Real-Time Systems

Prof. Dr. Rajib Mall

Department of Computer Science and Engineering

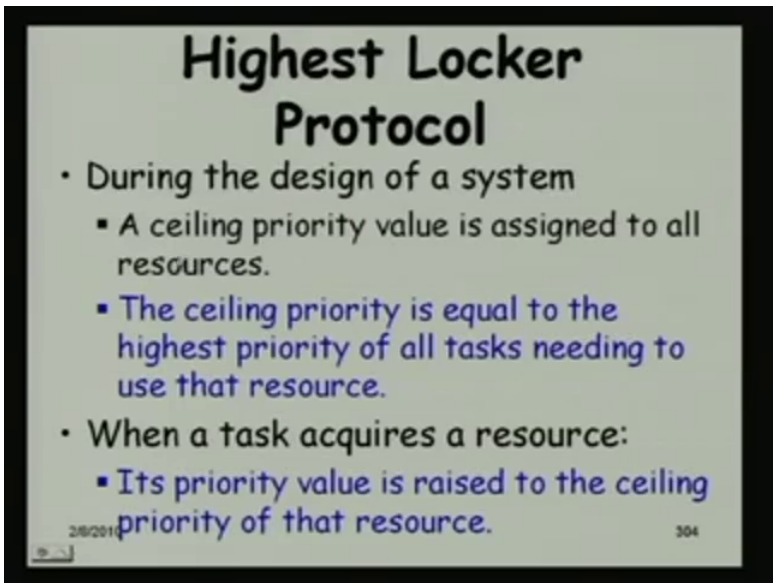
Indian Institute of Technology, Kharagpur

Lecture No. # 14

Highest Locker and Priority Ceiling Protocols

Good morning,let us get started from where we left last time. We were discussing about resource sharing among real time tasks and we examined variouskinds of problems that might arise.We saw the basic problem of priority inversion, and unbounded priority inversion. We had seen the simple protocol, the priority inheritance mechanisms, and the basic mechanism for supporting resource sharing among real time tasks.

(Refer Slide Time: 01:04)

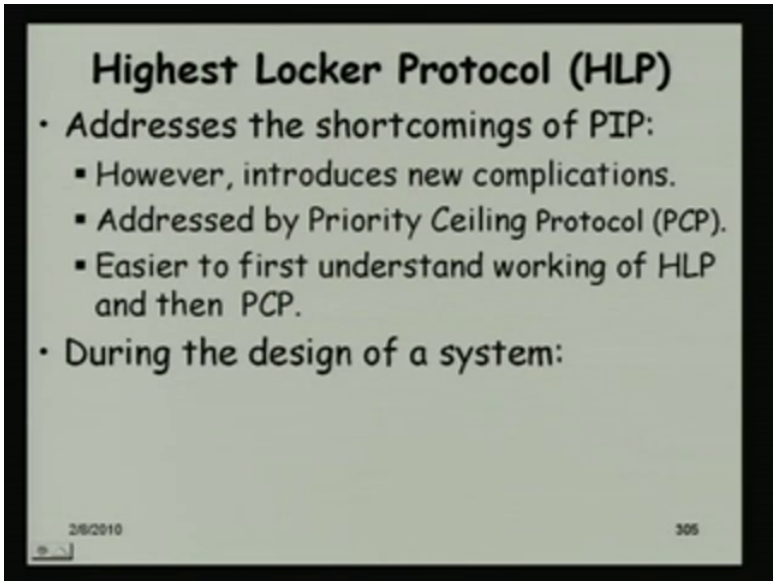


In the highest locker protocol,the ceiling priority value of a resource is the highest priority of all tasks needing to use that resource.

We have to know before what are the tasks that would be using a resource, and then set the ceiling priority of that resource equal to the highest of the task priorities that aregoing to use the resource. Then, whenever a task acquires a resource, its priority value is automatically raised to the ceiling priority of that resource.If it acquires multiple resources then it will be

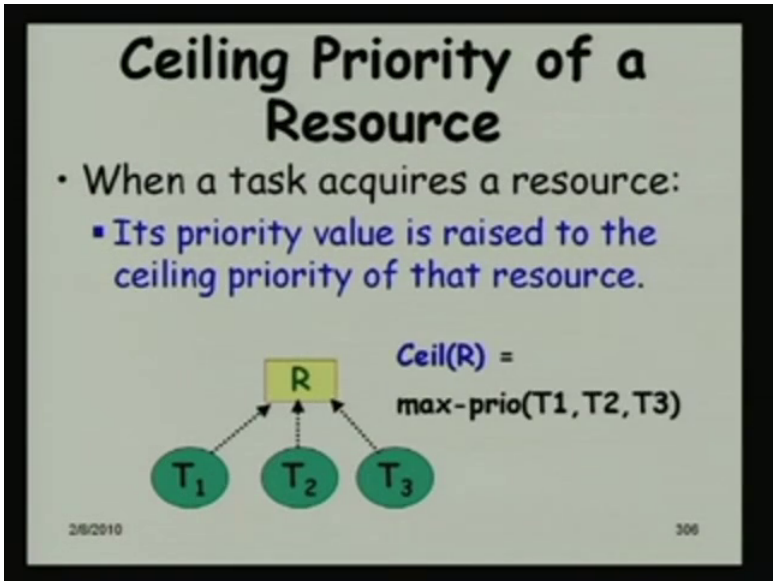
naturally the highest of all these priorities. It addressesthe soft comings of the basic inheritance scheme.

(Refer Slide Time:02:03)



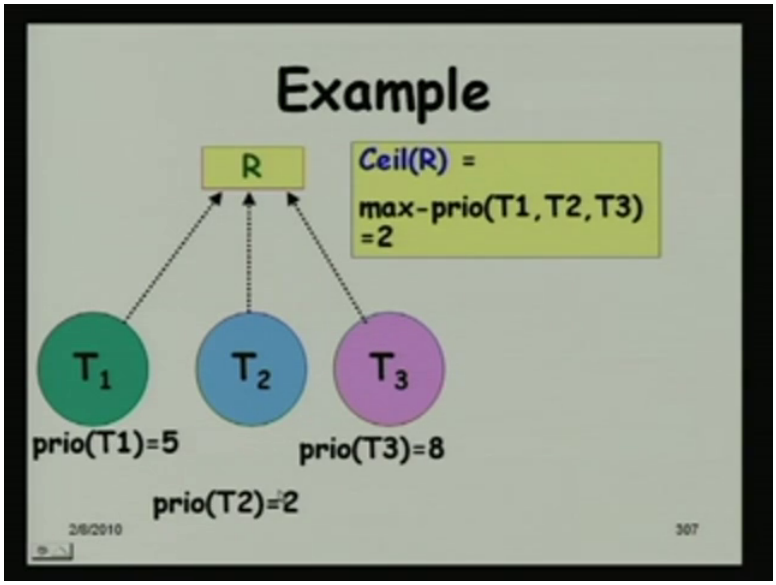
Unfortunately, the highest locker protocol introduces new problems. Therefore, highest locker protocol by itself is not used. But it gives us the basic understanding based on which we can develop the priority ceiling protocol which is the one which is actually used.

(Refer Slide Time: 03:27)



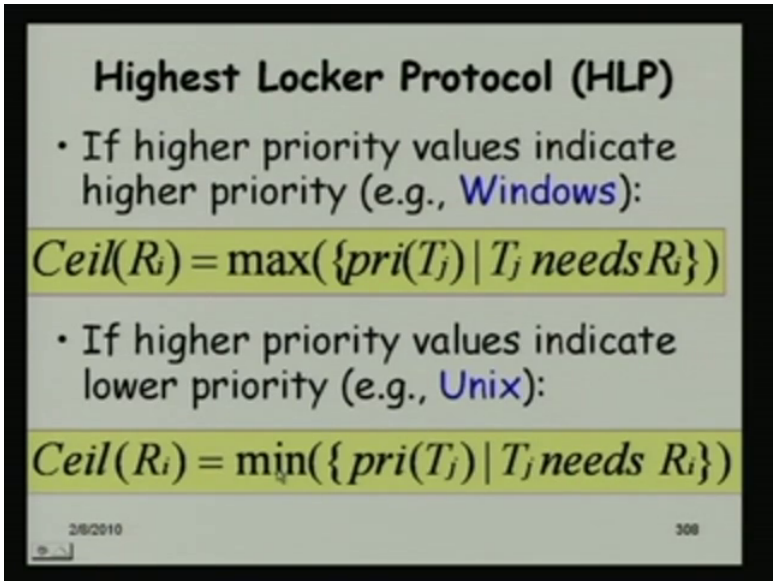
During the designof the system, the ceiling priority is assigned to all critical resources and the priority is equal to the highest priority of all tasks that might use the resource. This is the situation where resource Ris being used by tasks T1, T2, T3. Rwill be associatedwith the maximum priority of T1, T2,T3.

(Refer Slide Time: 03:50)

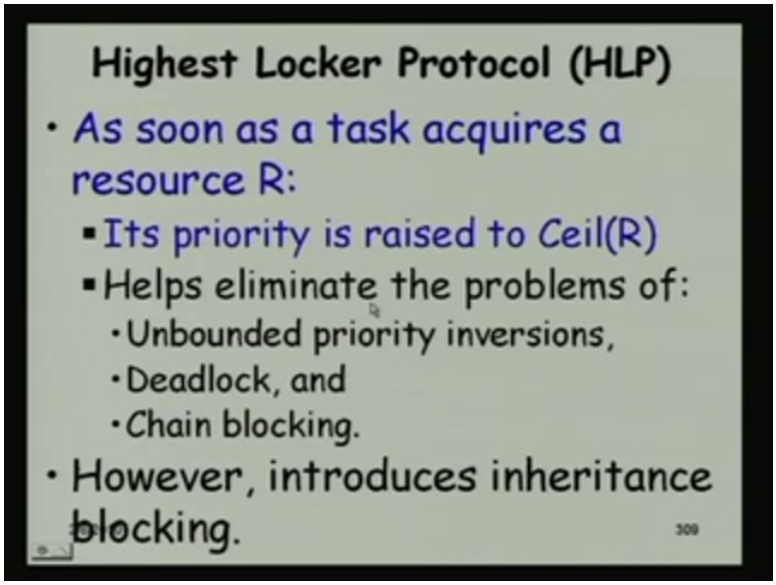


Example:We have priority of T_1 is 5, and the priority of T3 is 8, and priority of T_2 is 2. Then, the maximum priority will be 2 and ceiling value associated with R will be 2. Whenever a task acquires R, if T_3 acquires R, its priority will become 2.

(Refer Slide Time: 04:33)

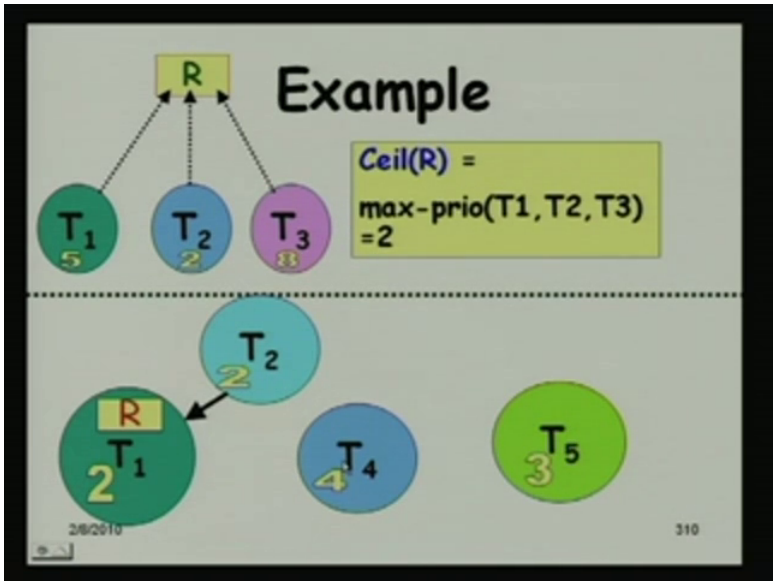


(Refer Slide Time: 05:05)



As soon as the task acquires the resource,its priority is raised to the ceiling value, and it will help to eliminate the unbounded priorityinversions, the dead lock and the chain blocking. But it would introduce a new problem which will call as the inheritance blocking.

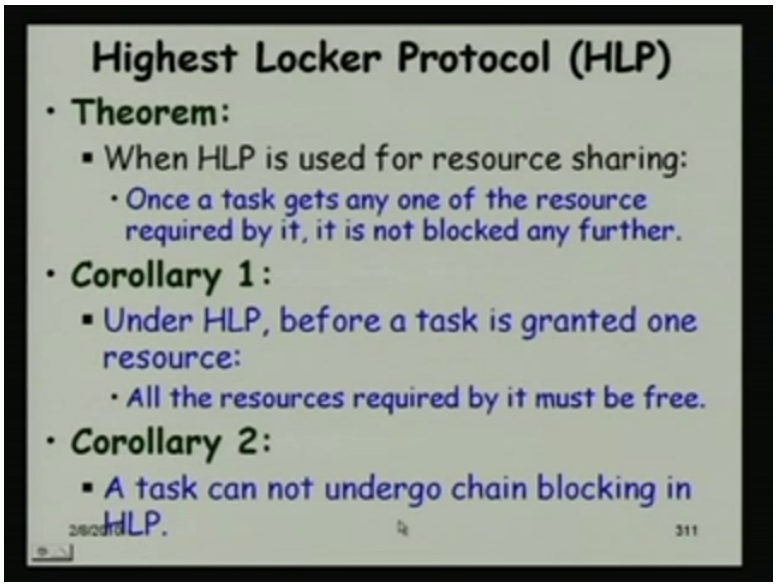
(Refer Slide Time: 05:28)



This isan example of the working of the algorithm. T_1, T_2, T_3 are sharing the resource R, the ceiling value associated with R is 2, because these have priorities 5 to 8 and the maximum of this is 2. T_1 acquires R, T_1's own priority is 5 and as soon as it requires R, its priority would become 2. If T_2 gets ready by that time then it would have to wait.

The task 5,its priority has become 2it is executing,whereas, T2whose priority is 2 is waiting, but the thing is that unbounded priority inversions are avoided. T 4does not need any resource cannot preempt T 1 from CPUuses, because T 1's priority has become more, similarly, T 5.

(Refer Slide Time: 06:56)



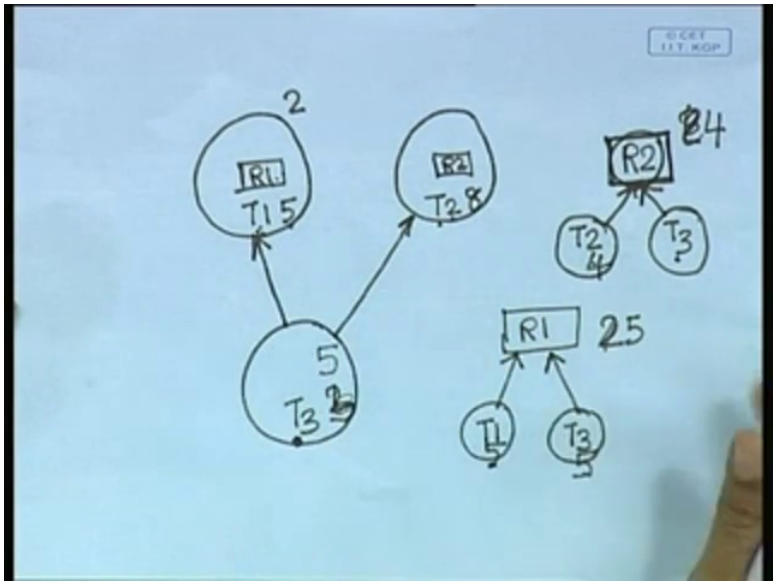
According to the theorem,when highest locker protocol is used for resource sharing, once a task gets any one of the resource, it will not block any further. In other words, there is no chain blocking.Under highest locker protocol there is no chain blocking. There are 2 corollaries of this theorem.

Corollary 1: Once a task is granted a resource, all resources required by it must be free. A task is granted one resource under HLP when none of the resources could have been held by any other task. If task gets one resource, then other resources must be free at that time.

Corollary 2: It cannot undergo chain blocking. Once it is get a resourcethen it will not be blocked any further.

We will prove it by contradiction. Assume that there are two taskswhich are holding resources R1 and R2. We have a higher priority task. That is T 1, T 2 and T 3.Let us assume T 3 is higher priority that way the priority inversion will occur.

(Refer Slide Time: 08:43)

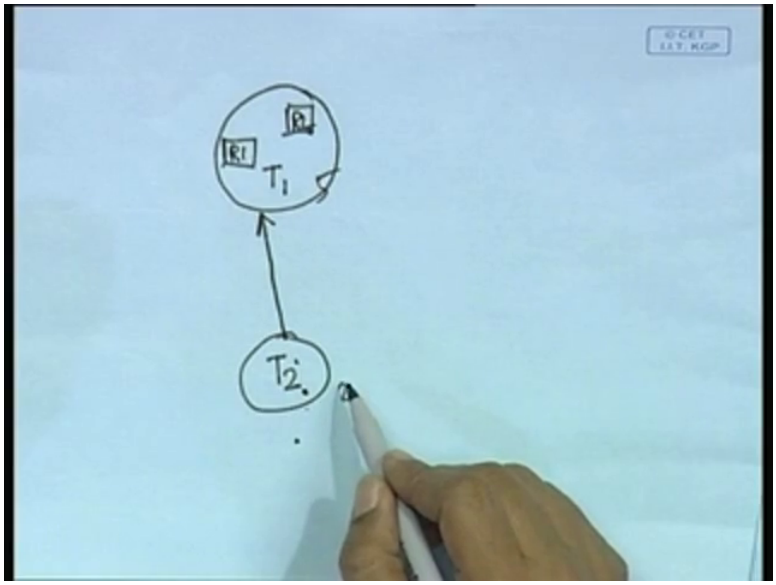


As soon as T_1 acquires R1 its priority will be raised to the ceiling priority menu. That is the basic protocol. According to this scheme, the resource R1 is required by T_1 and T_3. Similarly, resource R2 is required by T_2 and T3. We are saying it a rectangle, let us follow that convention. R 2 is required by T_2 and T3. Assume that its priority = 5, priority = 8 and priority = 2 respectively for T1, T2 and T3. The ceiling priority of R2 will be maximum of 2 and 8. Because R2 is used by T_2 and T_3.The ceiling of R2 will be T_2. Similarly, the ceiling of R1 will be 2.

Assume that T 2 blocks forresource R1, and then it gets the resource, and then computes and by that time T 2 is holding R2, and then it again as it requires R2 it againblocks.When T 1 acquires R1,its priority must have been set to 2.And if its priority were set to 2, then it is not possible that another task can acquire a resource R2, because it is alower priority task to acquire R2, because it is already a higher priority,its priority has already increased to 2 after acquiring the resource.

As soon as its priority,would have it would have acquired the resource its priority would have got raised to 2, there is no chance that T 2 could have acquired R2.Similarly, we can show that if T 2started executing and acquired R2. R1 could not have possibly been acquired by R1sorry R1 could not have possibly been acquired by T 1.

(Refer Slide Time: 16:41)

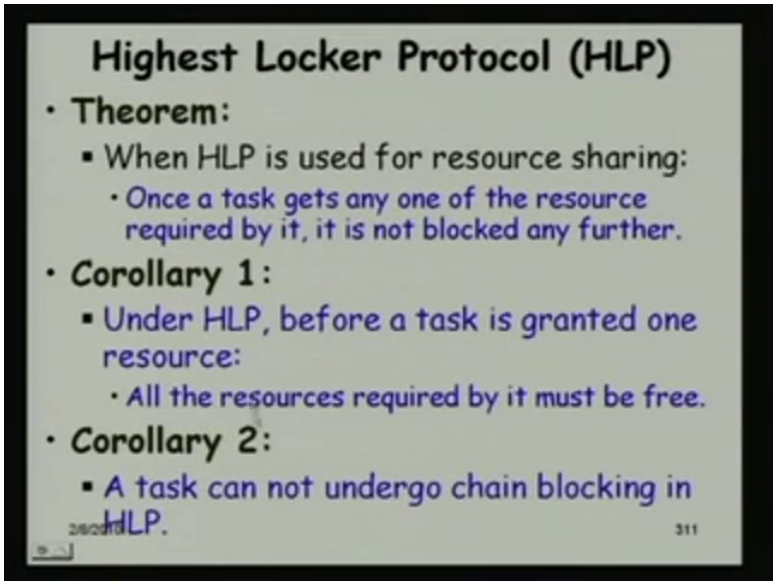


If a task is holding both the resources,that is also a possibility that - a task T 1 was holding both R1, and R2 and T 2 which is a higher priority task. E.g. T 2's priority is 2, and T 1's priority is 5. So, T 1 was holding both R1 and R2, and T 2 was blocking on R1.

First time T1 has blocked for R1, and then after some time it completed using R1, and then T 2 started executing and again it block for R2, because it also needed R2. Again it started blocking for R2,so two times it has blocked.But that also are not possibly, exactly that is not possible.Because as long as it has it is holding some resource R2 which is required by T2, any resource required by T 2its priority will become at least as much as this. Unless it releases both these resources, it completes execution of both these resources, T 2 will not get a chance to execute.

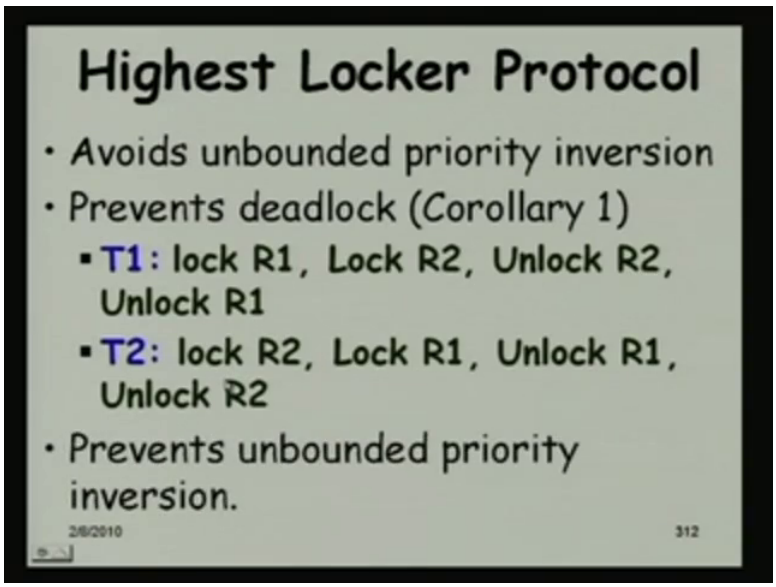
Once it gets one of its resources, it will not block any further for any other resource.Or in other words, chain blocking is not possible under highest locker protocol. So, you can try it more examples, and then look at the formal proof. See, that the proof outline is just similar excepting that - we just generalize instead of taking instances 5, 2, etcetera we just generalize right. Or if you find the better proof - easier proof - you just report it to me. We will consider that for a bonus mark.

(Refer Slide Time: 18:46)



We know thatunder highest locker protocol, chain blocking is not possible. Once it gets one resource,other resources must be available.

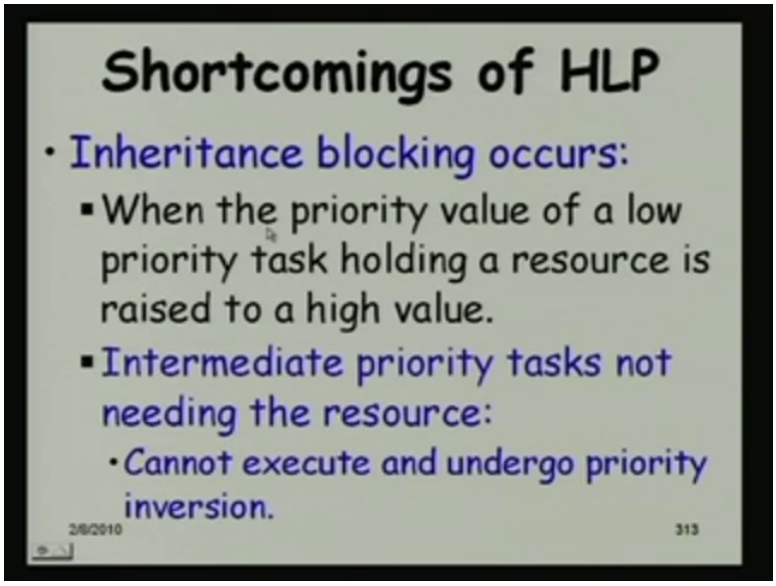
(Refer Slide Time: 19:00)



We have seen thatit avoids unbounded priority inversion. From corollary 1 said that once a task gets one resource, other resources must be free. Based on that we can showthat there cannot be a deadlock. Because it has got one resource, all other resources are free.

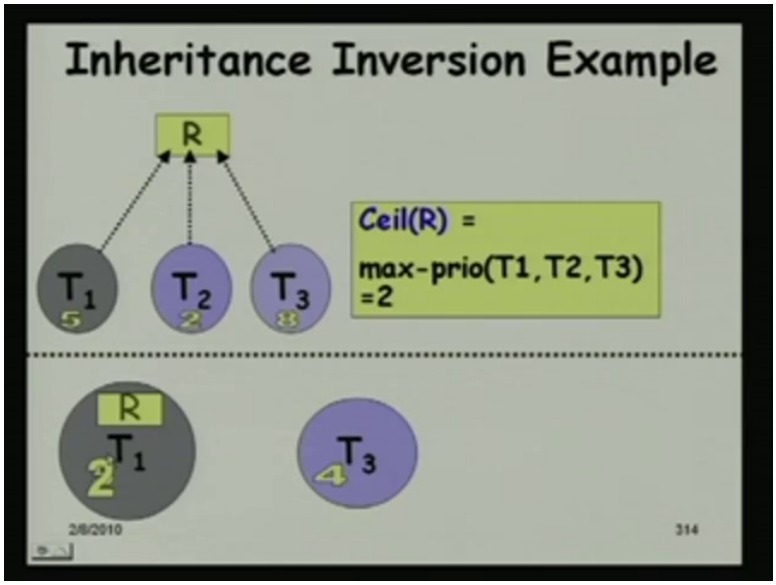
This is the example where the basic inheritance scheme was getting in to deadlock. T1 and T2 are two tasks requiring resources R1 and R2, and T2 is a lower priority, T1 is a higher priority. T2 started executing and locked R2, and then T1started executing and issue the command lock R1, but that is not possible. Because as soon as T2 has acquired R2, the ceiling priority of R2 must be as much as the priority of T1 and therefore, this assumption that it will lock R1 is not possible right. So, dead lock is prevented.

(Refer Slide Time: 20:28)



The main problem of the highest locker protocol is the inheritance related inversion or inheritance blocking. When a low priority task gets a resource its priority is raised to high value, and then the tasks whose priority is more than this low priority task which are the intermediate priority tasks, even though they do not need any resource.They will be prevented from execution, because the low priority tasks priority has increased right. This is called as the inheritance blocking.

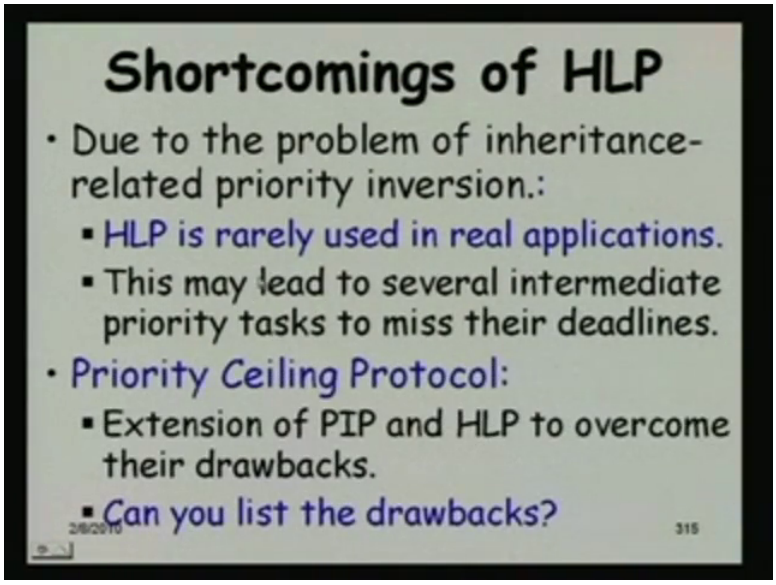
(Refer Slide Time: 21:10)



This is an example of an inheritance blocking.We have three tasks with priority 5 to 8. The ceiling priority of the resource is 2. T 1 whose priority is 5 acquired R, priority becomes 2 by the highest locker protocol, andthen T 3 priority is 4it is more than T 1, cannot execute, even though it is ready cannot execute. We should have named this task as T 4 or T 5, because T 3's priority is 8.So, it should be T 4 or T 5 something whose priority is 4 cannot execute. It is undergoing inheritance-related inversion. Similarly T 5,its priority is 3 cannot execute, because its priority has become 2. So, these tasks are undergoing inheritance inversion until T 1 completes. What is the duration for which T 3 and T 5 might undergo inheritance inversion?

The duration for which T 1 needs the resource R -that is the duration for which T3 and T 5 will undergo inheritance inversion.

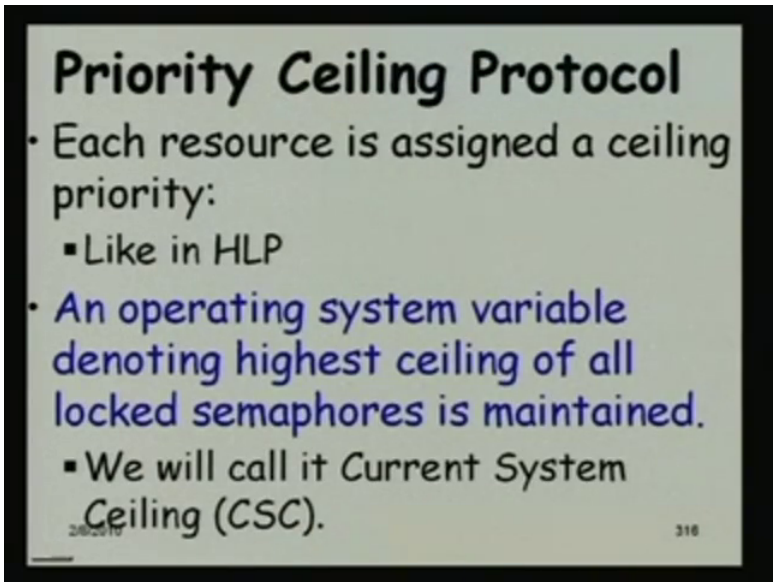
(Refer Slide Time: 23:11)



There is a chance that the lowest priority task will become the highest priority task, and then it blocks everything else.For example, a logging task which keeps the logs, it can become the highest priority task, and even the most sensitive tasks will be blocked.This can occur for many low priority tasks,rarely used in real applications.

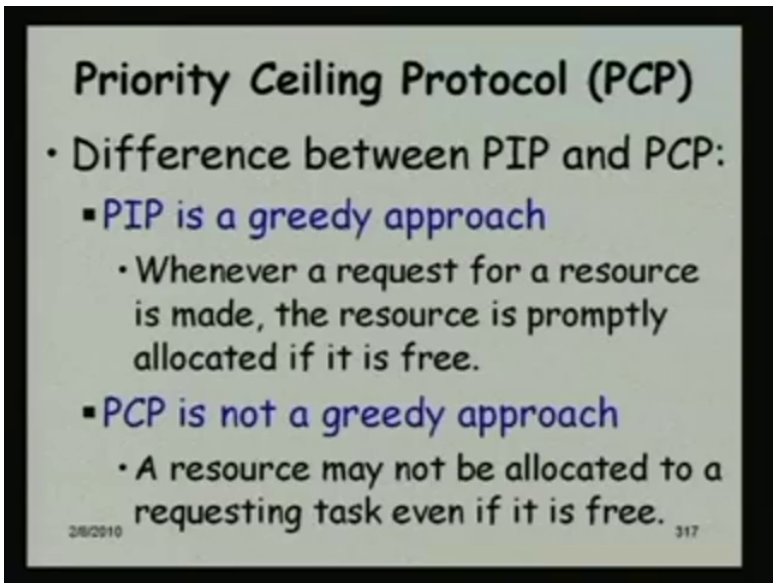
The problems of highest locker protocol are overcome by the ceiling protocol to thepriority ceiling protocol.It isan extension of the highest locker protocol. Even though it avoided the unbounded inversion, and deadlock, and chain blocking but it had a new problem i.e. the inheritance related inversion.Let us see how the priority ceiling protocol overcomes that.

(Refer Slide Time: 24:28)



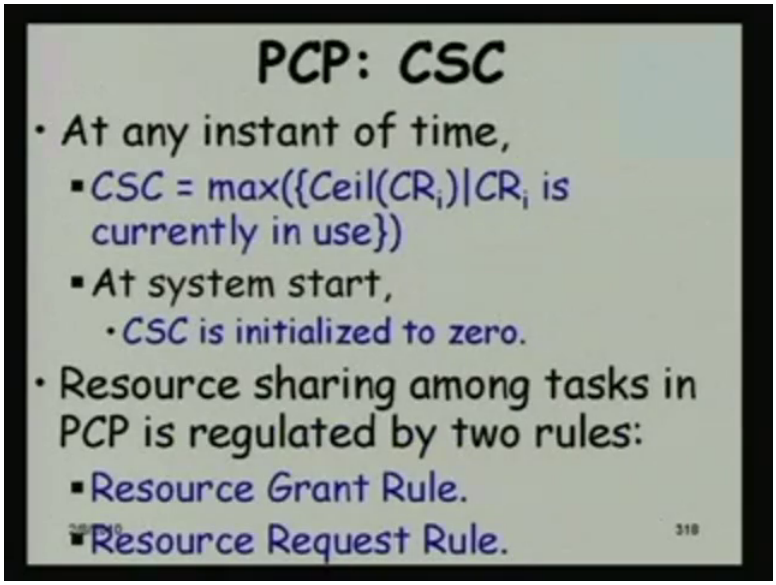
A ceiling priority is computed for every resource. But in the highest locker protocol, whenever a resource was acquired by a task its priority was raised to the ceiling value. In this we have an operating system variablewhich denotes the highest ceiling of all locked semaphores. If R1,and R2are two resources that are been used and they have ceiling value of C1 and C2. Then, the operating system variable will get maximum of C1 and C2. The operating system variable will call as the current system ceiling or CSC, it stores the highest of all ceiling protocols of the resources under use.

(Refer Slide Time: 28:09)



The priority inheritance protocol is a greedy approach. Because whenever a request to a resource is made as long as the resource is available, it will be allocated. If it is being used by some task, then it might block, and then the inheritance scheme applies. But as long as the resource is there and some task requests it, it gets it.So that is a greedy approach.In the priority ceiling protocol, a resource may be there, but the task will be prevented from acquiring it. So, it is not a greedy approach.

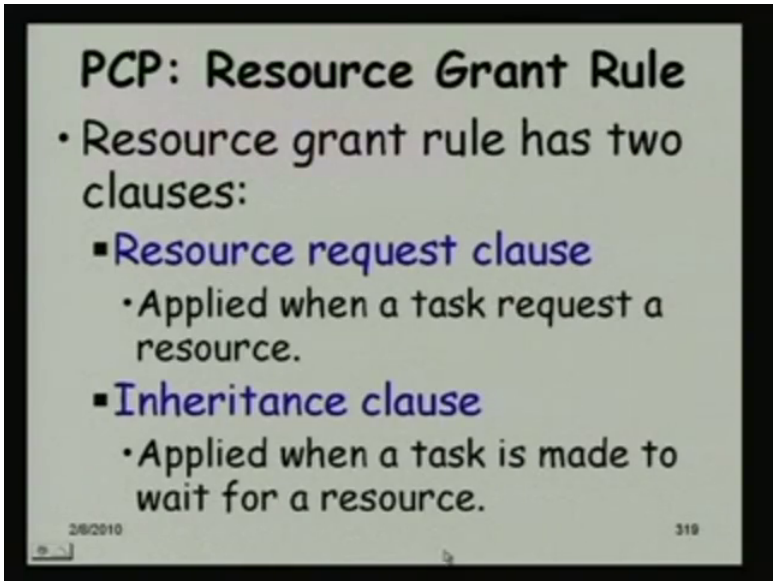
(Refer Slide Time: 29:22)



The current system ceiling as we were saying is set to the maximum of all ceiling values of the resources in use, we just a formal writing that -the systemthe current system ceiling is equal to the maximum of all ceiling values of resources that are currently in use.And to start with, the current system ceiling is initialized to zero, because no resources are being used. Assuming that zero means that nothing is being used. Or we can even set it to a very large value depends on the kind of convention we are following.

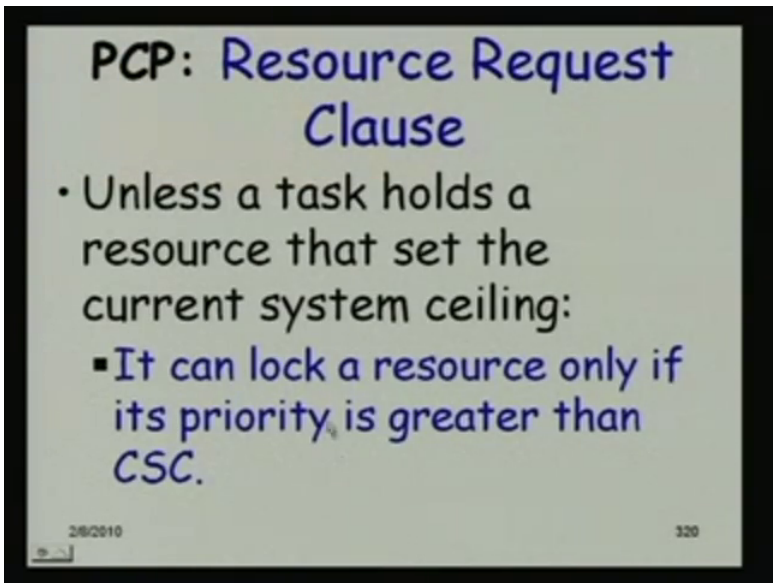
The priority ceiling protocol basically has two rules. One is the resource grant rule and the resource release rule.

(Refer Slide Time: 30:31)



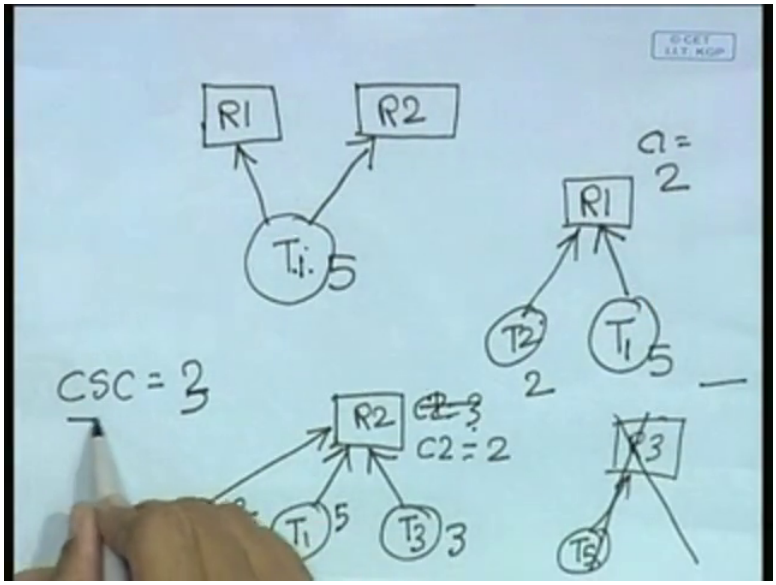
The grant rule has two clauses. When asource is to be granted first the request occurs, and then there is an inheritance clause that is applied.

(Refer Slide Time: 31:01)



A task holds a resource that set the current system ceiling.It cannot lock a resource, unless its priority is more than the current system ceiling.

(Refer Slide Time: 31:55)



We have a situation, where we have a task, which needs two resources R1 and R2. R1 and R2 are needed by T_1. And the R1 is also used by T_2. T_1's priority 5, and T_2's priority 2. So, the ceiling priority of R1 will become 2. R2 is being used by T_1 and also T_3, and T_1's priority is 5 and T_3's priority is 3 or maybe you can consider 8 or something. So, R2's ceiling priority. C1 is 2 and here C1 is equal to 3.

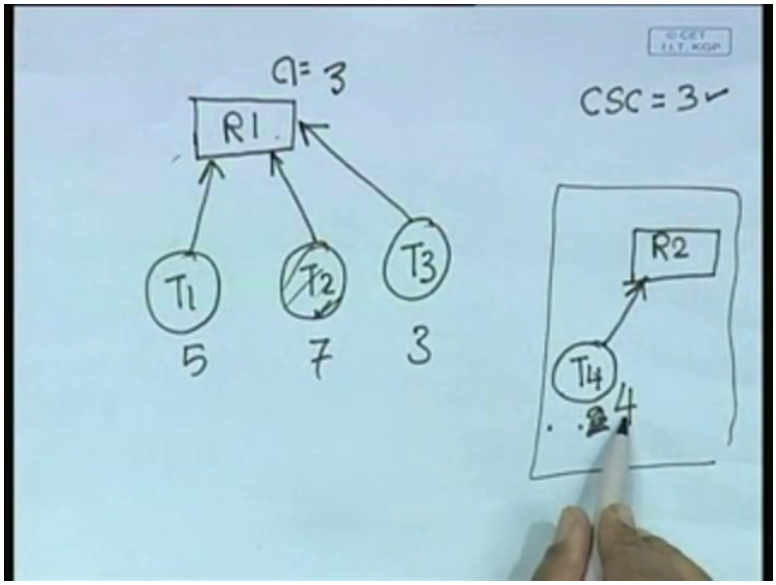
T_1 first acquired R2. The current system ceiling will be set to 3. T_1 first acquired R2, R2 ceiling priority is 3. The current system ceiling will be made 3. After sometime T_1 wanted to lock R1, and then it will be directly granted access to R1, the current system ceiling will not be looked at. It will because it is the one which had set to the current system ceiling.

The CSC is not a consequence, but T_1. But let us say, we have the task T_2, another task, let me just give R3 which is being used by T_5. T_5 if it request R3, its current its priority will be checked T_5's priority will be checked with the current system ceiling. And if it is priority is less than the current system ceiling, it will be prevented from acquiring the resource that it needs. Rather than complicating with R3, assume that T_2 whose priority is 2 also needs R2. I think that will. T_2's priority is 2. So, R2's ceiling will become 2, maximum of all these priorities. T_1 first acquired R2; its priority will become 2.

The priority is do not change, the priority of T_1 do not change. Remember that under CSC the priority does not change upon acquiring a resource, unlike the highest locker protocol. Only

the current system ceiling is set to 2. T_3 wants to acquire R2, because its priority was not increased -T_1's priority was not increased by acquiring R2. T_3 is started executing, because it is priority is 3. And after sometime, it wanted to use resource R2, then T_3's priority will be checked against T_2, it is higher priority than 3. T_3 will be prevented from locking.

(Refer Slide Time: 37:01)



Example: Resource request clause in the grant procedure. We have resource R1and it is required by three tasks, T1,T2 and T3. And their arbitrary priority values 5,7, 3respectively.

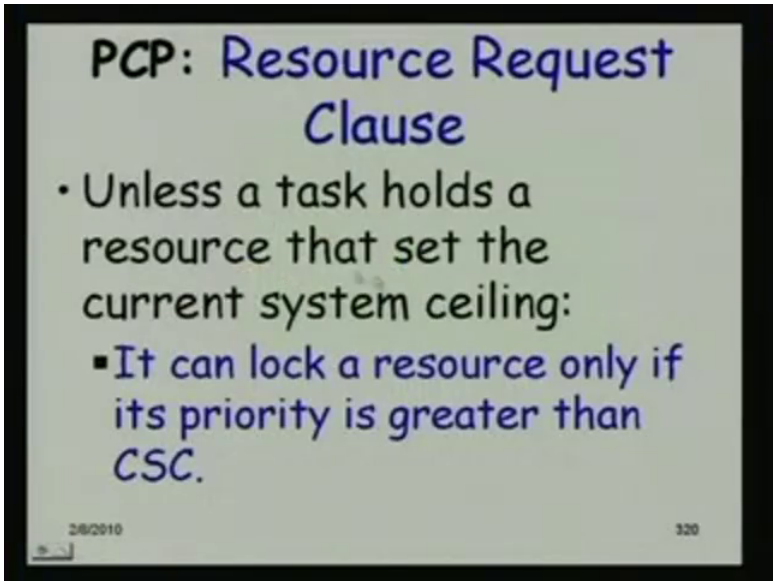
The ceiling value of R1 will be3. As soon as any task acquires the resource, this current system ceiling will be set to the ceiling value. CSC will be made 3,assumeT2 has acquired the resource. But T2's priority does not change.It is possible that another task T4 whose priority is 2 or 4, it will start executing, because it has the priority more than T 2.T 2 has acquired R1 and T 2's priority has not changed,T 2 has acquired R1 and the current system ceiling is set to 3, and T 4 priority 4 more than 7,T4 starts executing.

Consider the example where T 4 also needs resource.T 4 requests for R2 then T 4's priority will be checked with current system ceiling, and then if T 4's priority is larger than the current system ceiling it will be granted R2. But it is not here their priorities only 4. It will be prevented from locking R2.

The basic idea is that intermediate priority tasks are notblocked.They are allow to execute; unlike,thebasic the highest locker protocol where these tasks were undergoing inheritance-

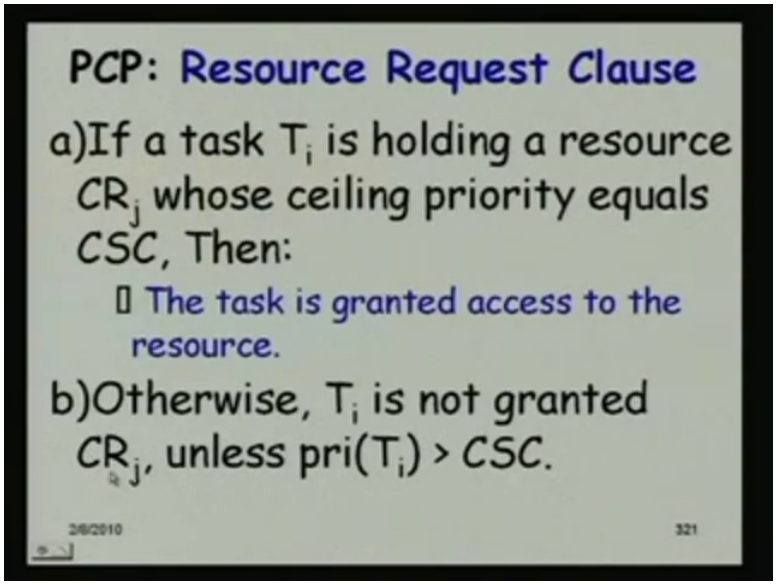
related inversion. They do not undergo inheritance related-inversion; no priority inheritance is applied till now.It is only the current system ceiling is changed.

(Refer Slide Time: 43:24)



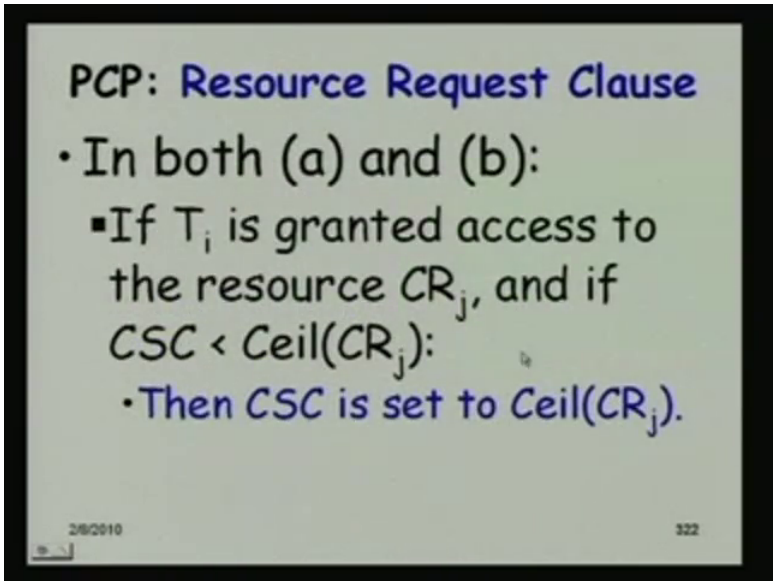
We discuss the the other clauses. The first one is the resource request clause. We know that unless a task holds a resource that set the current systemceiling. It will be prevented from locking any resource except when its priority is greater than CSC.

(Refer Slide Time: 44:02)



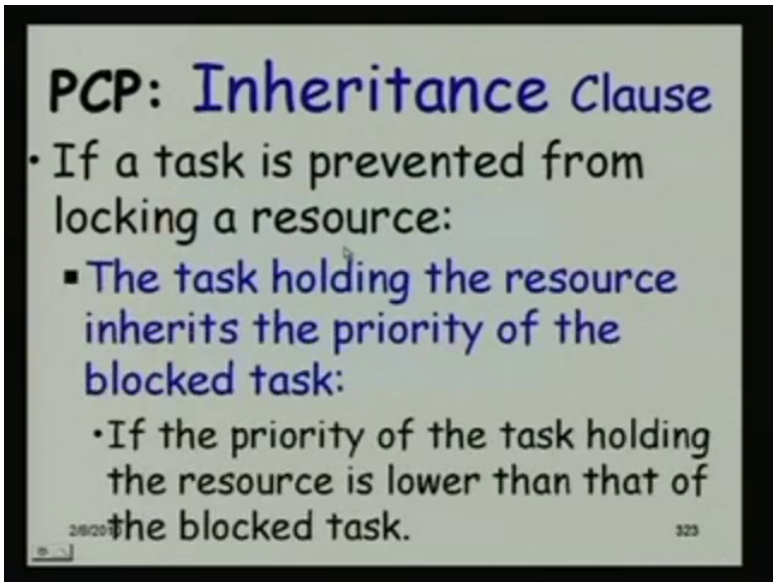
If a task is holding a resource with ceiling priority equals the CSC,then the task is granted access to the resources. Otherwise, T_i is not granted requested CR j, unless the priority of T_j it should have been T_i is not granted the resource. Unless, the priority of T_i is greater than CSC.

(Refer Slide Time: 44:51)



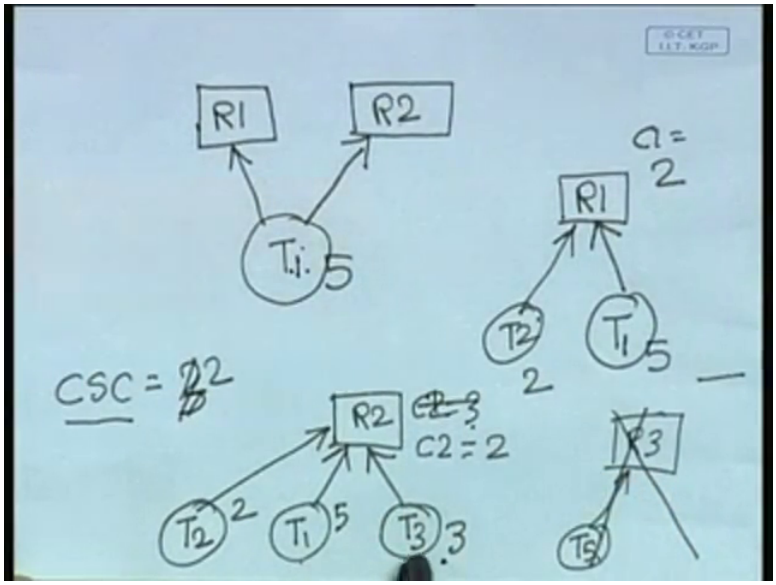
In both (a)and (b), if T i is granted access to theresource, then CSC is set to the ceiling CR j.As soon as there is a resource allocation,the ceiling values will the CSC will get updated. Because CSC will keep track of the highest ceiling value that is of the resource currently in use right. So, whenever we grant a resource we have to change the CSC if required.CSC can decrease by the course of time.

(Refer Slide Time: 45:44)



The second clause in the grant scheme is the inheritance clause.If a task is prevented from locking a resource, and then the inheritance clauseapplies. The task holding the resource will inherit the priority of the blocked task. For example:There are three tasks and two resources R1 and R2, and T 1requested R2, and CSC is set to 2.

(Refer Slide Time: 46:23)



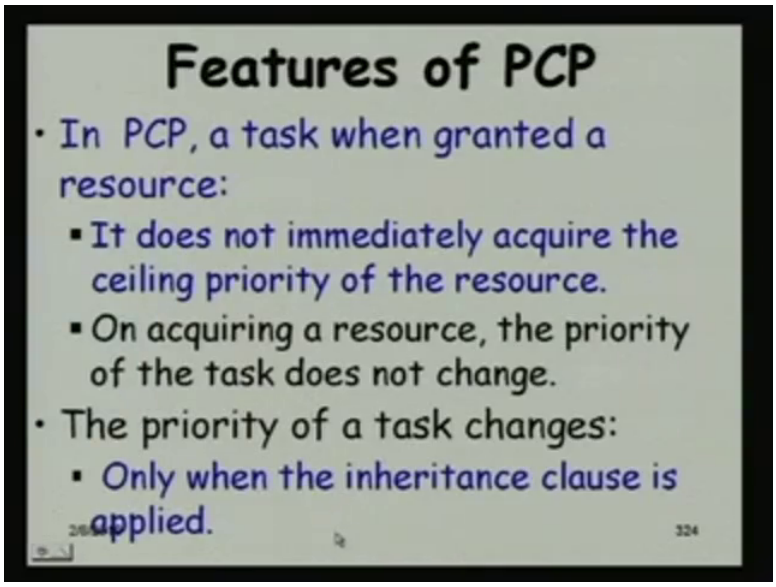
T_2 requests R1 at that time, then T_2 will get blockedbecause the resource is not available. Then the priority of T_1 will be raised to 2 by the inheritance clause. T_1 started executing, and its priority is 5 and then acquired the resource R2.Once it acquired the resource R2 its priority

did not change; it is still priority 5. The only thing that changed was the CSC value was set to 2. Now T_2, because its priority 2 it is started executing. As soon as it became ready started executing preempted T_1, the low priority task started executing, after sometime it needed R1.

Whenever a higher priority taskacquired a resource,its priority does not change only the CSC value changes. Whenever any other task is prevented from acquiring any resource and blocks, than the task that was holding the resource its priority increases by the inheritance clause.

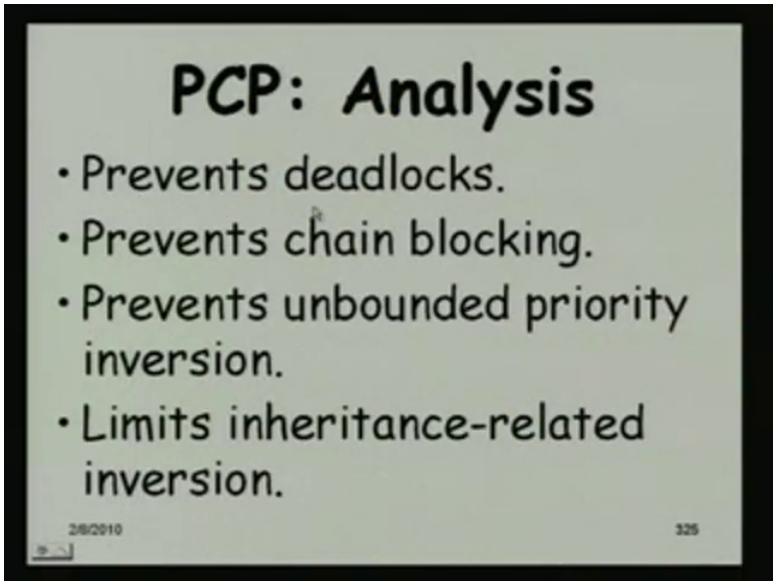
Inheritance clause applies only when the priority is the way the blocked task has a higher priority than the task that is executing. If it was a lower priority, then even this situation will not arise. It started executing and thenrequested resource that scenario will not exist at all. We are talking of only higher priority task requesting a resource.

(Refer Slide Time: 49:58)



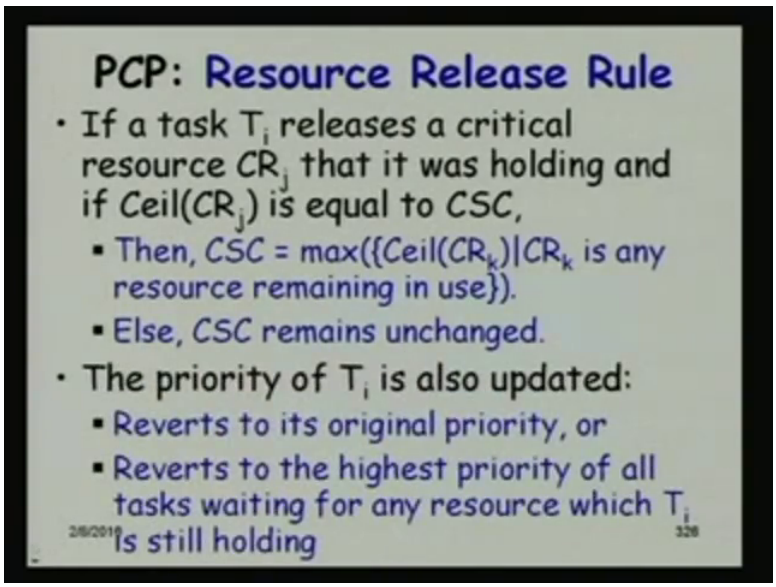
We have seen that a task when it gets a resource, its priority does not change. When a higher priority task waits for beingdenied a resource, then the inheritance clause is applied.

(Refer Slide Time: 50:43)



This simple scheme not as simple as highest locker slightly more complex, because we have a system variable which is set, and when a resource is requested there are two clauses one is request clause where it is checks against the ceiling priority. If it is the task which set had set the CSC then it isit is not checked against CSC.This will prevent the deadlock situation,and also prevent chain blocking, unbounded priority inversion reduce the inheritance related inversion to a large extent.

(Refer Slide Time: 52:56)



When aresource is released by a task which was holding the ceiling, then if this was the maximum of allceilings that are of the resources currently in use, then the CSC is changed to the onethat is current maximum. If it is not the maximum then nothing changes it just releases the resource.If an inheritanceclause was applied then that also needs to be reverts.

If itinheritance clause was applied,then it will revert to its original priority.If it was not holding any resource then, it will get the highest priority of all the blocked tasks on those resources.

The CSC value will get updated if applicable and it will revert to its old priority. We will just stop here,and meet for the next class.