

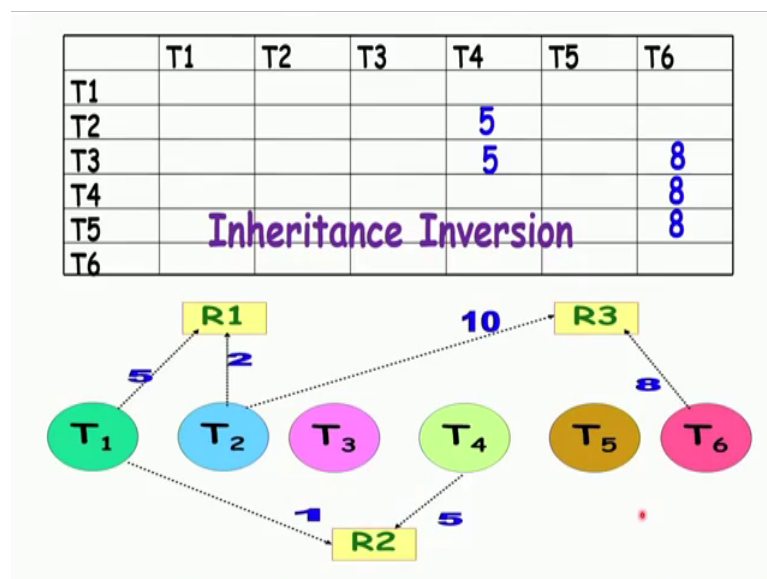
Real Time Operating System
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 18
Analysis of PCP Priority Inversions

Welcome to this lecture, in the last lecture we are trying to do some numerical analysis to identify the inversions that are suffered by different tasks and the duration for which the inversions occur, for an example problem we had done the analysis for direct inversion.

Now, let us continue from there, we will consider the same problem and we will now try to identify the inheritance related inversions and the duration and we will represent in the form of a table.

(Refer Slide Time: 01:00)



This is the same problem that were considering in the last lecture and now we are trying to identify the inheritance related inversion. T1 and T2, will not cause any inversion to the lower priority tasks because they are of already higher priority and higher priority tasks does not cause inversions. It is the lower priority tasks, which cause inversion to the higher priority tasks.

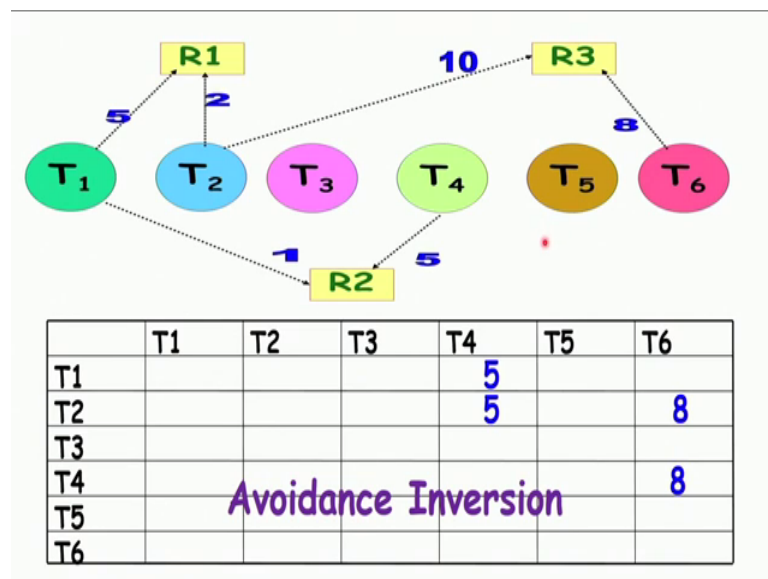
Now, let us see whether T4, let us look at the resource R2 here and T4 is using the resource and then T1 is waiting for the resource, in that case the priority of T4 will get enhanced to be equal to T1 and now the tasks which are of higher priority T2 and T3, can

undergo inheritance inversions, but what will be duration for which they will undergo inversions T2 and T3, they will undergo inversion for 5 units time, that is the uses of T4 of the resource R2, but what about the resource R3?

When T6 is using the resource R3 and T2 is waiting for the resource T3, T4, T5 will be prevented from using CPU, because T6 is executing with a high priority that is T, the priority of T2 and what will be the duration for which T3, T4 and T5 will undergo the inheritance related inversion, so that is equal to 8. So, all the 3 of them will undergo 8 units at most inversion on account of T6 using the resource R 3 and that we have represented here that T3, T4 and T5 can undergo 8 units of inversion on account of T6 and T2 and T3 can undergo 5 units of inversion on account of T4 using the resource R2.

Now, let us do the analysis for avoidance related inversion for the same example problem.

(Refer Slide Time: 04:03)



So, this is the problem, now let us identify the inheritance sorry, the avoidance related inversions. Now, let us see whether T1 and T2 will cause avoidance related inversion? No, because these are of low priority tasks and high priority tasks do not cause inversion to lower priority tasks, it is the low priority tasks which cause inversion to higher priority tasks.

Now, let us see on account of this resource R2, let us say T4 is using a resource R2, then the current system ceiling will set equal to the priority of the task 1, T1 and in that situation T2 will not be allowed access to any of the resource that it needs. So, to what extent to what duration it can suffer avoidance inversion? That is 5, that is equal to the resource uses of T4 for R2 and even T1 can suffer inversion, when it requests resource R1 because its priority is not greater than the ceiling priority.

Now, what about R3? When T6 sets the, it uses the resource R3, the csc is set equal to the priority of T2 and in that case if T4 is trying to use the resource R2, it will be prevented and the maximum duration for which it can undergo inversion is 8 units and similarly T2 will not be allowed to use a resource R1 because its priority is not greater than the ceiling priority of R3.

So, we can represent that in the form of a table, on account of T4 using R2, T1 and T2 undergo 5 units of inversion, on account of T6 using resource R3, T4 and T2 undergo inversion, T3 and T5 do not go avoidance related inversion because they do not use any resource.

(Refer Slide Time: 07:13)

Analysis of Inversions

- Each inversion table is an upper triangular matrix:
 - Why?
 - A task does not suffer any inversions due to higher priority task.

252

If we look at the tables that we developed they are always in the form of an upper triangular matrix. So, all the values are in the upper triangle, why is it so? Can you think of any reason? The answer is that a task does not suffer any inversion due to higher priority tasks, if a higher priority task is using a resource the lower priority task need to

it, that is not a inversion inversions occur when a lower priority task is using resource and higher priority task is waiting.

So, it is not really symmetrical the inversion, occur in one way, when the low priority task is using resource and high priority task is waiting and not the other way round and therefore, it is a upper triangular matrix.

(Refer Slide Time: 08:19)

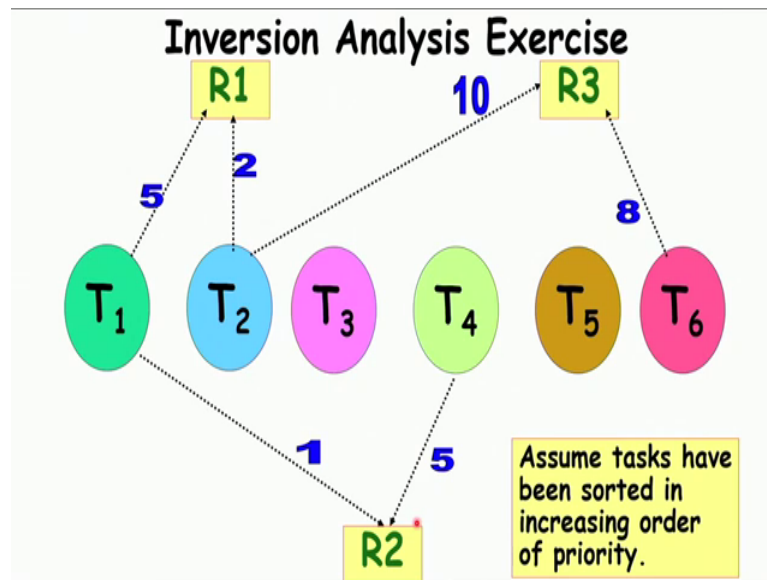
Maximum Inversion for a Task

- A task can suffer:
 - At best one of direct, inheritance, or avoidance-related inversion.
- Therefore the maximum inversion that a task can suffer is:
 - The maximum of the entries of the corresponding row of the inversion table₂₅₃

Now, let us identify what is the maximum duration for inversion that a task may undergo, considering all types of inversion. The first thing to note is that a task can suffer at best only one of the inversions either direct inheritance or avoidance related inversion, it cannot suffer multiple inversions of these type. Once, we agree to this result that a task can suffer at most, one of the inversions.

Then to determine the maximum inversions that a task can undergo, we need to look at the 3 tables that we developed and then look at the maximum value of the entry corresponding to that task in the 3 tables and that is the maximum duration for which it can undergo inversion.

(Refer Slide Time: 09:23)



There are some practice problems, you can please practice some problems where you may be able to construct 5 tasks or 7 tasks, 2 resources or maybe 4 resources, 3 resources and so on, and please try to work out some examples in the analysis of the inversions that different tasks can undergo.


(Refer Slide Time: 09:53)

Liu and Lehoczky Condition Under Resource Sharing

- Let b_i denote:
 - The longest time for which a task T_i can undergo priority inversions due to resource sharing.
 - T_i will meet its first deadline if,

$$(b_i + e_i + \sum_{j=1}^{i-1} \left\lceil \frac{p_i}{p_j} \right\rceil * e_j) \leq p_i$$

where p_i is the period of task T_i and

$$p_1 \leq p_2 \leq p_3 \leq \dots \leq p_n$$


Now, even if we know, what is the maximum duration for which a task can undergo inversion, how do we check the schedulability? Please remember, that in the rate monotonic scheduler we are using the Liu and Lehoczky result to determine the

schedulability for each tasks we were using (Refer Time: 10:25) time of the task and time for which the all higher priority tasks need to use the resource in the form of a such expression and then checking whether it is less than equal to the deadline of the period of the task, deadline is equal to the period.

Now, when the tasks say resources, when we need to just art this term here b_i and b_i is the total blocking time for a resource on account of the lower priority tasks. So, once we determine develop the 3 tables for 3 different types of inversions when priority ceiling protocol is used, we determine the highest entry corresponding to that task and use that value here and then determine whether the tasks set can be schedulable.

(Refer Slide Time: 11:27)

PCP for Dynamic Priority Systems

- The priority ceiling values need to change dynamically with time.
 - **A solution:** Each time the priority of a task changes:
 - Update the priority ceiling of each resource and the current system ceiling.
 - However, this would incur unacceptably high processing overhead. *

256

Now, can the priority ceiling protocol be used in the dynamic priority systems for example, the earliest deadline first scheduler, can you use a resource sharing protocol with EDF. In the dynamic priority systems, the priorities of the tasks keep on changing with time depending on how close there to the deadline the priority keeps increasing, but if you look at the protocols that we considered for resource sharing, we assign ceiling priority based on the priorities of the task that uses that resource and if the priorities of the tasks keep on changing, then we have to keep on re computing the ceiling priorities.

Each time the priority of a task changes, we need to change the current system ceiling, need to change the ceilings associated with the resources and this can be too much of overhead because there can be many instances, frequently the priorities of tasks will

change and each time we need to do this that the current system ceiling the ceilings associated with resources and so on.

We need to change and therefore, if you are using a dynamic priority scheduling algorithm it becomes very difficult to have any resource sharing protocol to be used in that situation and therefore, if you analyze all real time applications, where the tasks are non pre-emptive, they share resource with each other, then it is invariably rate monotonic scheduler or small variation of that is used.

(Refer Slide Time: 13:38)

• **PIP:** **Comparison of Resource Sharing Protocols**

- Simplest --- requires minimal support from the OS.
- Effectively overcomes the unbounded priority inversion problem.
- **However, tasks may suffer from chain blockings and deadlocks.**

257

Now, let us compare these 3 protocols that we looked at because all the 3 protocols are used in some applications or other depending on their advantages. So, let us just recapitulate their advantages and disadvantages. The simplest protocol was the priority inheritance protocol, if you are using a very small application, a small embedded system which hardly has an operating system, then the priority inheritance protocol is the one to use, it requires very little support from the operating system, it overcomes the unbounded priority inversion, but then it suffers from chain blocking and deadlocks and we need to program carefully. So, that chain blocking and deadlocks will not arise. So, the priority inheritance protocol is used for small applications, but what about the highest locker protocol?

(Refer Slide Time: 14:46)

Comparison of Resource Sharing Protocols

- **HLP:**
 - Requires moderate support from the OS.
 - Solves the chain blocking and deadlock problems.
 - However, intermediate priority tasks:
 - May suffer from inheritance-related inversions.

258

It requires moderate support from the operating system, solves the chain blocking and deadlock problem, but then it introduces a new problem that is blocking by the intermediate priority tasks, they suffer inheritance related inversions and that can be substantial.

(Refer Slide Time: 15:13)

Comparison of Resource Sharing Protocols

- **PCP:**
 - Overcomes shortcomings of PIP.
 - Free from deadlocks and chain blocking.
 - Low inheritance-related inversions.
 - Priority of a task on acquiring a resource does not change :
 - Until a higher priority task requests the resource.

259

The priority ceiling protocol overcome most of the problem of the basic inheritance of the protocol and the highest locker protocol. It is free from deadlock and chain blocking and in contrast to the highest locker protocol; the inheritance related inversions are really

low here. The main reason is that the priority of a task on acquiring a resource does not change until a higher priority task request a resource.

(Refer Slide Time: 15:48)

Multiprocessor Scheduling: A Difficult Problem

- “The simple fact that a task can use only one processor even when several processors are free at the same time adds a surprising amount of difficulty to the scheduling of multiple processors” [Liu’69]

Now, with that discussion let us conclude our discussions on resource sharing among tasks real time tasks.

Let us, now start discussing about multi processor scheduling because now a days the multi processor have become common place, even small embedded applications they use multi processor and invariably all desktops, all servers, laptop, handheld devices etcetera all use multi processors, but the results that we have so far seen are for uni processors. Let us see, how those results can be extended for use with a multi processor situation.

But multi processor scheduling is a much more difficult problem than the single processor. In the words of Liu, who is worked considerably in the area of scheduling real time tasks, the Liu Lehoczky Liu lel and algorithms are all due to him, in his words the simple fact that a task can use only 1 processor even when there are several processors which are free at the same time adds a surprising amount of difficulty to the scheduling of multiple processors.

Now, let us see what are the difficulties and will look at some simple algorithm, will not spend too much time here because this, the large topic, but will not really discuss in depth, will just look at some simple strategies for scheduling on multi processor.

(Refer Slide Time: 17:45)

Multiprocessor RT Task Scheduling

- Most theoretical results reported over last 30 years
- Many are NP-hard problems
- Few optimal results
- Heuristic approaches
- Simplified task models

Most of the results here having obtained in the last 30 years and most of the real time multi processor scheduling problems are NP-hard problems and therefore, simple optimal solutions do not adjust. There are very few optimal results, mostly heuristic approaches are used and also we use a simplified task models.

(Refer Slide Time: 18:19)

The Multiprocessor Scheduling Problem

- **Actually two separate problems.**
 - **Task assignment problem:**
 - How to assign tasks to processors?
 - **Scheduling problem:**
 - How to schedule the tasks on the processor to which it has been assigned.



If we look at the multi processor scheduling problem, then there are actually 2 sub problems 1 is task assignment problem, the other is the scheduling problem. In the assignment problem, we first decide which task will run on which processor and then


how or in what time it will run. So, there are 2 sub problem 1 is which task will run and processor? The second is how it will run the scheduling algorithm?

So, the task assignment problem is how to assign the task to the processor? The scheduling problem with; how to schedule the task and the processor to which it has been assigned?

(Refer Slide Time: 19:05)

Optimal Schedulers?

- Optimal schedulers for uniprocessors:
 - **Static** --- **RMA**
 - **Dynamic** --- **EDF**
- What are their complexities?
 - **Linear** for RMA
 - **Log n** for EDF
- General real-time task scheduling in multiprocessor /distributed systems :**NP hard**



But, as you already indicated that optimal schedulers for multi processors are not there because these are all NP-hard problems, but then, what are the optimal schedulers for uniprocessors? We have already seen this, that for static priorities systems rate monotonic schedulers are the optimal schedulers. In a dynamic priority situation the earliest deadline first is the optimal scheduler, but then what are the complexities of this schedulers, the rate monotonic scheduler at EDF.

If you look at the implementation aspects that we are considering, it is linear for rate monotonic schedulers because if you remember and set that there are fixed priorities and then the implementation that we heard we could run the scheduler in $O(n)$ priorities. So, the linear for R rate monotonic and $\log n$ for EDF and even rate monotonic we have more efficient scheduler one scheduler that also we had discussed, but the general real time scheduling multi processor distributing systems is a NP-hard problem.

(Refer Slide Time: 20:38)

A Few Important Task Assignment Algorithms

- **Static:**
 - Utilization balancing algorithm
 - Next-fit algorithm for RMA
 - Bin packing algorithm for EDF
- **Dynamic:**
 - Focused addressing and bidding algorithm
 - Buddy algorithm

264

Now, let us look at some task assignment algorithms and then we will see the scheduling algorithms because multi processor scheduling consists of 2 things, the task assignment and also the task scheduling once they have been assigned to a processor.

So, let us look at the task assignment algorithm, there are some static assignments. So, here the tasks are assigned statically to a processor then execute only on that processor once we have assigned, whereas we also have dynamic algorithms, where a task can migrate from 1 processor to other and a different time instance it can execute on different processors, naturally the static ones are simple and more efficient and few examples are the utilization balancing algorithm, next-fit algorithm for RMA, bin packing algorithm for EDF, whereas for dynamic priority where the focused addressing and bidding algorithm and the buddy algorithm.

We do not have time to look at all these algorithms, even though in the textbook these are given we will just look one of the algorithm to get a flavor of the task assignment algorithm; we will look only at the utilization balancing algorithm.

(Refer Slide Time: 22:02)

Utilization Balancing Algorithm

- Maintain tasks in a queue:
 - In increasing order of their utilizations.
- Remove tasks one by one from queue:
 - Allocates to the lowest utilized process.
- Usually, $\bar{u} \neq u_i$
- Used when tasks in the node are scheduled using EDF. Why?

265

Here, the tasks are maintained in a queue and these are arranged in the increasing order of the utilization, we keep we take out 1 task from the queue at a time and check which is the processor that is currently the most underutilized processor, we assign it to that, so basically a greedy algorithm.

We will look at the 1 which is currently the least utilized and then assign the task to that and finally, we will see that the utilization of the individual processors after the algorithm completes assignment is not really equal to the mean utilization or in other words the utilization is not really fully balanced by using this greedy algorithm and the utilization balancing algorithm is used in the conjunction of EDF. The allocation is done by the utilization balancing algorithm, but the individual processor scheduling is done by the EDF algorithm.

Can you think of any reason why EDF is used not the rate monotonic scheduler? The answer is that the in the EDF the balance the utilization of the processor can be up to 1, whereas in the rate monotonic we have seen, how many the utilization depends on how many tasks are assigned to the processor and so on that we have not considered in our utilization balancing algorithm and therefore, it is the EDF that is used along with the utilization balancing algorithms, will not really look into the other algorithms, because this is a short course of only 10 hours. Now, we will look at few more basic issues in real time operating systems.

So far, we have been looking at a very central issue of all real time operating systems that is task scheduling. Now, let us look at some issues for real time operating systems. Can we distinguish a real time operating system from a traditional operating system? Or to state in other words, can we identify how does a real time operating system differ from a traditional operating system? What are the points that we need to consider to check and say that whether in operating system is a you can be used for real time application or is it traditional non real time operating system? The resource that we need to consider are, support for real time priority levels.

(Refer Slide Time: 25:35)

Basic Requirements of an RTOS

- Real-time priority levels
- Real-time task scheduling policy
- Support for resource sharing protocols
- Low task preemption times:
 - Of the order of milli/micro seconds
- Interrupt latency requirements

267

We have seen that the real time scheduler specially the rate monotonic algorithm, there once we assign priorities to the tasks during design time, the priority should not change this is also called as static priority level or real time priority level. As we will see in our next lecture that the traditional operating systems they keep on changing tasks priorities, every time the task is run it is priorities recalculated and new priority values are assigned. So, those are not really static priority levels, a real time operating systems must have support for static priority level, once the designer assigns a priority to a task, it should remain fixed to that priority level.

The second is real time task scheduling policy; the operating system should support some real time tasks scheduling policy like the rate monotonic scheduler and this scheduling policy the central requirement for all real time operating systems that we had seen.

Support for resource sharing protocols, protocols such as priority ceiling protocol or at least the basic inheritance scheme should be supported by the operating system to be able to develop any non trivial real time application. Also, the preemption time of task should be of the order of micro seconds or may be milli seconds, but it cannot be, should not be seconds because many real time application, the response time itself is only few milli seconds.

But, if you look at the traditional operating system such as the Unix the preemption time is the order of a second and also we have interrupt latency requirements, the latency should be small that is when interrupt occurs to the time the interrupt is processed the time should be small.

(Refer Slide Time: 28:00)

Additional Requirements of an RTOS

- Memory locking support
- Timers
- Real-time file system support
- Device interfacing

268

There are additional requirements like memory locking support that once a task uses certain memory, there should not be soft to the hard disk and so on; it should be locked to the memory.

Support for timers, because timers are very crucial in real time applications both the periodic timer and the aperiodic timers. Support for real time file support, real time file system should be supported and the resource will elaborate in the next lecture and the device interfacing facilities because in many of this real time applications, we need to work with different types of peripheral devices, with this we will stop here, where at the end of the lecture and will elaborate this issues in the next lecture.

Thank you.