as we have already discussed that the major shortcoming of the highest locker protocol is the inheritance blocking. Since the priority value of a low priority task as soon as it acquires a resource is raised to a high value, the other intermediate priority tasks which do not need the resource undergo inheritance blocking, and the intermediate priority tasks can miss their deadline; because for multiple lower priority tasks they may face inversions priority inversions and they may miss their deadline. let us look at the blocking time for a task.

(Refer Slide Time: 07:36)

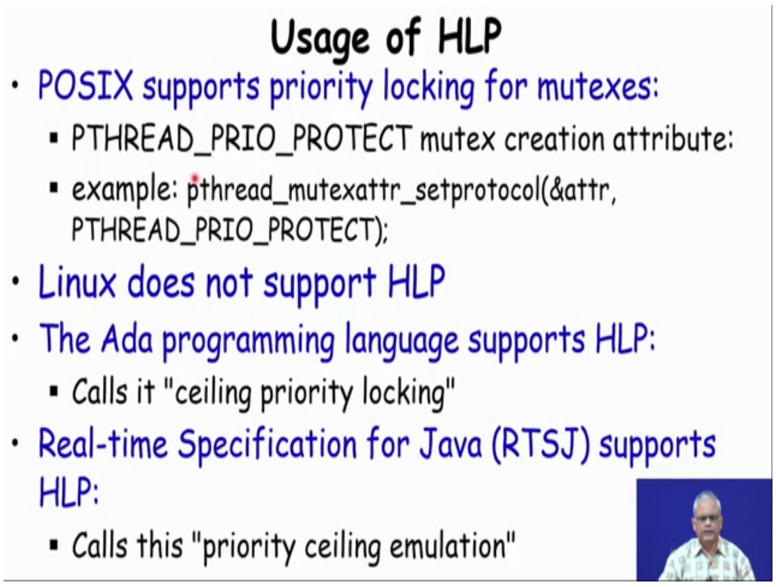


HLP: Blocking Time of a Task

- Let
  - Use(S) is the set of all tasks that use S
  - C(Tk,S) denote the computing time for the critical section for task Tk using resource S.
- The maximal blocking time B for task Ti is as follows:
  - B=max{C(Tk,S)| Tk in Use(S), pr(k)<pr(i)}

Let us assume that uses s set is the set of all tasks that use the non preamptable resource S, and T k S C T k s is the computation time using the resource S for the task T k therefore, any task T i can be blocked to the maximum time which is the maximum of T k using the S and T k is in uses S. What it really means is that, for if we are trying to compute what is the maximum blocking time for a task T i which does not need the resource. We find all the lower priority tasks. So, that is pk priority of k. So, that is all lower priority tasks and we find that what are the resources it uses and then find what is the maximum computation time using the resource. For all the lower priority tasks what is the resources, they use and what is the maximum time they use and that is the blocking time per the task I.

(Refer Slide Time: 09:18)

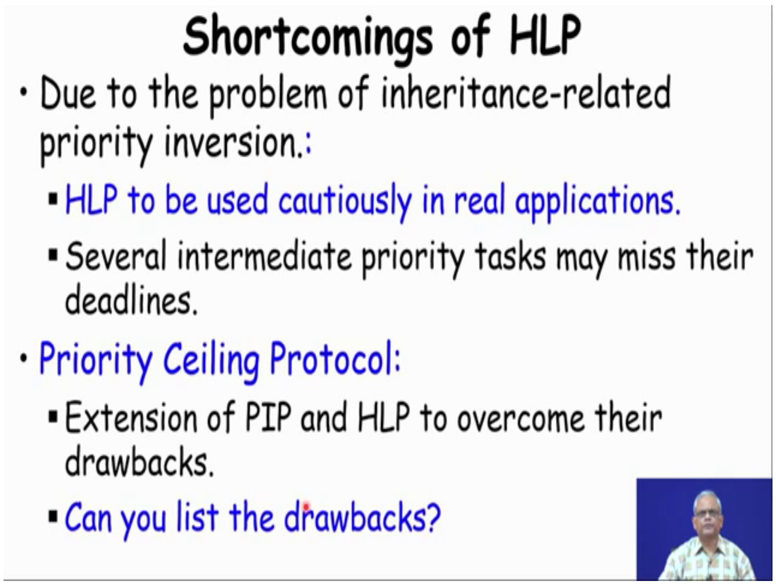


Usage of HLP

- POSIX supports priority locking for mutexes:
  - PTHREAD_PRIO_PROTECT mutex creation attribute:
  - example: pthread_mutexattr_setprotocol(&attr, PTHREAD_PRIO_PROTECT);
- Linux does not support HLP
- The Ada programming language supports HLP:
  - Calls it "ceiling priority locking"
- Real-time Specification for Java (RTSJ) supports HLP:
  - Calls this "priority ceiling emulation"

The highest locker protocol is used in many operating systems for example, the POSIX standard there we have the priority locking mutex, where we have this set priority to p thread and PRIO protect; the PRIO protect basically is the highest locker protocol. So, its supported by POSIX, Linux does not support highest locker protocol, but other programming languages supports highest locker protocol, even the real time specification for java supports highest locker protocol though it calls it a the priority ceiling emulation.

(Refer Slide Time: 09:55)

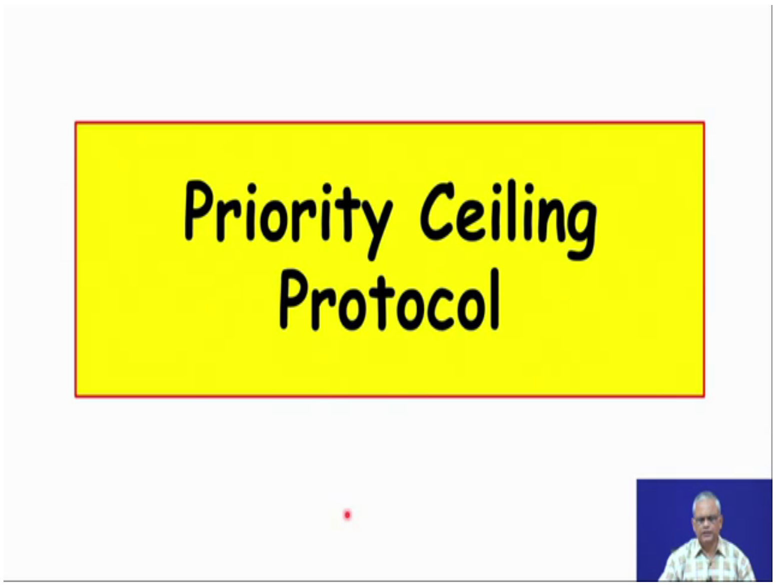


Shortcomings of HLP

- Due to the problem of inheritance-related priority inversion.:
  - HLP to be used cautiously in real applications.
  - Several intermediate priority tasks may miss their deadlines.
- Priority Ceiling Protocol:
  - Extension of PIP and HLP to overcome their drawbacks.
  - Can you list the drawbacks?

The inheritance related priority inversion as we have already mentioned is a severe problem, tasks not needing the resource the intermediate priority task undergo inversion, and the priority ceiling protocol overcomes this drawback to a large extent. By this time I hope that we have understood; what are the drawbacks of the basic priority inheritance scheme, and also the highest locker protocol.

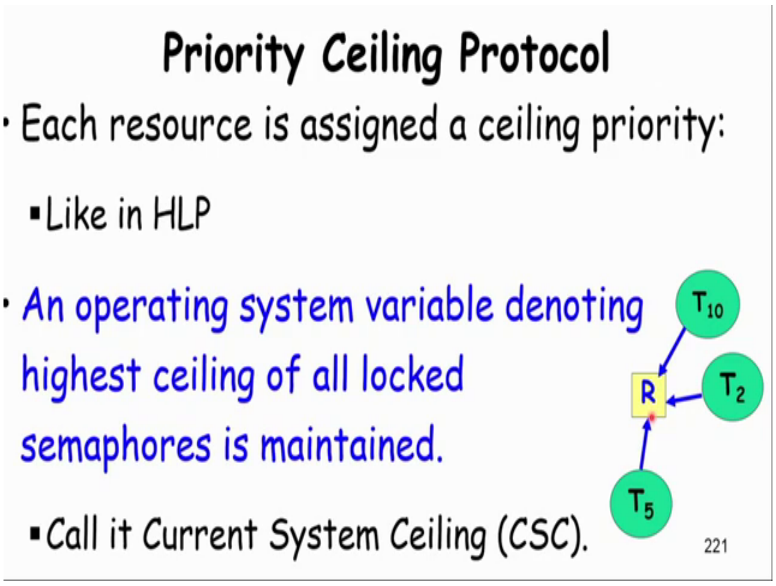
(Refer Slide Time: 10:34)



Priority Ceiling Protocol

Now let us look at the priority ceiling protocol, which is a improvement over the highest locker protocol and most of the sophisticated real time operating systems use the prio priority ceiling protocol or PCP very very simple operating systems, do not use PCP because the scheme is slightly more complicated, than the basic priority inheritance scheme in the highest locker scheme.

(Refer Slide Time: 11:03)



Priority Ceiling Protocol

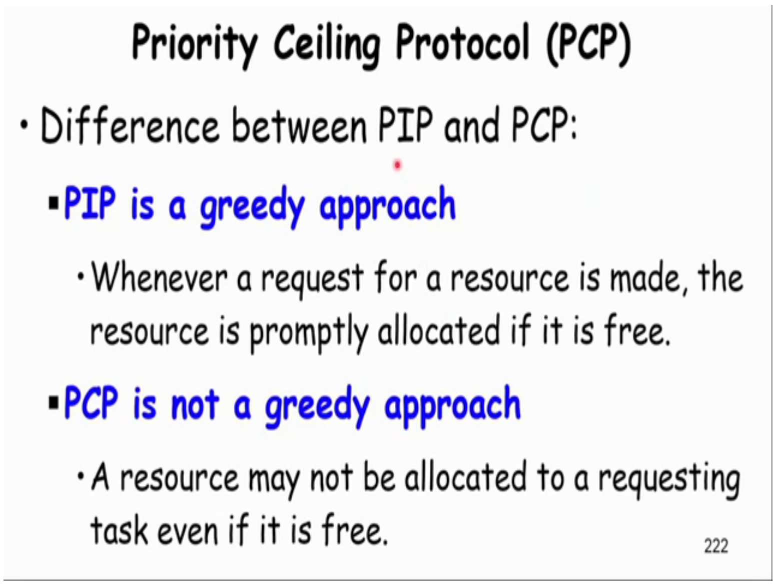
- Each resource is assigned a ceiling priority:
  - Like in HLP
- An operating system variable denoting highest ceiling of all locked semaphores is maintained.
  - Call it Current System Ceiling (CSC).

T_{10}
R
T_2
T_5
221

Now, let us see the main idea here just like the highest locker protocol, each resource is assigned a priority ceiling. And the main difference with respect to the highest locker protocol is that a tasks priority does not become equal to the ceiling priority of the resource as soon as it acquires a resource. But here its only a system variable the value becomes equal to the ceiling value, the tasks priority for acquiring a resource does not change, but only the a variables a system variable whose name is current system ceiling the value of that increases.

Let us look at this example if R is a non preamptable resource being used by three tasks T 10, T 2 and T 5 then the ceiling value associated with R is equal to T 2 and any task which acquires the resource, the tasks priority does not increase unlike the highest locker protocol, it continues with the same priority, but then there is a sys there is a operating system variable whose value becomes two when any of the task acquires the resource.

(Refer Slide Time: 12:49)



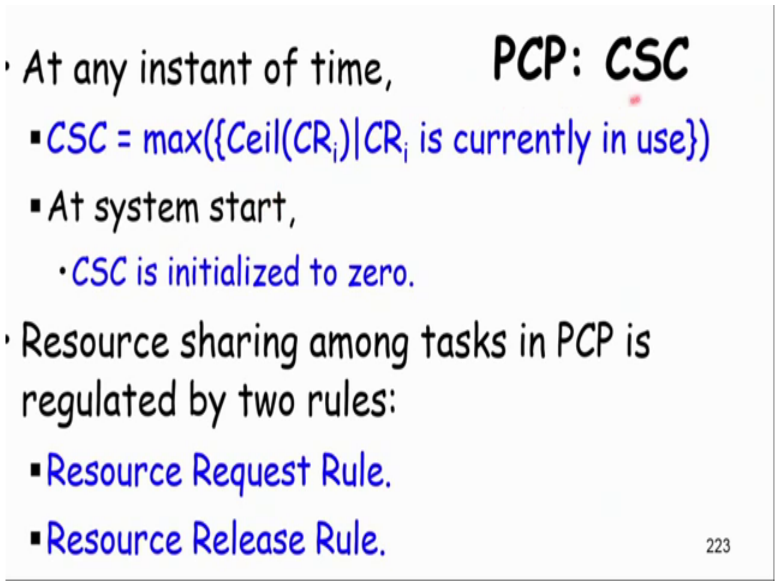
Priority Ceiling Protocol (PCP)

- Difference between PIP and PCP:
  - PIP is a greedy approach
    - Whenever a request for a resource is made, the resource is promptly allocated if it is free.
  - PCP is not a greedy approach
    - A resource may not be allocated to a requesting task even if it is free.

222

To look at the difference between priority inheritance protocol and priority ceiling protocol, the priority inheritance protocol is a greedy approach, whenever a task requires a resource it gets it if the resource is free. But in the priority ceiling protocol we will see that even if a resource is available a task executing requesting that resource may not get it under some circumstances. So, we can say that priority ceiling protocol is not a greedy approach in contrast to the priority inheritance protocol which is a greedy approach.

(Refer Slide Time: 13:34)



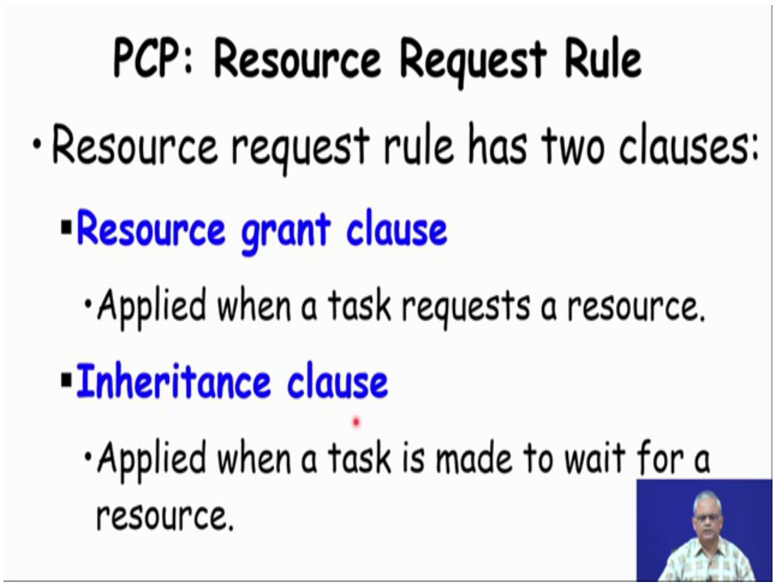
PCP: CSC

- At any instant of time,
  - $CSC = max(\{Ceil(CR_i) | CR_i \text{ is currently in use}\})$
  - 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.

223

Now let us look at the current systems ceiling, this is operating system variable, and whenever a task acquires a resource the tasks priority does not increase it is only the current system ceiling which gets the value of the maximum ceiling of all the resources that are being used. So, if C R i is the set of resources that are currently being used, then the c ceiling of the C R i maximum of that becomes the CSC. If there are two resources let us say C R 1 and C R 2 the ceiling value of C R 1 is 5 and the ceiling value of C R 2 is let say 2 and 2 is higher priority then if both resources are become used then CSC will be set to 2. At the start of the system the CSC is initialized to 0 now let us see how does the resource sharing occur under the ceiling protocol. There are actually two rules the first rule applies when a tasks requests a resource is called as the resource request rule, the second rule is called as a request release rule this rule now applies when a task releases a resource.

(Refer Slide Time: 15:08)

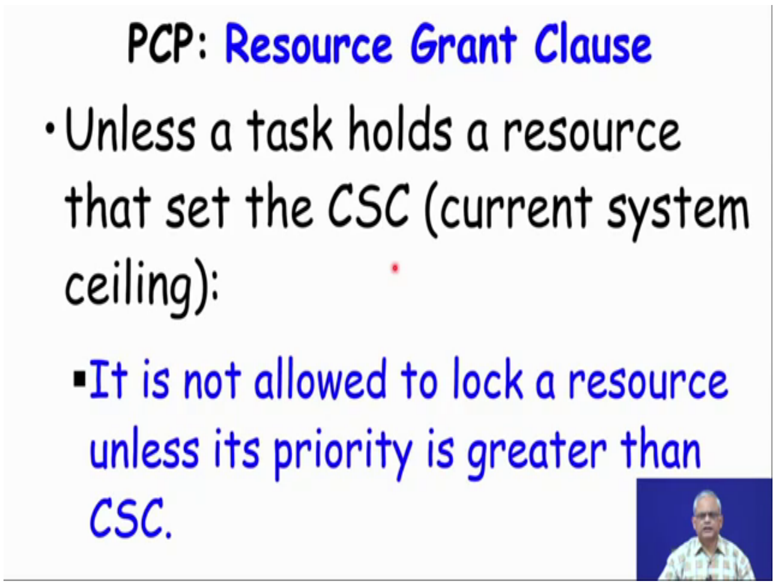


PCP: Resource Request Rule

- 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.

First let us look at the resource request rule; in the request in the resource request rule there are two clauses first is called as a request grant clause, where a task requesting a resource gets a resource, the second is called as the inheritance clause where a task does not get the resource, but it blocks. Let me repeat again in the resource grant clause this clause is applied when a task is granted a resource, where as a inheritance clause is applied when a task does not applied a does not get the resource it requested, but it blocks first let us look at the request grant clause.

(Refer Slide Time: 15:57)



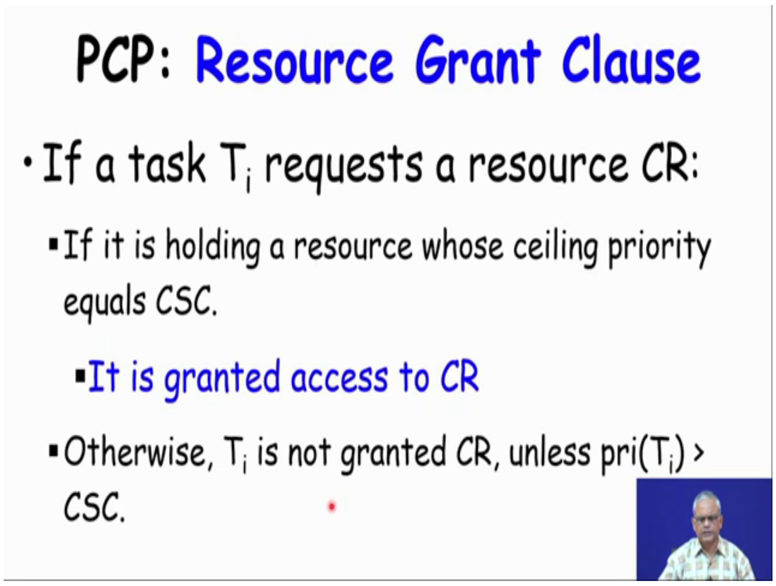
PCP: Resource Grant Clause

- 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.

Simple clause here once the task gets the resource, then the current system ceiling is checked with the ceiling of the resource it has got and if the ceiling is greater than the current system ceiling then the current system ceiling is set to the resource, and the task is not allowed to lock a resource unless its priority becomes greater than CSC. In the resource grant clause the tasks priority is checked with a current system ceiling, and if its priority is greater than the current system ceiling, then it is granted the resource, and the current system ceiling is set to be equal to the ceiling value of the resource that was granted.

Let me repeat again because this a important clause here, resource grant clause when a task requests the resource first its priority is compared with a current system ceiling; and if its priority is greater than the current system ceiling it gets the resource its granted the resource, and the CSC value is set equal to the ceiling priority of the resource. But if the task requesting the resource priority of the task is less than CSC, it is not granted the resource and it blocks.

(Refer Slide Time: 17:45)

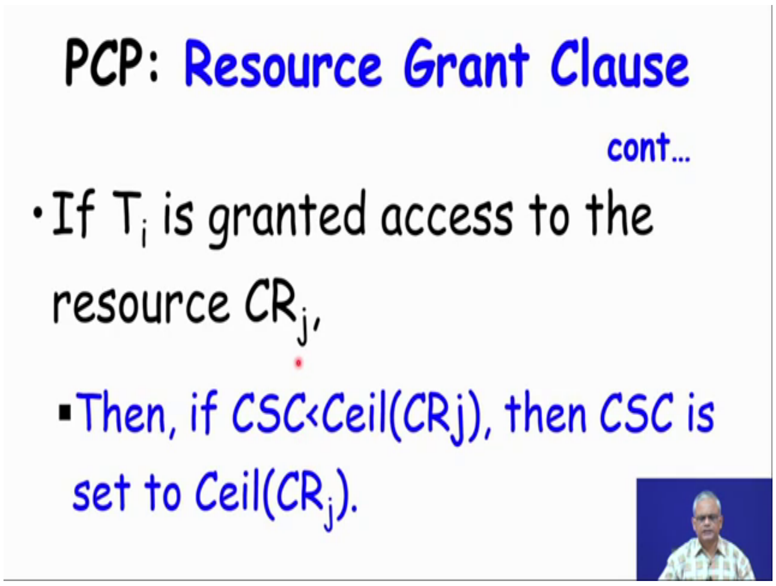


PCP: Resource Grant Clause

- 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.

So, in the resource grant clause, a task requesting a resource if it is holding a resource. So, ceiling priority is equal to CSC then it is granted access to CR. So, there are two clauses here in the resources resource grant clause, the first is if it is priority of the task is greater than the ceiling priority it is granted the re resource and the current system ceiling become equal to the ceiling associated with the resource. But even if its priority is not greater than the current system ceiling, but then if it is holding any resource whose ceiling priority is equal to the CSC, then also it is granted access to C R otherwise it is not granted access to the resource and it blocks.

(Refer Slide Time: 18:50)



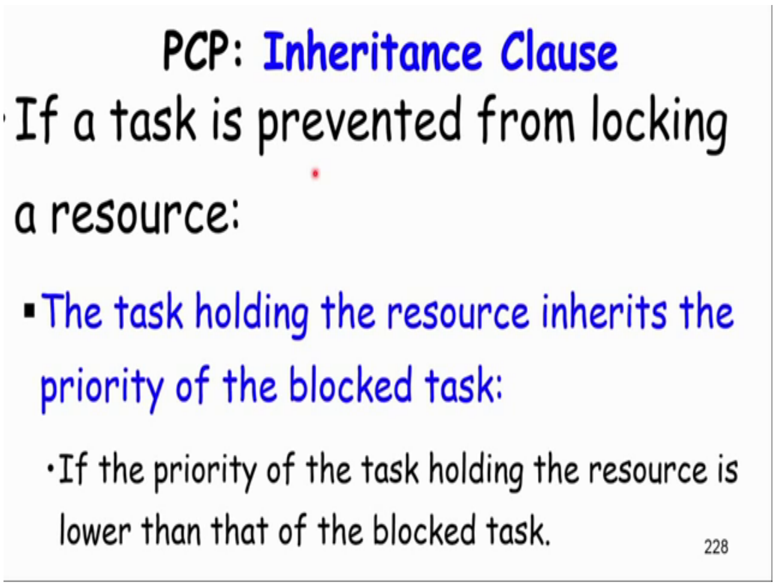
PCP: Resource Grant Clause

cont…

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

And as I was saying that as soon as task is granted the critical resource, the current system ceiling is made equal to the ceiling of the resource that is granted if the ceiling of the resource is greater than the current system ceiling.

(Refer Slide Time: 19:07)



PCP: Inheritance Clause

If a task is prevented from locking a resource:

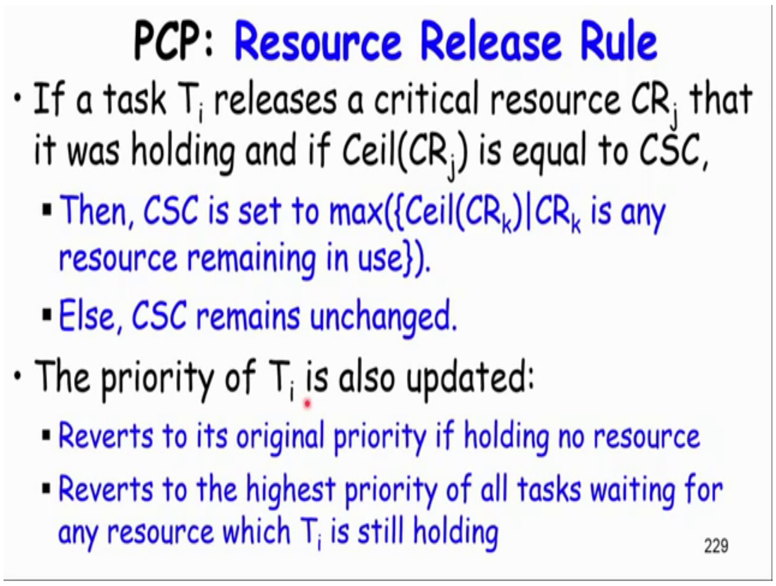
- The task holding the resource inherits the priority of the blocked task:
  - If the priority of the task holding the resource is lower than that of the blocked task.

228

Now let us look at the inheritance clause, if a task is made to block its not granted the resource, then the task holding that resource inherits the priority of the blocked resource.

Let me repeat again if a task blocks for a resource, then the task holding the resource inherits the priority of the blocked task. A task that has holding a resource does not change its priority unlike the highest locker protocol, but only when a task blocks per a resource it inherits the priority of the task.

(Refer Slide Time: 19:57)



PCP: Resource Release Rule

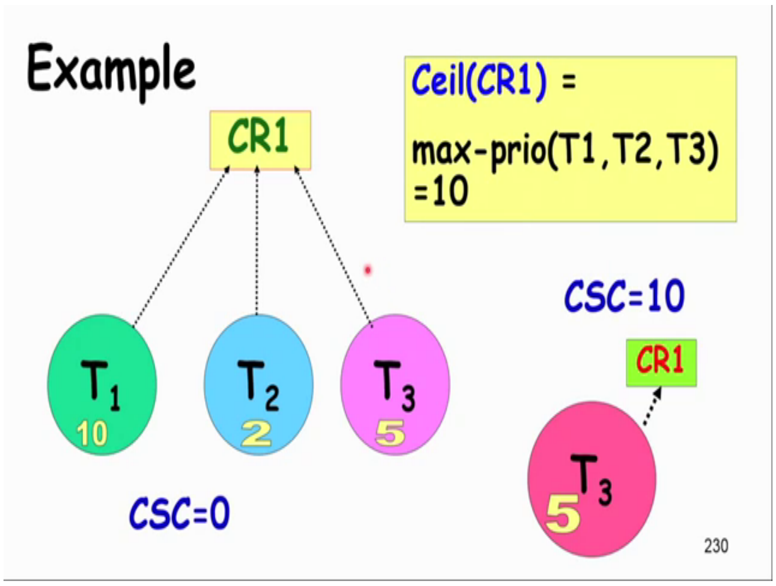
- If a task T_i releases a critical resource CR_j that it was holding and if Ceil(CR_j) is equal to CSC,
  - Then, CSC is set to max({Ceil(CR_k)|CR_k is any resource remaining in use}).
  - Else, CSC remains unchanged.
- The priority of T_i is also updated:
  - Reverts to its original priority if holding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

229

Now let us look at the resource release rule, its just the reverse of the resource grant rule. Here as soon as a task releases the resource, the CSC is again checked with all the tasks that are under all the resources that are under use, and the ceiling is set to the maximum of that. But if the current system ceiling is equal to the ceiling of the current the task sorry the resource that is released then the CSC remains unchanged.

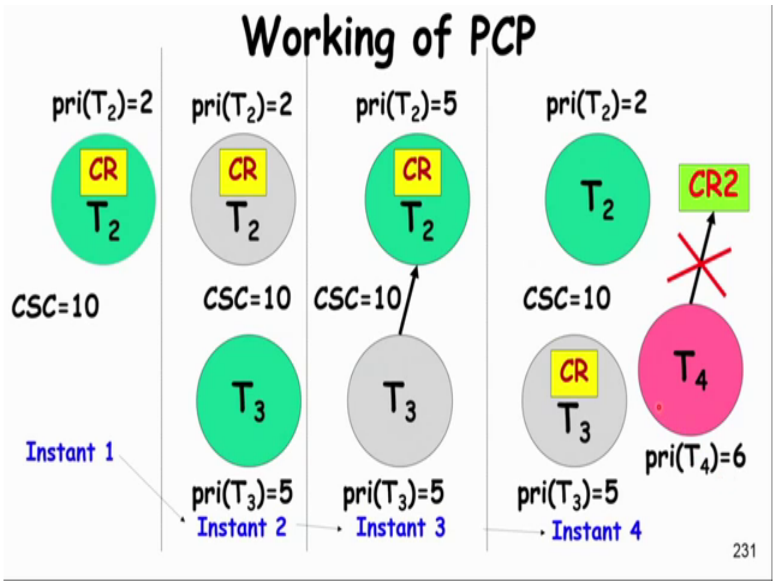
Let me repeat again the current system ceiling as you know it is the maximum ceiling of the resources under use, if a resource is returned and that is equal to the CSC, then all the resources are checked their ceiling values obtained and the maximum of that ceiling value is assigned to the CSC, but if the ceiling value of the current resource is less than CSC then CSC remains unchanged and also the priority of T 1 is updated if it is released a resource, then any task that was blocking on that resource it ma might have inherited the priority. So, the tasks priority in that case is reverted back to the original priority.

(Refer Slide Time: 21:44)



Let us look at the working of the ceiling protocols with some examples, this is the resource critical resource being used by three tasks T 1 ,T 2, T 3 priority of T 1 is 10, priority of T 2 is 2 and priority of T 3 is 5. And since three tasks can use C R 1 the ceiling priority associated with C R 1 is 10 assuming that 10 is the highest priority just like in a window based system to start with the current system ceiling is set to 0 and let us assume that T 3 locks C R 1. T 3 priority does not change unlike the highest locker protocol it continues with high priority, but then the current system ceiling is equal to the ceiling priority of C R 1 which is equal to 10. So, the CSC value only increases.

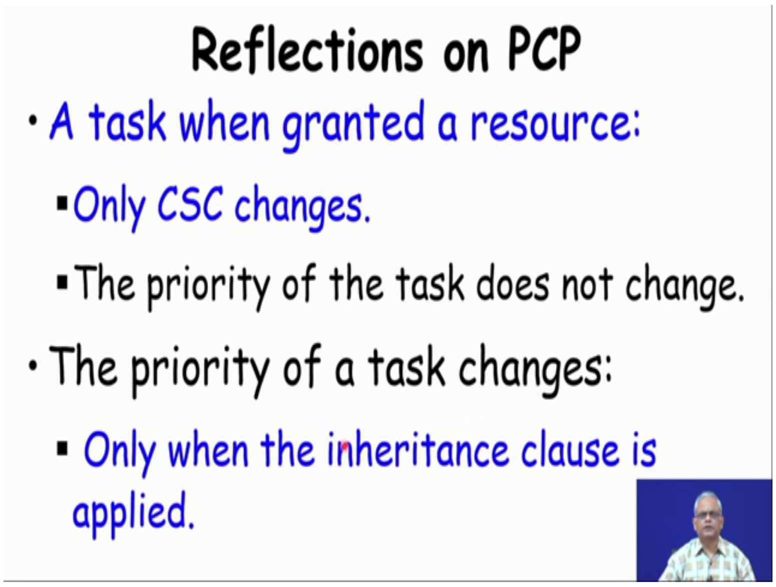
(Refer Slide Time: 22:58)



And let us say next time another task requests a resource, it is the task T 2 whose priority is two has acquired the resource and the current system ceiling is set to 10 and now let us say there is a task T 3, which is executing because t twos priority does not change it continues to operate at the priority level 2 and let us say a priority five task T 3 executes. And after sometime it requests T 2 and in that case the inheritance clause will apply and the priority of T 2 will become 5. And as soon as it releases the resource the release clause will apply and its gets back its older priority of two and the current system ceiling is still 10 because T 3 is being eh using C R. So, the ceiling priority of the resource is assigned to the current system ceiling and.

Let us say T 4 is another task which is needing a different resource C R 2 it will not be allowed to lock this resource because the priority of T r t 4 will compared to the CSC which is 10 even though the priority of T 4 is 6 which is greater than t three, but then it is requesting a resource which is idol C R 2 will not be granted because in priority grant clubs the priority of T 4 will be compared to the current system ceiling and since the current system ceiling is more than the priority T 4 will not be allowed to lock the resource.

(Refer Slide Time: 25:25)



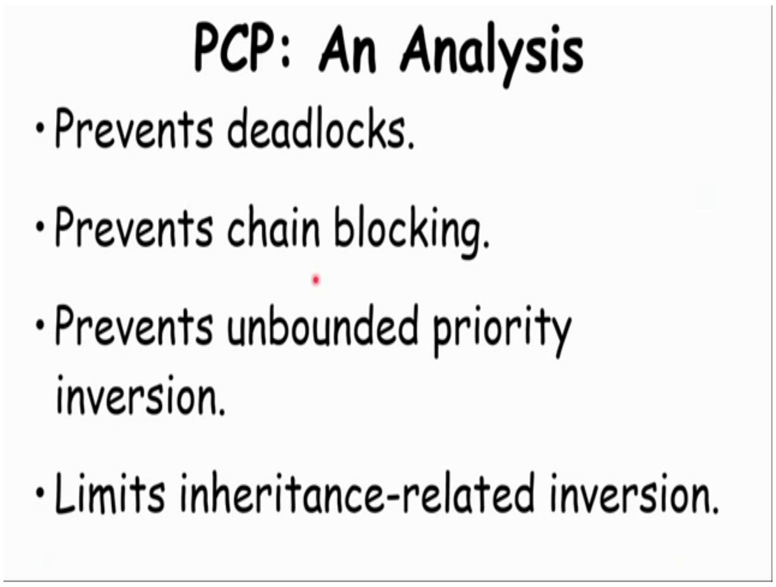
Reflections on PCP

- 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.

So, let us examine the priority ceiling protocol what really happened when a task is granted a resource, its priority actually does not change only the current system ceiling changes unlike the highest locker protocol, the task continues to execute at its own priority.

So, other tasks which do not need resource they do not get blocked, unlike the highest locker protocol were blocking by a task just because a lower priority task has acquired a resource does not occur and that we had called as inheritance related inversion the inheritance related inversion. In that sense does not occur here because as soon as a task acquires a resource its priority value does not change. The other tasks continue to execute as usual no inheritance re related blocking occurs. The priority of task changes only when another task requests the resource that is holding the inheritance clause becomes applicable and the task priority increases.

(Refer Slide Time: 26:50)



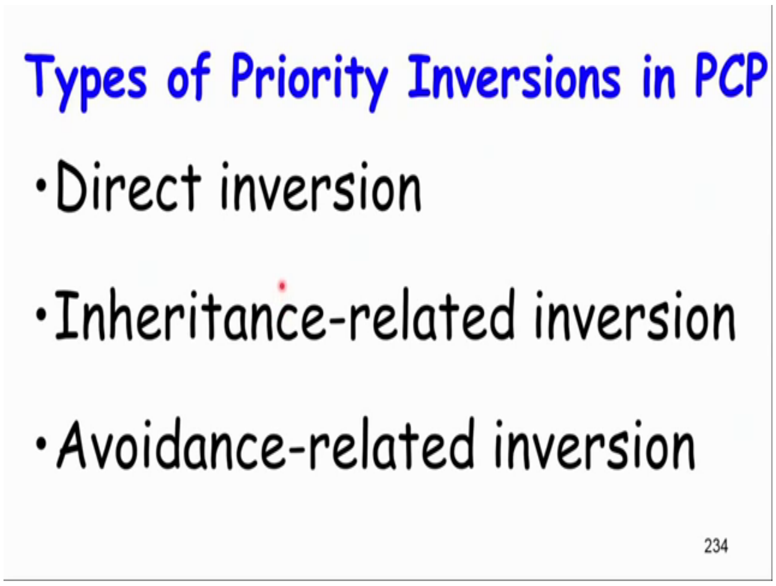
PCP: An Analysis

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

If we analyze the priority ceiling protocol, we will see that it prevents deadlocks, prevents chain, blocking prevents unbounded priority inversions and also limits the inheritance related inversion.

The inheritance related inversion is a big problem with highest locker protocol, and that is reduced to a large extent by the priority ceiling protocol and therefore, priority ceiling protocol though it is a slightly more complicated protocol then highest locker protocol, but is being used in many applications.

(Refer Slide Time: 27:34)



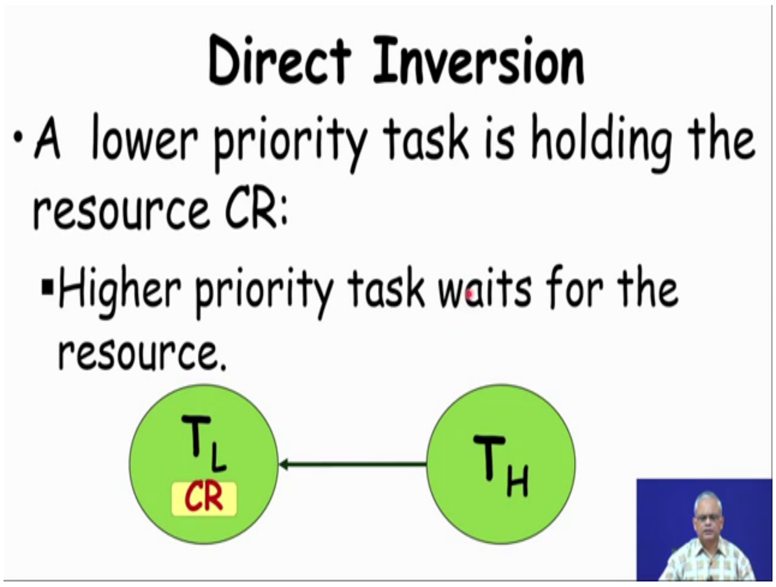
Types of Priority Inversions in PCP

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

234

Now, let us do a small analysis of the priority inversions that can occur in the ceiling protocol priority ceiling protocol if we look at the different types of priority inversions that can occur in the priority ceiling protocol, we will have three types of inversions one is called as direct inversion the other is inheritance related inversion and the third we call is avoidance related inversion.

(Refer Slide Time: 28:00)

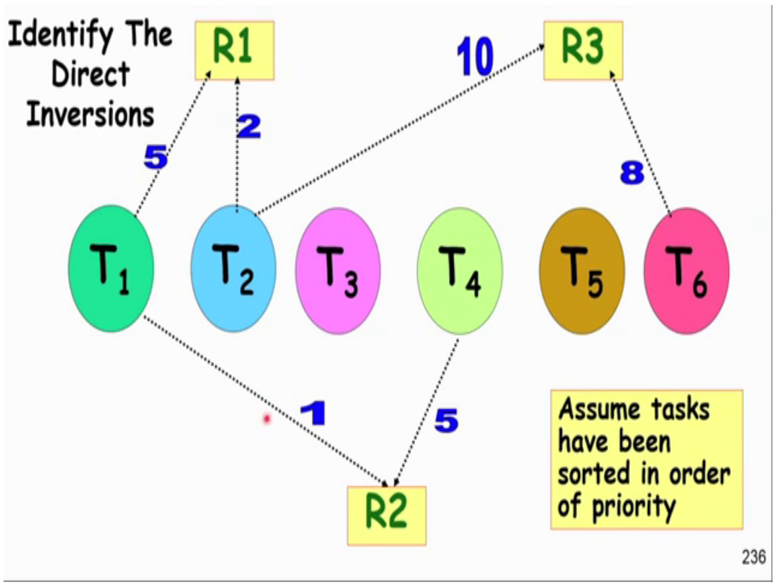


Direct Inversion

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

The direct inversion is simple, here when a lower priority task is holding a resource a higher priority task has to wait until the lower priority task releases the resource. Let me repeat again that in a direct inversion, a lower priority task is holding a resource the higher priority task has to wait a higher priority task needing that resource has to wait until the lower priority task completes its uses of the resource and releases it.

(Refer Slide Time: 28:41)



Identify The Direct Inversions
R1
R3
R2
10
5
2
8
1
5
T_1
T_2
T_3
T_4
T_5
T_6
Assume tasks have been sorted in order of priority
236

Now this is a small exercise here there are 6 tasks, and in this task graph what are the resources critical resources each task needs and what is the duration for which needs has been represented. T 1 and T 2 need R 1, T 1 needs it for five units T 2 needs for two units r three is needed by T 2 and t six and R 2 is needed by T 1 and T 4 for 1 and 5 unit respectively.

Now please identify the direct inversions that can occur, and assume that the tasks have been sorted in terms of their priority values that is T 1 is the highest priority and t six is the lowest priority. tTe direct inversion can occur in case the T 2 is holding R 1 T 1 has to wait even though T 1 is a higher priority and the time for which T 1 has to wait is two; on account of R 1 and T 2. Similarly T 2 has to undergo inversion direct inversion on account of T 6 and resource r three for a duration of eight similarly T 1 has to T 1 mean the worst case undergo 5 units of inversion on account of T 4. I hope that the direct inversion is clear that a higher priority task will undergo direct inversion account of a lower priority task holding the resource we are just running out of time for this lecture we will stop here and we will look at the other types of inversions that can occur in the priority ceiling protocol in the next lecture.

Thank you.