

Real Time Operating System
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 15
Highest Locker Protocol

Welcome, to this lecture over the last few lectures we have been looking at the Real Time Schedulers, initially started with the simple task schedulers, which are the clock driven schedulers. Then looked at the event driven schedulers and we identified two important schedulers the red monotonic scheduler and the earliest fast dead line scheduler, and between these two, we said that red monotonic scheduler is overwhelmingly popular, being used in many, many applications and also directly supported by the Real Time Operating Systems.

We start with we simplified the problem, we considered the task state is independent and then slowly we were trying to make it applicable to realistic applications and in the last class, we are looking at the problems that arise when tasks share resources, non-preemptible resources. And we have identified that the traditional operating system of sigma 4 does not work well. The problems that are faced are priority inversion and unbounded priority inversion, priority inversion as it is can be solved by careful programming by reducing the time per which a low priority task uses its critical section.

But unbounded priority inversion is a very, very severe problem and many applications in the past half field for not properly addressing the unbounded priority inversion problem. We had explained, what exactly is the problem and then we are looking at the solutions. The simplest solution is the priority inheritance protocol, where a low priority task inherits the priority of higher priority task waiting for the resource. We saw in the last lecture that this protocol the simple, priority inheritance protocol does overcome the unbounded priority inversion problem.

But then it introduces some problem and also does not address well some other problems. Now let us first identify these disadvantages of the basic priority inheritance protocol then we will look at more sophisticated protocols.

(Refer Slide Time: 03:28)

Shortcomings of the Basic Priority Inheritance Scheme

- PIP suffers from two important drawbacks:
 - Susceptible to chain blocking.
 - Does nothing to prevent deadlocks.

200

So, let us look at the short comings of the basic priority inheritance scheme, there are two important drawbacks of the priority inheritance protocol. The first one, is a chain blocking, we will see what exactly is chain blocking and then the second is it does nothing to prevent deadlocks.

So, when we use the basic priority inheritance scheme, the tasks may get deadlocked, but we will see that more sophisticated algorithms, which are basically improvements over the basic priority inheritance protocol they overcome these problems.

(Refer Slide Time: 04:19)

Deadlocks

- Consider two tasks T_1 and T_2 accessing critical resources CR_1 and CR_2 .
- Assume:
 - T_1 has a higher priority than T_2
 - T_2 starts running first
- T_1 : Lock R_1 , ... Lock R_2 , ...Unlock R_2 , Unlock R_1
- T_2 : Lock R_2 , ... Lock R_1 , ...Unlock R_1 , Unlock R_2

201

First let us, try to see how deadlocks occur when the priority inheritance protocol is used. Let us assume that there are two real time tasks T 1 and T 2 and these both of these tasks need two resources. T 1 needs both CR1 and CR2 and T 2 also needs both CR and CR1 and CR2, and let us assumes that by the rate monotonic scheme T 1 has higher priority than T 2.

But then, T 2 starts running fast because there are no jobs for T 1. Because, T 1 phasing is later than T 2 or may be the T ones instance has just completed and T 2 is starting to use. Now T 2 runs first and then these are the sequence of operations, they undertaker T 2 it executes the instruction lock R 2, and since the critical resource R 2 is available it can successfully lock R 2 and then it does some extra processing some other processing it does by that time the task T 1 instance arrives being a higher priority it preempts T 2.

But, then after some local processing it locks R 1 and then does more processing using R 1, and then needs the resources R 2. And when it locks R 2, R 2 has already been locked by T 1 and therefore, sorry R 2 has already been locked by T 2 and therefore, T 1 blocks and then T 2 gets the control T 2 starts executing, it does some executions using R 2, but then it needs R 1 and as its tries to lock R 1 R one is already locked by T 1 and that is the deadlock situation.

So, using the basic priority inheritance scheme, deadlocks can arise of course, you can see here that once T 2 sorry T 1 locks on T 2, T 2 priority increases to T 1, but that does not help still, the deadlock arises.

(Refer Slide Time: 07:31)

Chain Blocking

- A task needing to use a set of resources undergoes chain blocking, if :
 - Each time it needs a resource, it undergoes priority inversion.
- Example:
 - Assume a high-priority task T_1 needs several resources

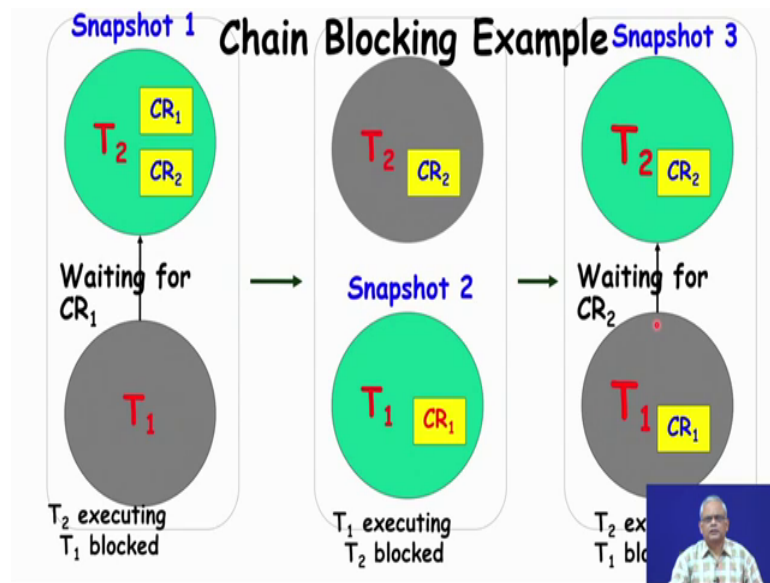
202

Now, let us look at the second problem that the basic priority inheritance scheme suffers from goes by the name chain blocking. First let us see what do you mean by chain blocking, the chain blocking arises when a task needs a number of resources and.

The chain blocking causes each time the task needs a resource different resource it undergoes priority inversion, that is if a task needs n resources, each time it requests for one of the resources. It undergoes inversion waits for the resource and by that time lower priority tasks execute and multiple priority inversions occur and in the chain blocking situation, when a task needs multiple resources the waiting time all together can be very high and high priority task can easily miss the deadline.

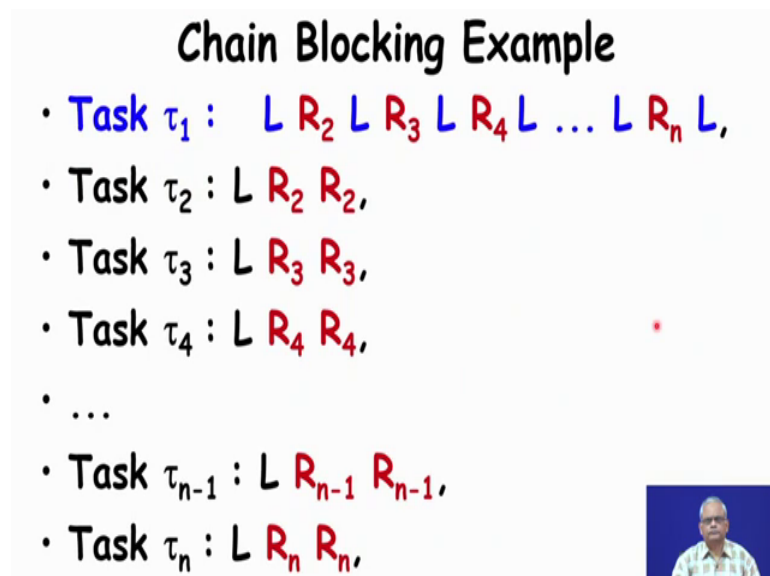
So, chain blocking is a severe problem, now let us throw a schematic diagram. Let us, try to understand what exactly is the chain blocking, let us assume that the highest priority task T_1 needs several resources, let us say it needs two resources.

(Refer Slide Time: 09:10)



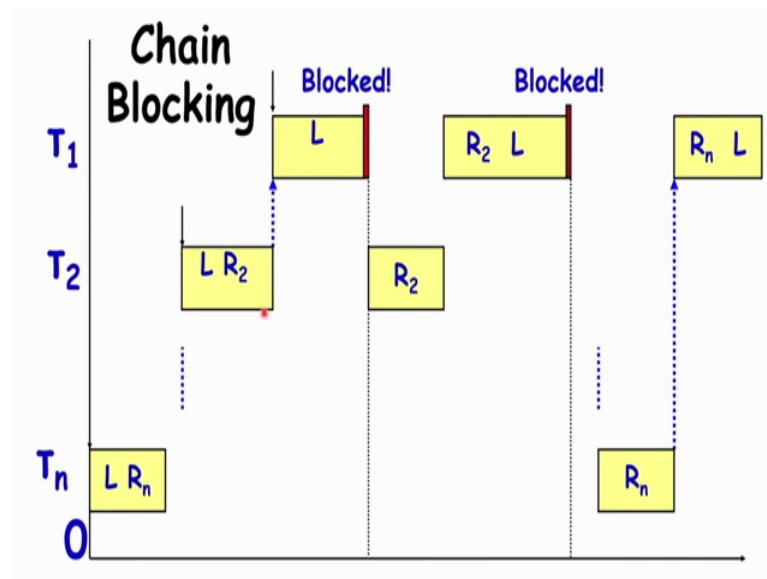
So, task T_1 , initially blocks for CR_1 because T_2 is holding CR_1 and CR_2 . After sometime, T_2 releases CR_1 and T_1 starts executing, but then after sometime it needs CR_2 and again it blocks for CR_2 , but in the process it undergoes priority inversion. So, a high priority task under the priority inheritance scheme can undergo multiple priority inversion. If it uses multiple non pre-emptible resources, but we will see that more sophisticated algorithms to overcome the chain blocking problem.

(Refer Slide Time: 10:13)



This is another more generalized example, let us assume that task T_1 is a high priority task. It does some local processing without using resource then it needs the resource R_2 and then it does further processing, using R_2 then it needs R_3 , then it needs R_4 and. So, on now let us assume that there are several low priority tasks, which need this resources for example, T_2 needs the resource R_2 it does some processing with R_2 , T_3 does processing with R_3 , T_4 with R_4 , T_{n-1} with R_{n-1} and T_n needs the resource R_n .

(Refer Slide Time: 11:10)



Now, Let us assume that initially, the task T_n which is the lowest priority starts executing, and it locks R_n . Similarly, all other tasks they lock their resources and at that instant the task T_1 arises, an instance of T_1 arises. It does some processing initially, but then it needs the resource R_n , gets blocked sorry, it needs the resource R_2 and R_2 is already blocked by is already locked by T_2 and therefore, T_1 gets blocked.

Now, T_2 starts executing, because T_1 has blocked CPU becomes available and it does local processing and releases R_2 , and then T_1 gets R_2 starts processing and needs T_3 sorry, R_3 which is held by T_3 and so on. So, for all the n resources it undergoes priority inversion one after the other.

What is the maximum time that it gets blocked, the maximum time the task T_1 gets blocked is equal to the time per for each of these holds is a sum of the time per, the duration per which for each of the task held holds their respective resource for example,

for this situation. The total blocking time for the task T_1 , due to the n resources will be the time per which T_2 will use the resource R_2 , T_3 uses the resources R_3 and so on. And T_n needs the resource R_n , which can be large.

So, chain blocking is a serious problem in the simple priority inheritance scheme. The simple priority inheritance scheme as we will see requires very minimal support from operating system and therefore, in simple applications it is actually used, but then the programmer the designer of the application must keep in mind, these two serious problems that they might face the deadlock problem and the chain blocking problem.

(Refer Slide Time: 13:57)

Properties of PIP

- **Theorem 1:**
 - If a task T_a can be blocked by k lower priority tasks $T_1 \dots T_k$ due to a single resource usage,
 - Then the worst case duration for which it can block is $\max(e_i)$; where e_i is the critical section time of task T_i .
- **Theorem 2:**
 - If a task needs to use k critical resources:
 - The maximum duration for which it can block is $\sum \max(e_j)$ for each critical resource.
 - e_i is the longest execution duration by a task for resource r_i

Now, let us look at some properties of the simple priority inheritance protocol, let us say a task T_a needs a resource, which is used by k other lower priority tasks let me repeat, that a task T_a needs a resource non pre-emptible resource which is also used by k lower priority tasks. Then under the priority inheritance scheme what is the maximum duration for which the task T_a may suffer priority inversion, the answer is that it is maximum of the execution time of the k lower priority tasks for which they need the resource the non pre-emptible resource.

If e_i is the time for which the task T_i needs the resource, then the worst case blocking time for the task T_a is maximum of e_i considering all the tasks T_i . The second theorem is that if a task needs k resources, the first one we had looked at one resource needed by a task here a task needs k resources and each of these k resources is used by several of

the lower priority tasks. Now what will be the total waiting time for the task T_a in the worst case as you can see that this is a generalization of the first theorem, and also we know.

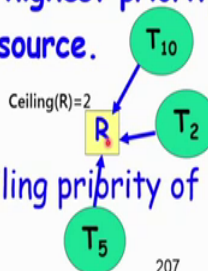
That the priority inheritance scheme suffers from chain blocking. So, for each of the resources it requests it may undergo blocking. So, it will be sum of the maximum of e_i for each of the resources.

If there are three resources and these three resources are used by different sets of lower priority tasks then the total blocking time for the high priority task. In the worst case, will be the blocking time for the first resource, plus the blocking time of the second resource, plus the blocking time of the third resource, where the blocking time for each resource is given by theorem 1.

(Refer Slide Time: 17:00)

Highest Locker Protocol

- During the design of a system
 - A ceiling priority value is assigned to all resources.
 - 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.



207

Now, let us look at some improvements to the basic priority inheritance scheme and we have seen that the priority inheritance scheme is a simple, but powerful protocol it does overcome the unbounded priority inversion problem, which is a severe problem can cause tasks to miss their deadline, but then we saw that it is rather two simple and suffers from two major problems the deadlock problem and chain blocking problem.

Now let us look at the first improvement which is done by the highest locker protocol, the highest locker protocol there is also a priority inheritance scheme, but it is slightly

different here for each resource that is under used by a task a ceiling priority value is assigned. The ceiling value is computed as the highest priority task that can use that resource. So, let me repeat that the main idea in the highest locker protocol is that a priority value is assigned with to a resource. It is called as a ceiling value, for that resource which equals to the highest priority task that may ever use that resource.

Now, let us look at an example if R is a resource and is used by 3 tasks T 2, T 5 and T 10 and let us assume, that T ten sorry let us assume that T 2 is the highest priority task. Then for the resource R, there is a ceiling value assigned which is equal to 2. And the highest protocol requires that when a task requests a resource, then the priority of that task is raised to the ceiling value for example, if the resources R is available and let us say the task T 10, gets the resource R, requests and gets the resource R.

Then T 10s priority value will be raise to 2, in the highest locker protocol each resource is associated with a ceiling value and as soon as any task locks to the resource the priority of that task becomes equal to the ceiling value.

(Refer Slide Time: 20:03)

Highest Locker Protocol (HLP)

- Addresses the shortcomings of PIP:
 - However, introduces new complications.
 - Addressed by Priority Ceiling Protocol (PCP).
 - Easier to first understand working of HLP and then PCP.
- During the design of a system:
 - A ceiling priority value is assigned to all critical resources.
 - The ceiling priority is equal to the highest priority of all tasks using that resource.

208

We can easily see that this protocol overcomes, the shortcomings of the basic priority inheritance scheme, but then it introduces a very difficult complication. We will see what is the complication a little bit while from now, but then we will also look at an improvement to the highest locker protocol which is called the priority ceiling protocol,

that overcomes the not overcomes to large extent overcomes the problem that highest locker protocol introduces.

The highest locker protocol is small improvement of the basic priority inheritance scheme, and if we understand the highest locker protocol then it becomes easy to understand the ceiling protocol. The priority ceiling protocol the priority ceiling protocol by itself is little more complicated than both the priority inheritance scheme and the highest locker protocol, and if we know the two protocols the priority inheritance protocol and the highest locker protocol then it will become very easy to understand the working of the priority ceiling protocol.

But then the priority ceiling protocol is the best protocol even though it is slightly more complicated, but it overcomes largely overcomes all the problems of the basic priority inheritance scheme and the highest locker protocol. In the highest locker protocol, during the design of an application what are the tasks and what resources they need is analyzed and then. A ceiling value is assigned to all critical resources and whenever a task acquires a resource a critical resource its priority becomes equal to the ceiling. And that is the simple protocol, let me repeat again that during the design of the system all the critical resources are assigned a ceiling value.

The ceiling value is equal to the highest priority task that may ever use that resource and each time a task a lower priority task or any other task acquires the resource its priority increases to the ceiling value, that is the inheritance scheme here that assumes as a task acquires the resource, its priority value increase becomes equal to the ceiling value associated with the resource.

(Refer Slide Time: 23:10)

Ceiling Priority of a Resource

- When a task acquires a resource:
 - Its priority value is raised to the ceiling priority of that resource.

$Ceil(R) = \max\text{-prio}(T1, T2, T3)$

Let us look at an example, let us assume that there is a critical resource R and the resource R is being used by 3 tasks, T 1, T 2 and T 3 and there will be a ceiling value associated with the R which is equal to the maximum of the priorities of T 1, T 2 and T 3.

(Refer Slide Time: 23:39)

Example

$Ceil(R) = \max\text{-prio}(T1, T2, T3) = 2$

Let us see, schematically how does the protocol work we have these let us assume that T 1s priority is 5, T 2s priority is 2 and T 3s priority is 8, now what will be the ceiling value that will be associated with R. Let us assume that 2 is the highest priority and 8 is the lowest priority, in that case the ceiling value associated with the resource R is equal to 2,

because 2 is the highest of 2, 5 and 8. So, the ceiling value that will be associated with R is 2.

(Refer Slide Time: 24:25)

Highest Locker Protocol (HLP)

- If higher priority values indicate higher priority (e.g., **Microsoft Windows**):

$$Ceil(R_i) = \max(\{pri(T_j) \mid T_j \text{ needs } R_i\})$$

- If higher priority values indicate lower priority (e.g., **Unix**):

$$Ceil(R_i) = \min(\{pri(T_j) \mid T_j \text{ needs } R_i\})$$

211

But, one thing I need to clarify is that sometimes saying the 2 is the highest priority and some other time we are using 8 as the highest priority. Actually the confusion arises because different operating systems, they use different conventions for example, in Microsoft windows and some other operating systems the priority value is the high, higher the priority, the higher is the priority value.

So, T 10 will have higher priority than T 5 or T 1. So, the higher the priority value the higher is the priority, T 10 is higher priority than T 5 or T 1 whereas, in Unix and the Unix derivative operating systems. The exactly reverse is the conversion a lower priority indicates higher priority, a lower priority value indicates higher priority and higher priority value indicates lower priority. So, if a task is used by a 3 sorry, a resource is used by 3 tasks T 2, T 5 and T 8 then.

The ceiling value will become 2 because, 2 is the highest priority whereas, in the operating system, such as windows based operating systems. It will become 8 and that is why we have been using this two different conventions.

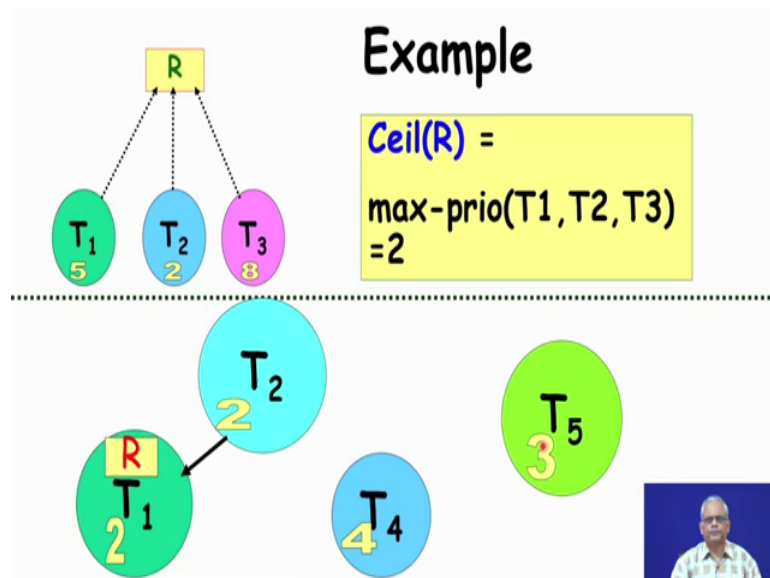
(Refer Slide Time: 26:16)

Highest Locker Protocol (HLP)

- As soon as a task acquires a resource R:
 - Its priority is raised to $Ceil(R)$
 - Helps eliminate the problems of:
 - Unbounded priority inversions,
 - Deadlock, and
 - Chain blocking.
- However, introduces inheritance blocking₂₁₂

The protocol is simple once, the ceiling values are associated with resources each time a task acquires a resource. It inherits the ceiling priority and then as it relinquishes the resource it completes processing with the resource and releases the resource then it gets back its earlier priority, this protocol helps to overcome the problems of unbounded priority inversion, deadlock and chain blocking, but then it creates a new problem which is called as the inheritance blocking.

(Refer Slide Time: 27:01)



Let us look at an example, let us assume that the resource R is a critical resource used by 3 tasks T 1, T 2 and T 3. T 1 with priority 5, T 2 with 2 and T 3 with 8 and the ceiling value associated with the resource is equal to 2, and the resource was available and acquired by T 1 whose priority is 5 and T 2 blocks on the task T 1, T 1s priority becomes 2 and then the tasks T 4, T 5 etcetera. These are higher priority than T 1, but then since T 1 priority is increased they cannot really get the CPU, because T 1 priority has been increased and T 4 T 5 etcetera.

Undergo inversion and that we call as inheritance related inversion, T 1 as soon as it requires the resource its priority increases to 2 and the other tasks needing the which are ready, but higher priority than T 1 cannot get CPU and they undergo inversion, we call it as the inheritance related inversion. Now we are the end of the lecture we will stop at this point and will continue in the next lecture.

Thank you.