

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

Lecture - 17
PCP Priority Inversions

Welcome to this lecture, in the last lecture and even before that lecture we have been looking at an important resource sharing protocol, the name of the protocol is PCP, the priority ceiling protocol

(Refer Slide Time: 00:34)

PCP: An Analysis

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


We had seen the protocol itself; we had seen the consequences of the protocol and this lecture we will do some analysis of the protocol to see, using the protocol whether a task set in the rate monotonic scheduling will remain schedulable, because the initial results of the rate monotonic scheduler considered independent tasks

Now, that the tasks share resources will have to see whether the task set remain schedulable using the priority ceiling protocol. Let us look at the priority ceiling protocol analysis, we had seen that it prevents deadlocks, it prevents chain blocking these are the major problem of the simple priority inheritance scheme, it prevents the unbounded priority inversion which is the problem that arises if we use a simple resource sharing scheme such as semaphore and also it limits inheritance related inversions.

(Refer Slide Time: 02:05)

Types of Priority Inversions in PCP

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



Now, let us see what are the inversions that can occur in the priority ceiling protocol and based on that we can determine the maximum time for which a task can block due to low priority tasks and that will give us a handle to check, whether a task set would remain schedulable under resource sharing.

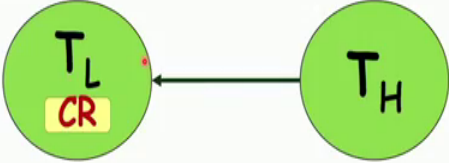
There are 3 main types of inversions in the priority ceiling protocol. The direct inversion, where a higher priority task waits for lower priority task just because a lower priority task is holding a non pre-emptable resource, inheritance related inversion, where a higher priority task needs to wait for a lower priority task just because the lower priority tasks priority has been enhanced due to the inheritance clause.

The avoidance related inversion; here even if a resource is available a task cannot use that resource because it is prevented by the priority ceiling protocol because when a task requests for resource the priority of the task is checked against the current system ceiling and if the priority of the tasks is less than the current system ceiling, less than or equal to then it is not allowed to access the resource and this is the scheme that is used for deadlock prevention because if it requests a resource and it is just granted it can cause deadlock. Now, let us see what are the durations for which the different inversion can occur and the tasks due to which the different inversion can occur? Let us do 1 example problem, then I will give you an exercise problem because these are important design problems in real time system development.

(Refer Slide Time: 04:33)

Direct Inversion

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



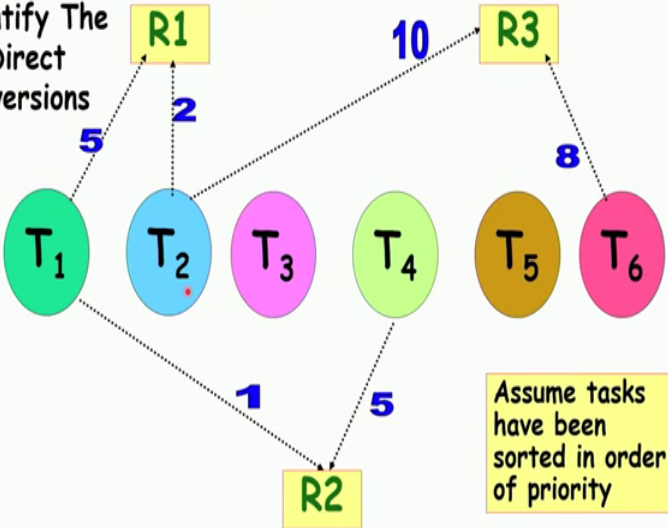
The diagram shows two green circles representing tasks. The left circle is labeled T_L and contains a yellow box labeled 'CR'. The right circle is labeled T_H . A solid black arrow points from T_H to T_L , indicating that T_H is waiting for T_L to finish using the resource CR.

235

First please look carefully, the direct inversion this occurs when a lower priority task T_L is holding a non pre-emptible resource mention CR here and the high priority task also needs the resource and therefore, it just waits for the low priority task to complete it is uses of the resource and release it.

(Refer Slide Time: 05:11)

Identify The Direct Inversions



The diagram shows six tasks T_1 through T_6 and three resources $R1$, $R2$, and $R3$. Tasks are represented by colored circles and resources by yellow boxes. Dotted lines with numbers indicate resource usage durations: T_1 uses $R1$ for 5 units, T_2 uses $R1$ for 2 units, T_2 uses $R3$ for 10 units, T_4 uses $R2$ for 5 units, T_5 uses $R2$ for 1 unit, and T_6 uses $R3$ for 8 units. A yellow box at the bottom right states: "Assume tasks have been sorted in order of priority".

236

Now, let us see in my example problem whether you can identify the direct inversions that can occur for a task and what is the duration of that inversion. Please look at these tasks set, we have 6 tasks and the resource uses indicated here that 3 resources $R1$, $R2$

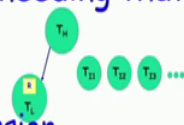
and R3; R1 is used by for 2 units by T2 and 5 units by T1; R2 is used for 1 unit by T1 and 5 units by T4 and R3 is used by for 10 units by T2 and 8 units by T 6. Now, please identify the direct inversions that can occur in this application, where there are 6 tasks in the resource is as given. Please remember that, it is the lower priority task that causes the inversion to the higher priority task.

The low priority task does not face inversion due to high priority task; it is a high priority task which faces inversion due to a low priority task. So, T1 will face direct inversion on account of R1 for 2 units, when T2 has already been using the resource R1. Similarly, T2 will suffer inversion for 8 units on account of T6 using R3, also T1 can face inversion on account of T4 using R2 and that will be for 5 units in the worst case.

(Refer Slide Time: 07:22)

Inheritance-Related Inversion

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



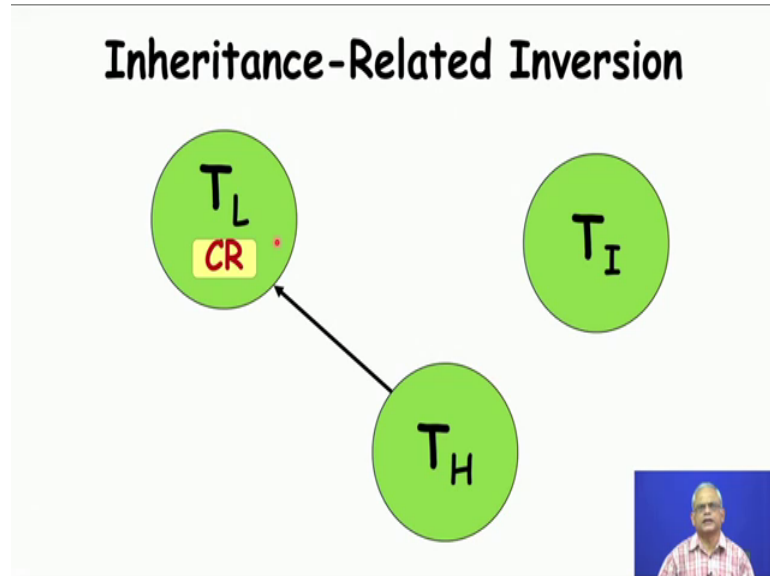
237

Now, let us look at the other type of inversion, the inheritance related inversions occurs when a lower priority task is using a resource and a high priority task waits for that resource in that case the priority of the low priority task is enhanced to be equal to the priority of the high priority task or in other words the low priority task inherits the priority of the high priority task that is waiting for the resource.

But then, there are intermediate priority tasks, which do not need the resource, they just want to use the CPU, but they are prevented from using the CPU, because the priority of the high priority task has been enhanced due to the inheritance clause in the priority ceiling protocol and this is the explanation for the inheritance related inversion. Now,

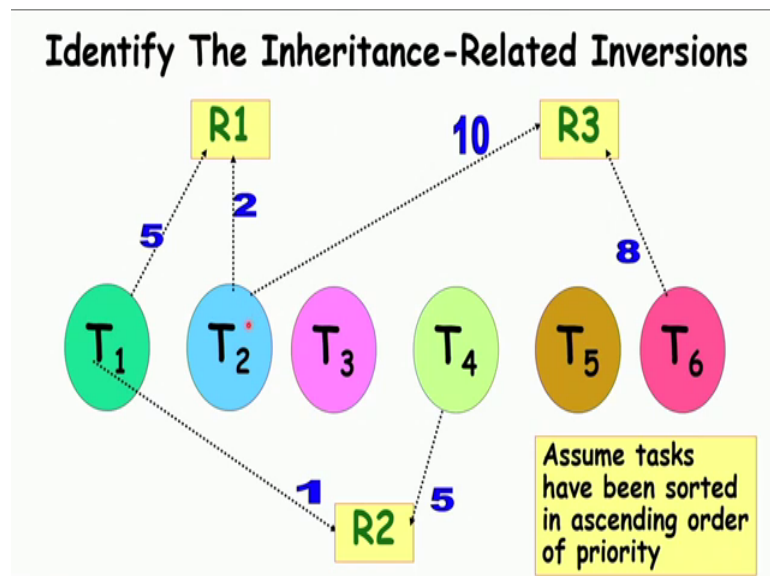
with this explanation let us look at a problem and identify what are the inheritance related inversion that can occur and for how long?

(Refer Slide Time: 08:47)



So, this is an example of inversion, here a low priority task is holding the resource, a high priority task is waiting for it and therefore, the priority of the low priority task is enhanced to be equal to the priority of the high priority task and in the mean while, tasks which are of intermediate priority they cannot use the CPU because the priority of the low priority task has been enhanced, so this is an inheritance related inversion.

(Refer Slide Time: 09:33)



Now, let us look at these problem, in a certain application we have 6 tasks, the tasks priorities are in descending order. So, that is T1 is the maximum priority, T2 is the second maximum priority, T3 is the third maximum and T6 is the lowest priority here. There are 3 resources R1, R2 and R3 and the tasks that use the resource for the certain time is indicated by the number along the arrow, T1 uses R1 for 5 units, T2 needs to use R1 for 2 units and so on.

Now, please identify the inheritance related inversions that can occur in this application. When T2 is using R1 and T1 is blocking for the resource R1, T2's priority gets enhanced to be equal to T1 by the inheritance clause, but will there be any inheritance related inversions will any of the task T3, T4, T5, T6 suffer in a inheritance related inversion, answer is no, because T2 is already the higher priority compared to T3, T4, T5, T6.


So, it does not cause any inversion to the low priority tasks. So, the resource used here by T1 and T2 that do not cause any inheritance related inversion to the other tasks because these are already a high priority. Now, let us look at the resource R2, when T4 is using the resource R2 and T1 is waiting, T4 priority gets increased to that of the T1. So, which are the tasks that would undergo inheritance related inversion, these are the tasks T2 and T3 because T4 priority is increased to that of T1, T2 and T3 will be prevented from using the CPU.

But, for what duration T2 and T3 can at most suffer inheritance related inversion for 5 units on account of T4 using R2. Now, what about R3, when T6 is using R3 and T2 is waiting T6 priority gets enhanced to that of the T2. So, the tasks which are of higher priority than T6, but less than T2 is also offers inheritance related inversion T3 T4 and T5, but for how long that is 8 because T6 can use at most 8 units and these 3 tasks will suffer inheritance related inversion for at most 8 units.

(Refer Slide Time: 13:21)

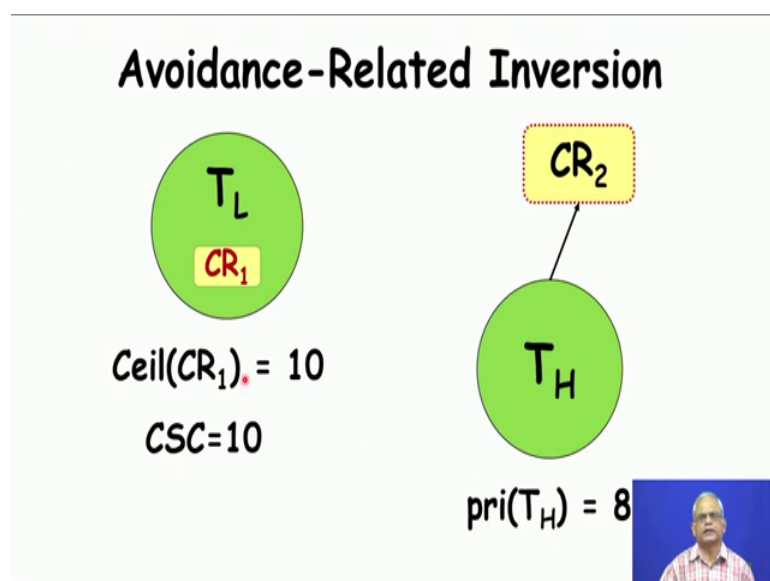
Avoidance-Related Inversion

- Consider a low priority task that is holding a resource:
 - CSC is made equal to the ceiling of the resource being held.
 - A higher priority task, whose priority is lower than the CSC, needs a resource currently not in use:
 - Undergoes avoidance-related inversion
 - Due to the resource grant rule
- Also called priority ceiling-related or dead avoidance inversion.



Now, let us look at the avoidance related inversion, there is a third type of inversion and an avoidance related inversion is also called dead lock avoidance inversion. Here, when a task requests a resource its priority is compared to the current system ceiling and only if its priority is more than a current system ceiling or it is the task that sets the current system till it is granted a resource, otherwise even if the resource is unused still it will not be allowed access to the resource and we say that it undergoes avoidance related inversions, this is also called the deadlock avoidance inversions or ceiling related inversion etcetera.

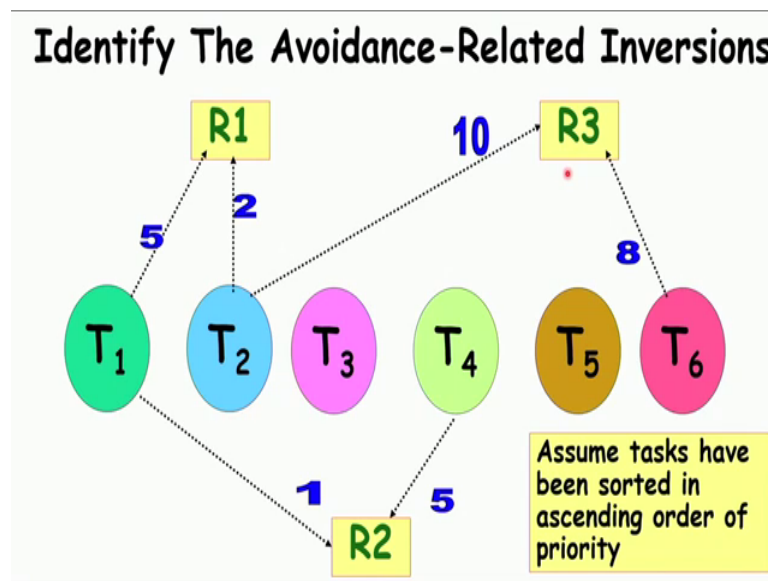
(Refer Slide Time: 14:21)



Just to give an example, a low priority task is using a resource CR 1, but then the CR 1 is also sometimes used by a high priority task with priority 10 and therefore, the ceiling of CR 1 is 10 and when the low priority task uses the resource, the current system ceiling will be set to 10.

Now, let us assume that at that moment a high priority task whose priority is 8 much more than the low priority task here, it starts executing and after some time it needs the resource CR2; CR2 is available, but then the resource will not be granted because the priority of TH will compared with the common system which is 10 and since it is priority is let us than the current system ceiling, will not be granted access to the source which is not being used lying idle, but still it will be not granted and that will call a avoidance related inversion.

(Refer Slide Time: 15:49)



Now, let us try to identify the avoidance related inversions that can occur in the same example that we have been considering. What about uses of R1 by T1 and T2? Will there be any avoidance related inversion? No, not really because a high priority task cannot cause inversion to a low priority task, if only the low priority task can cause inversion to a higher priority task.

Now, let us see, whether R2 can cause inversion to any task. When T4 is using R2, the current system ceiling will be set equal to 1 because T1 also uses R2 and therefore, the ceiling value is associated with R2 is 1 and the current system ceiling will be set to 1.

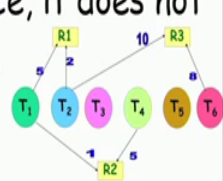
Now, let us assume that T2 requires a resource may be R1 or may be R3 and then it would not be granted resource it will block. So, T2 can suffer avoidance related inversion on account of T4, but for how long, that is for 5 units. Now, what about this resource R3? When T6 is using the resource R3, then the current system ceiling will be set to 2 because the ceiling value associated with R3 is 2.

Now, let us at that time T4 needs the resource R2. T4 priority will be compared with the current system ceiling which is 2 and it will be denied the resource. So, T4 will undergo a avoidance related inversion on account of T6 using R3 and the duration of that will be 8, but what about T1? Can it undergo avoidance related inversion? No, T1 will be granted by R1 because its priority is greater than a system ceiling, but what about T2? Can T2 undergo avoidance related inversion? when it requests a resource let us say R1? Yes, because its priority is just equal to the current system ceiling, it will not be allowed to access to R1 and T2 can also undergo avoidance related inversion.

(Refer Slide Time: 19:05)

Avoidance-Related Inversion

- **Theorem:** Tasks are single-blocking under PCP.
- Once a task acquires a resource, it does not undergo any priority inversion.
- **Corollary 1:**
 - Under PCP a task can undergo at most one priority inversion during its execution.



The diagram illustrates the relationship between tasks and resources. Tasks T1 through T6 are represented by colored circles with priority values: T1 (green, 5), T2 (blue, 2), T3 (purple, 1), T4 (pink, 10), T5 (yellow, 2), and T6 (red, 2). Resources R1, R2, and R3 are represented by yellow circles with priority ceilings: R1 (2), R2 (2), and R3 (2). Arrows indicate resource requirements: T1 and T2 require R1; T2 and T4 require R2; T2 and T6 require R3. The number of units of each resource required is shown next to the arrows: T1 (1), T2 (2), T4 (1), T6 (1) for R1; T2 (1), T4 (1) for R2; T2 (1), T6 (1) for R3.

243

Now, let us see some important results related to the avoidance related inversion. The theorem proof will be available in the text; will just give the basic idea here and the implications. In the priority ceiling protocol the tasks are single blocking, so that is once the tasks blocks a resource because the resource being held by a low priority task, it will not block for another resource that it may need. What is the argument behind that? The argument is that let us say a task needs a 2 resources R1, R3 and then or let us say we

take the R 4 may be, I am sorry T4 or may be T2, they need 2 resources each, sorry only T2 needs 2 resources.

So, ones T2 gets the resource R1, the current system ceiling will already equal to 1 and therefore, the other task cannot really be allowed to grant the resource, they will not be granting the resource. Similarly, if T2 is holding R3, in that situation also current system ceiling will be set equal to 2 and the load priority task like T4 cannot take another resource. So, due to the avoidance a task can be blocked only once for a resource and for other resource it will be not be blocked by lower priority tasks, has a corollary of that we can say that the priority ceiling protocol undergoes at most 1 inversion during it is execution, even though it may be using dozens of resources.

(Refer Slide Time: 21:31)

Why is PCP Deadlock Free?

- Deadlocks occur only when
 - Different tasks hold parts of each other's required resources.
 - Then they request for the resources being held by each other.
- Under PCP, when a task holds some resource,
 - No other task can hold a resource it may need.

244

Priority ceiling protocol is deadlock free, we can argue it in the following line that deadlock occurs when different tasks hold part of each other's resource and then they wait for other 2 release the resource and then they keep on mutually waiting, but here that situation cannot occur because once a task acquires resource, the priority value is set to a current system ceiling and therefore, the other task will not be allowed to access the resource if it is priority is less than a current system ceiling.

(Refer Slide Time: 22:23)

How is Unbounded Priority Inversion Avoided?

- A task suffers unbounded priority inversion, when
 - It is waiting for a lower priority task to release a resources required by it.
 - In the meanwhile intermediate priority tasks preempt the low priority task from CPU usage.



Unbounded priority inversion is also avoided, but how it is avoided, let us try to understand. When a task is using a resource and higher priority task is waiting and in the mean while the lower priority task, the priority of that is enhanced to be equal to the task that is waiting and intermediate priority tasks, they can create a low priority task from using it, but from here due to the inheritance caused due to priority inheritance protocol; priority ceiling protocol, the priority of the low priority task is enhanced to that of high priority task. So, the high priority task cannot undergo unbounded inversion because the low priority tasks priority is already increased and the intermediate priority task cannot really prevent it from CPU users.

(Refer Slide Time: 23:44)

How is Unbounded Priority Inversion Avoided?

- **Inheritance clause:** Whenever a high priority task waits for a resource held by a low priority task,
 - The lower priority task inherits the priority of high priority task.
 - Intermediate priority tasks can not prevent the low priority task from CPU usage.



So, it is the inheritance caused because of that the priority of the low priority task gets enhanced and the intermediate priority task cannot undergo, cannot cause the low priority task to be prevented and therefore, unbounded priority inversion cannot occur.

(Refer Slide Time: 24:07)

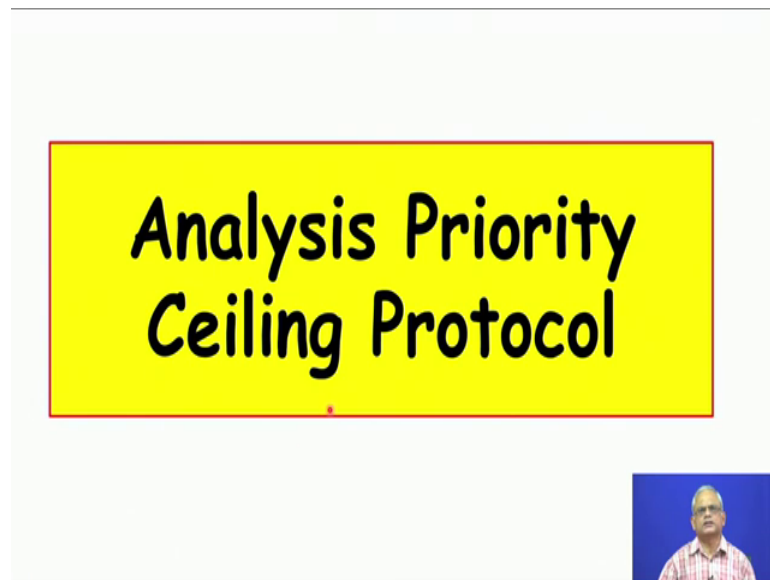
How is Chain Blocking Avoided?

- Already we have seen:
 - Resource sharing among tasks under PCP is single blocking.
 - This gives the clue as to how chain blocking is avoided.



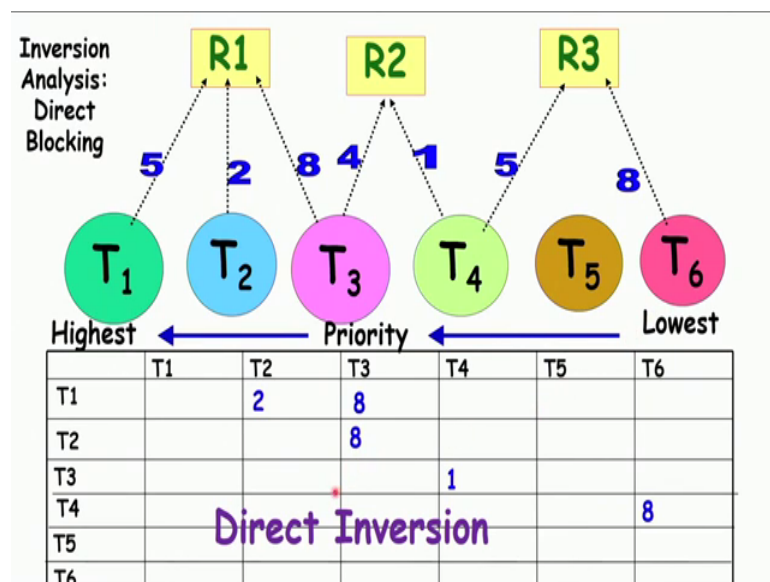
Now, let us to understand how a chain blocking is avoided? It is easy to see that, we have already argued that a task blocks only once per resource and the same proof tells us that it will not block multiple times for a different resources and therefore, chain blocking is avoided.

(Refer Slide Time: 24:35)



Now, let us do some numerical exercise to find out to what duration, to what extent a task may suffer inversions, when the priority ceiling protocol is used. Let us, take some example problems and then try to identify the various type of inversions that can occur.

(Refer Slide Time: 25:01)



We have 6 tasks here and T1 is the highest priority task and T6 is the lowest and the task have been ordered in terms of priority, the resource users that is indicated here, there are 3 resources R1, R2, R3 and the duration for which we need the resource is indicated on the arrow.

Now, first let us identify the direct blocking, which tasks will undergo direct blocking and for what duration and what about T1? T1 uses the resource R1 and if T2 is already using the resource it should have to wait for 2 units; T3 is also uses R1 and therefore, T1 might have to undergo inversion on account of T3 for 8 units.

Now, what about T2? T2 can undergo inversion on account of T3 already using the resource R1 for 8 units. What about T3? T3 does not undergo inversion due to the high priority task, it undergoes inversion for a low priority tasks. So, T3 can undergo inversion on account of T4 for 1 unit and similarly T4 can undergo inversion on account of T6 for 8 units and we have indicated in the form of a table in our problem solving, we will have to use tables to compute the inversions; the various type of inversions the direct inversion, the inheritance involved inversion and avoidance related inversion.

So, here T1 undergoes inversion for 2 units on account of T2 and it can undergo inversion of 8 units on account of T3. T2 can undergo inversion of 8 units on account of T3 and T3 can undergo inversion of 1 unit on account of T4 using the resource R2 and T4 can undergo inversion on account of T6 on the resource R3 and that is for 8 units and we have indicated that avoidance related inversion in the next lecture.

Thank you.