

Real Time System
Professor Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture: 29
Priority Ceiling Protocol (PCP)

Welcome to this lecture, in the last few lectures, we were discussing about how critical resources can be shared among real time tasks. The traditional operating system of semaphores is not suitable for task sharing among real time tasks, if we use traditional semaphores then the tasks will suffer unbounded priority inversions and tasks can easily miss their deadlines. And we had so far looked at the priority inheritance protocol and we were discussing about the highest locker protocol we had seen the locker protocol highest locker protocol or HLP. Now, let us look at some issues in HLP.

(Refer Slide Time: 01:04)

Highest Locker Protocol (HLP)

- During the design of a system:
 - A ceiling priority value is assigned to each resource.
 - The ceiling priority is equal to the highest priority of all tasks needing to use that resource.
- When a task acquires a resource:
 - Its priority value is raised to the ceiling priority of that resource.

The highest locker protocol, we had said that each resource is assigned a priority if there are a set of resources in a system R_1, R_2, R_3 then each resource is assigned a ceiling priority value C_1, C_2, C_3 . The ceiling priority value is the maximum priority of the tasks going to use the resource. So, if T_1, T_2, T_3 are going to use this resource and the resource usage by tasks can be known from a code analysis.

A static analysis of the code can show which resource is being used by which tasks and the ceiling priority is assigned based on the tasks that use the resource the highest priority of all the

tasks that might use the resource is the ceiling priority of the tasks. So, similarly, C_2 may be used by T_5 , T_6 and C_3 maybe T_1 , T_5 and T_7 . So, for each task, the ceiling priority is computed and is initialized during the system and initialization.

And now, once the ceiling priority has been assigned during system initialization, as and when a task is granted a resource the task priority changes to the ceiling priority of the resource. So, if the task let us say T_3 here acquires C_1 , then immediately T_3 priority will be set equal to priority of the ceiling of the R_1 which is ceiling of C_1 or the priority of T_1 .

Because priority of T_1 that the T_1 is the highest priority task using R_1 and C_1 is set to priority of T_1 . So, T_3 will get the priority of T_1 until T_3 completes usage of resource R_1 , that is the protocol. So, here the tasks that are using resource R are T_{10} , T_2 and T_5 . And assuming T_2 is the highest priority task using R the ceiling of R is equal to 2. That is the priority of the highest priority tasks using R .

(Refer Slide Time: 04:20)

Highest Locker Protocol (HLP)

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

The slide includes a diagram showing two resource boxes labeled R_1 and R_2 . An arrow points from R_1 to R_2 . Below R_1 is the label T_5 with an arrow pointing to R_1 . Below R_2 is the label T_7 with an arrow pointing to R_2 . A small video inset of a man is visible in the bottom right corner of the slide.

Now, the protocol is simple. That tasks priority is increased to the ceiling priority of the task when it acquires the resource when it returns the resource after completing the usage it gets back its own priority. For this protocol to analyze, analyze this protocol lets state one theorem that when HLP is used for resource sharing among real time tasks.

Once a task gets any one of the resources required by it, it is not blocked any further. So, that means, if a task T_5 needs resource R_1 and R_2 and it is granted R_1 then later when it needs R_2 it will not lock that is guaranteed. How can we prove this? You can just outline a sketch of the proof not go to the exact proof; the proof is that since T_5 is also using R_2 the priority of the ceiling priority of R_2 is at least that of T_5 .

So, if the resource R_2 was acquired by another task before T_5 acquired R_1 . Let us say a task T_7 had acquired R_2 before T_5 had acquired then the blocking can occur but once T_7 acquires R_2 the system ceiling will be set to at least equal to T_5 then T_5 could not have acquired R_1 because T_7 priorities raised to T_5 , T_5 cannot execute it, cannot preempt T_7 to execute. As soon as T_7 occurs R_2 its priority becomes at least equal to T_5 .

Now, let us so, before T_5 acquired R_1 , T_7 acquiring R_2 is ruled out. Now, let us say if T_5 is using R_1 whether T_7 can acquire R_2 after T_5 has started using R_1 that cannot occur because T_7 's priorities T_5 less than T_5 and it could not have preempted T_5 to acquire R_2 . So, that question also does not arise. So, in the same way, we can argue that all the resources required by T_5 could not have been acquired by a lower priority task.

And therefore, once T_5 gets one resource for other resources it cannot block. And the corollary of this theorem is that when a task is granted one resource, all of the resources required by it must be free. We can use the same theorem to say that since it is not blocked any further and it could not have been acquired by a lower priority task. And it being used by a higher priority task, then T_5 cannot really block for it T_5 won't be executed cannot start executing.

And therefore, all resources required by T_5 must be free. Any task that is granted one resource at that time, all of the resources required by it must be free. So, that is a corollary. And another corollary, which again follows from the theorem that a task cannot undergo chain blocking. Chain blocking is not possible in HLP. Because a task is single blocking all resources required by a task is free before it acquires any one of its resource. So chain blocking cannot occur in highest locker protocol.

(Refer Slide Time: 09:06)

The slide is titled "Highest Locker Protocol (HLP)". It contains two main bullet points:

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

Handwritten annotations on the slide include a red bracket under "lock R1" in T1's sequence, a red box around "lock R2" in T2's sequence, and a red "X" next to the second bullet point. A small circular inset shows a man in a striped shirt speaking. The slide has a decorative blue and green geometric background on the right side and logos at the bottom.

We can also see that HLP prevents deadlock, we had seen that in the basic priority inheritance scheme deadlock was a reality, tasks cannot enter into deadlocks. Now let us take the same example we had considered for the priority inheritance protocol and so that deadlock cannot occur. Now let us assume let us me repeat the same scenario that T1 and T2 are two tasks and they need the resources R1 and R2, but in different order.

First, T2 needs R2 and then R1 and T1 needs R1 and then R2 and T2 is the lower priority task and T1 is the higher priority task. Now let us say, T1 starts executing, the lower priority task executes and locks R2 but according to the highest locker protocol, R2's priority becomes equal to the ceiling priority of R2. And since R1, T1 is also using R2.

Then the ceiling priority of R2 is at least that of R T1 and therefore, T2's priority immediately increases becomes at least equal to T1 and therefore, T1 cannot execute until T2 is executing, and it cannot execute the instruction lock R1 because T2's priority has already been increased to T1. And therefore, deadlock cannot occur. We are not giving a rigorous proof.

But we are just repeating the same example where we had shown occurrence of a deadlock in the inheritance scheme. And we are saying that the same example deadlock cannot occur, but it is possible to work out a rigorous proof saying that deadlock cannot take place in highest locker

protocol, but we will exclude this from our discussions and also it prevents priority inversion by the very nature of the highest locker protocol, the priority of the task increases. So, it cannot happen that a lower priority task executes while this task keeps on waiting.

(Refer Slide Time: 11:52)

Shortcomings of HLP: Inheritance Blocking

- 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 resource cannot execute and undergo priority inversion.

NR

The slide features a speaker in a circular inset on the right and logos at the bottom.

Shortcomings of HLP: Inheritance Blocking

- 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 resource cannot execute and undergo priority inversion.

T3, T7, T10 (Pri=2)

The slide features a speaker in a circular inset on the right and logos at the bottom.

So, we could see that the highest locker protocol prevents unbounded priority inversion prevents deadlock and prevents chain blocking. So far so good, but then let us see if there are any problems with the HLP. HLP has one very bad problem, the name of the problem is inheritance blocking. The problem is that as soon as a low priority task let us say T10 acquires resource which is also used by T1.

So, as soon as T10 acquires the resource its priority becomes T1, it starts executing at the priority of T1. Now, there may be several other tasks like T5, T7 etc. Who do not need this resource but they will be prevented from executing because we have raised the priority to very high value. So, those tasks which are higher priority than this task, low priority task and not needing the resource now cannot execute and will undergo priority inversion.

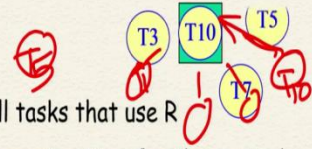
Let us take an example. Let us say a resource was there are and non-preemptable resource NR and then the task T10 acquired the resource. Now, the non-preemptable resource was being also, it can be used by T1. So, the priority of T10 will become exactly equal to that of T1 which is 1 so it will become the highest priority task. And let us assume that there were other tasks. Its priority let us say is 2 was the priority of the T2 was the highest priority tasks that can use NR.

Now, there may be other tasks like T7, T3, etc. these are higher priority than T10. And they do not need the resource. But still they cannot execute because T10 will start executing at a very high priority and T3, T7 cannot preempt T10. So, even though no process no task is waiting for the resource. It is only T10 which is executing the priority of T10 has become very high and therefore, several other tasks will undergo inversion and they may miss the deadline.

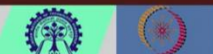

Because it may happen again, this T3, T7, etc. They may undergo inheritance blocking and therefore, they might miss their deadline.

(Refer Slide Time: 15:10)

HLP: Blocking Time of a Task



- Let:
 - $Use(R)$ is the set of all tasks that use R
 - $C(T_k, R)$ denote the computing time for the critical section for task T_k using resource R .
- Maximal blocking time B for task T_i not needing R :
$$B = \max\{C(T_k, R) \mid T_k \in Use(R), pr(k) < pr(i)\}$$





Now, let us compute the blocking time of a task under HLP. Let us say we have a resource R and we have a set of resources set of tasks that can use the resource and we also give a worst case requirement of the resource by different tasks. So, that is given by $C(T_k, R)$ indicates how much time the task T_k will need R for its execution. What is the maximum time that the task T_k can need R the resource R is given by $C(T_k, R)$?

Now, the maximum blocking time of a task that does not need the resource. So, let us say this was T_1 and this is T_{10} . So, as soon as T_{10} acquires the resource its priority becomes T_1 and T_5 will undergo inheritance blocking as soon as T_{10} acquires R , T_5 will have to wait. So, the maximal waiting for T_5 is the usage time of the resource R for all the tasks which are lower priority than the task T_i , T_5 and which are using the resource R . Assuming that they use it for once, an instance of a task uses the resource only once and then T_5 will undergo $\max C(T_k, R)$.

(Refer Slide Time: 17:22)

Usage of HLP

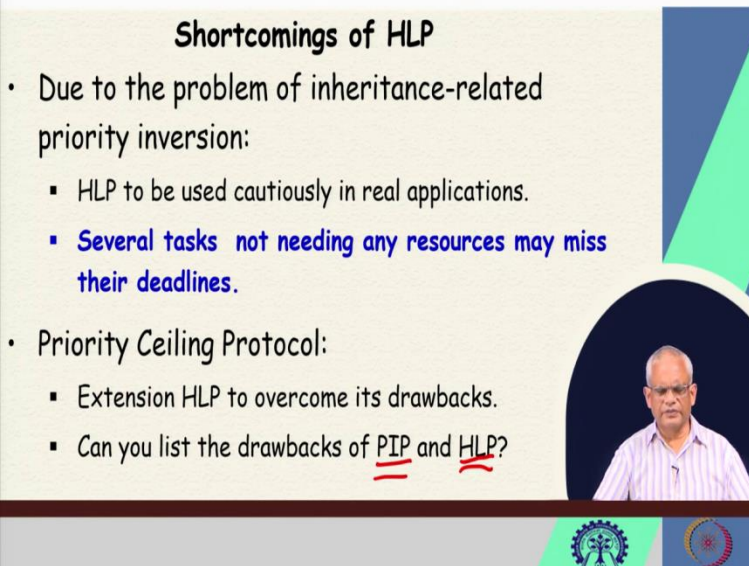
- POSIX supports priority locking for mutexes:
 - Example: `pthread_mutexattr_setprotocol(&attr, PTHREAD_PRIO_PROTECT);`
- Linux does not support HLP
- The Ada programming language supports HLP:
 - Calls it "ceiling priority locking"
- Real-time Specification for Java (RTSJ) supports HLP:
 - Calls this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



HLP is it used in practice yes. It is used in the POSIX standard. We can setprotocol to PTHREAD PRIO PROTECT which is the HLP. Linux does not support HLP but other programming language supports HLP and calls it the ceiling priority blocking and the real time specification for Java RTSJ also supports HLP and calls this the priority ceiling emulation.

The main advantages of HLP is that it is very simple and it prevents the deadlock, it prevents chain blocking prevents unbounded priority inversion, but the only negative aspect of the HLP is the inheritance blocking; different tasks not needing the resource can undergo inheritance blocking.

(Refer Slide Time: 18:23)



Shortcomings of HLP

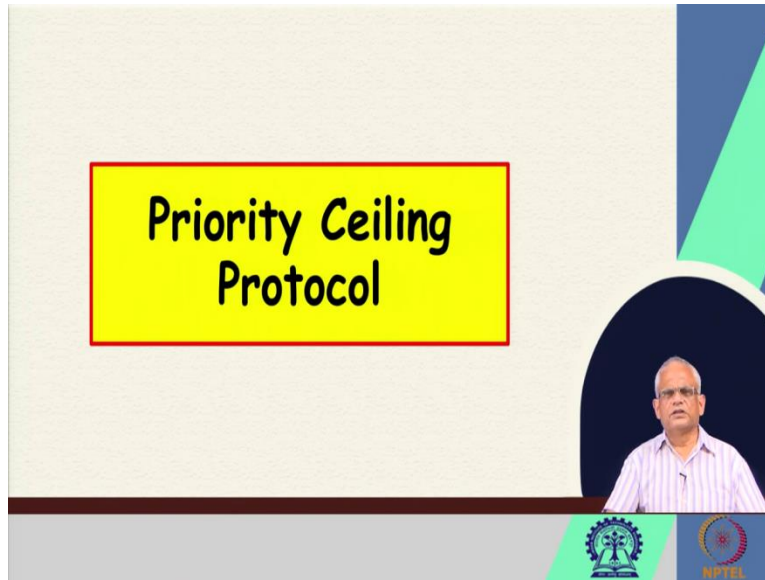
- Due to the problem of inheritance-related priority inversion:
 - HLP to be used cautiously in real applications.
 - **Several tasks not needing any resources may miss their deadlines.**
- Priority Ceiling Protocol:
 - Extension HLP to overcome its drawbacks.
 - Can you list the drawbacks of PIP and HLP?

The slide features a video inset of a man in a striped shirt speaking. The background is light green with a blue and green geometric design on the right side. At the bottom, there are logos for a university and a research center.

And because the tasks not needing resource can undergo inheritance inversion for long time HLP when it is used, we have to take abundance precaution before we use HLP otherwise tasks may not needing the resource miss their deadlines. The inheritance blocking is the major problem of HLP and the priority ceiling protocol was devised to overcome this drawback of HLP.

HLP had this inheritance blocking problem and basic priority inheritance scheme had several problems. We had seen that and the priority ceiling protocol overcomes all those problems. So, the HLP solves the problems of PIP, but introduces inheritance blocking problem and the priority inheritance protocol, had several problems I had seen that just try to recollect. What are the major problems of the basic priority inheritance principle?

(Refer Slide Time: 19:51)



Now let us look at the priority ceiling protocol. This protocol is definitely superior to both simple priority inheritance protocol and the highest locker protocol. But it is slightly more complicated and needs extra support from the operating system. If the system is very simple, you can use a basic priority inheritance principle extremely small system.


Otherwise, for simpler systems, you can use the highest locker protocol. But if the application is bit sophisticated, we need to use the ceiling protocol.

(Refer Slide Time: 20:41)

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
 - Called Current System Ceiling (CSC).

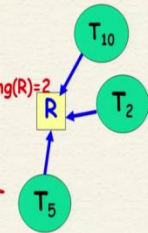

PATH
CSC



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
 - Called Current System Ceiling (CSC).

CSC → 100
Ceiling(R)=2
CSC = 2

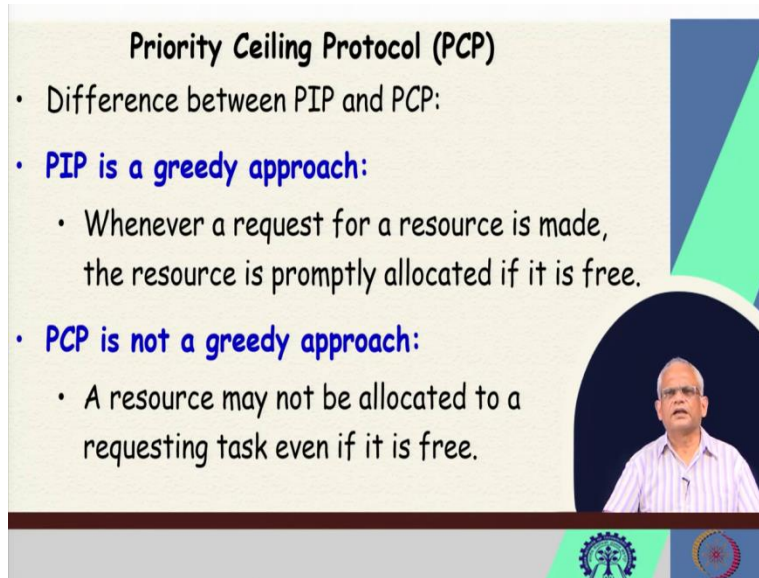
Now, let us try to understand the ceiling protocol. As I said that it is slightly more complex than the basic priority inheritance scheme and the highest locker scheme. Here just like HLP we assign a ceiling priority to all resources every resource is assigned a ceiling priority and ceiling computation is the same as the HLP.

The ceiling priority of a resource is equal to the highest priority task that usage this resource, and here we need a support for the operating system, we need a operating system variable which denotes the current system ceiling that is the highest ceiling among all the locked resources. So, we need a operating system variable. Every operating system maintains several environment variables. For example, we might have a PATH variable.

The PATH variable can be examined by different applications and by the operating system to know where a specific library exists. Similarly, in the set of environment variables, we need a variable called CSC and the CSC will indicate at present among all resources being used, what is the highest resource ceiling?

Ceiling value assigned to the resources. So, if this is the task R, then we have a ceiling computed and as soon as, let us say T₂ acquires R, then the CSC will be set to 2. To start during the system initialization, CSC will be set to the highest priority, maybe 100 or something, very low priority. Now, as soon as it acquires, the resource R the CSC will be set to 2.

(Refer Slide Time: 23:27)



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.

Now, before we look the details of the protocol, let us look at one difference between PIP and PCP that will give us an insight into the working of the ceiling protocol. We can say that the basic priority inheritance protocol is a greedy approach in the sense that if a task requests resource and the resource is free, then it obviously gets that resource, nothing prevents it from getting the resource. But in the PCP, the priority ceiling protocol we see that sometimes the resource will be available.

But a task requesting that resource, the free resource, it may not be granted and we call it as avoidance. We will look at the details of the protocol where when a resource is even free. We do not grant it and therefore it is not a greedy approach. Sometimes we just do not allocate resources just because it is free. Anticipating problems for the future, we do not grant the resource. We will see when we discuss the details of the protocol.

(Refer Slide Time: 25:07)

PCP: CSC

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

The current system ceiling is set to the maximum resource ceiling, ceiling of the resources that are currently being used. Let us say R1 and R2 are currently being used. And R5, R3, etc. They are not being used. So, between these two, R1 and R2, let us say the ceiling value of this is 5, and C2 the ceiling value is 2. So, if these two resources are being used, the system ceiling CSC will be set to equal to 5.

That is the protocol that the CSC at any time is equal to the maximum ceiling of all the resources being used. So, that is about how the system ceiling is set. But the actual protocol is described using two rules. One is called as the resource request rule, and the other is called as the resource release rule. And these two, they make use of the system ceiling value.

And there are several clauses in this rule. As are saying that this rule is slightly more complicated than the simple inheritance scheme, or the highest locker protocol. We are at the end of this lecture. We will stop here. And in the next lecture, we will look at the details of the priority ceiling protocol, and namely the resource request rule and the resource release rule with some examples.

And then we will analyze the impact of using the priority ceiling protocol on the schedulability of the tasks, or the blocking time of tasks. So, first, we will compute the blocking time of tasks

and then we will see. What is its impact on the schedulability of set of tasks being scheduled in the rate monotonic scheduler? So, we will stop here. Thank you.