Real Time Operating System

Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

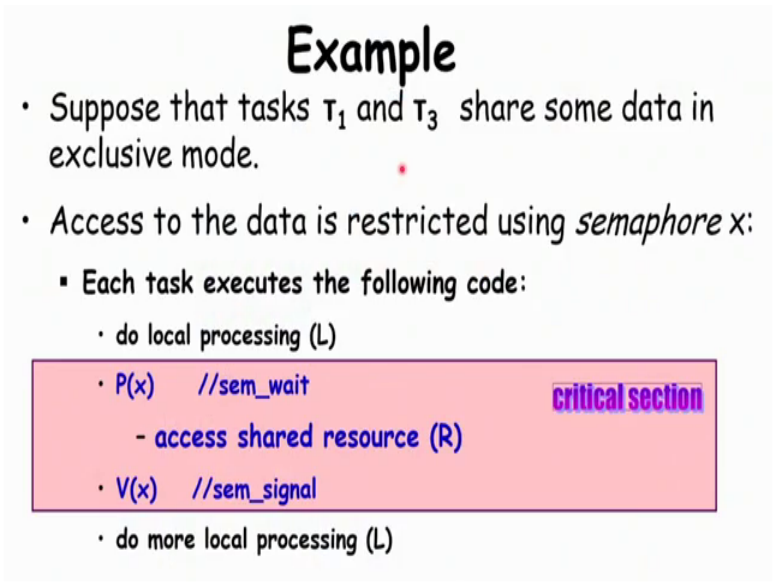
Lecture – 14
Solution to Priority Inversion Problem

Welcome to this lecture in the last lecture, we were looking at how resources can be shared among tasks in the framework of tasks scheduling maybe a rate monotonic scheduler or and earliest deadline first scheduler. And we had discussed about critical sections semaphores which are basically topics that are discussed, in the first level operating system course and we did not really spend time discussing semaphores and critical sections.

But then we had remarked that the semaphore solution used in a traditional operating system cannot be used as it is in a real time application because it can lead to severe problems. And the problems are known as priority inversion, and unbounded priority inversion a simple priority inversion arises when a low priority task has locked a non preemptable resource, and a higher priority task cannot execute and just waits for a resource. So, we have a priority inversion example where a low priority task is executing and high priority task is waiting, but then the priority inversion problem by itself is not too bad it can be solved using careful programming as we will see.

But then what really is a difficult problem is unbounded priority inversion, where the low priority task which is holding the resource is preempted from CPU uses by other intermediate priority tasks which do not need the resource, towards the end of the last lecture we are discussing about a critical section and how priority inversion can arise, unbounded priority inversions and what are the solutions.

(Refer Slide Time: 02:39)



Example

- Suppose that tasks T_1 and T_3 share some data in exclusive mode.
- Access to the data is restricted using *semaphore* x:
  - Each task executes the following code:
    - do local processing (L)
    - P(x) //sem_wait
      - access shared resource (R)
    - V(x) //sem_signal
    - do more local processing (L)

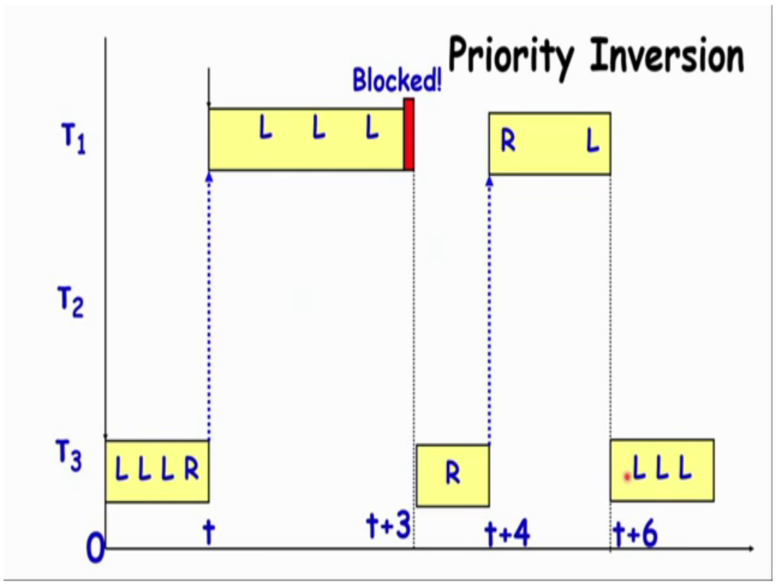
critical section

Now, let us look at that same example that we have task T 1 and task T 3 that need to share a non preemptable resource; that means, they need to access that resource in exclusive mode, and only when a task which is already using the resource completes its execution using the resource no other task can use it both the tasks are the following type of code, they have some part of the program where they do some processing without using the resource.

But then when they want to use the resource, because it is a non preemptable resource before they start to use the resource they invoke p x which is basically wait for semaphore, and after getting access to the semaphore they starting executing the shared resource. And after completing the code segment here after the part of the code using the shared resources is over they execute the instruction V x which is signal semaphore to release the resource.

And then more local processing without using the resource, the part of the code in which the shared resource is used by a task starting with a same wait signal till the same signal is the critical section of the code.

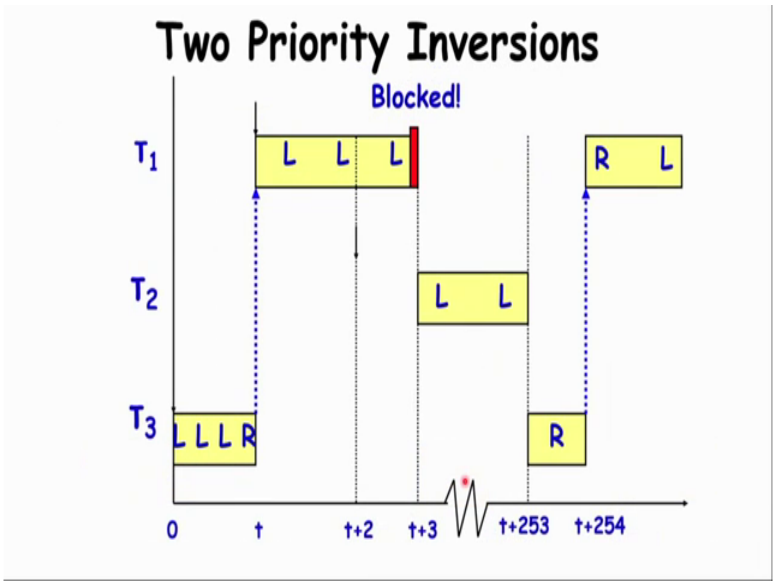
(Refer Slide Time: 04:30)



Now let us see how the priority inversion in such a situation can arise, we have a task T 3 which is of low priority does some local processing denoted by L and after sometime it gets into the critical section. So, before starting using resource it does a wait semaphore and because, the semaphore is available it could get the resource and starts executing using the resource, but at that point of time that is T a high priority task which is T 1 starts executing, and it does some local processing here, and until a point t plus 3 where it gets into its critical section and does a lock resources or a same width instruction execution.

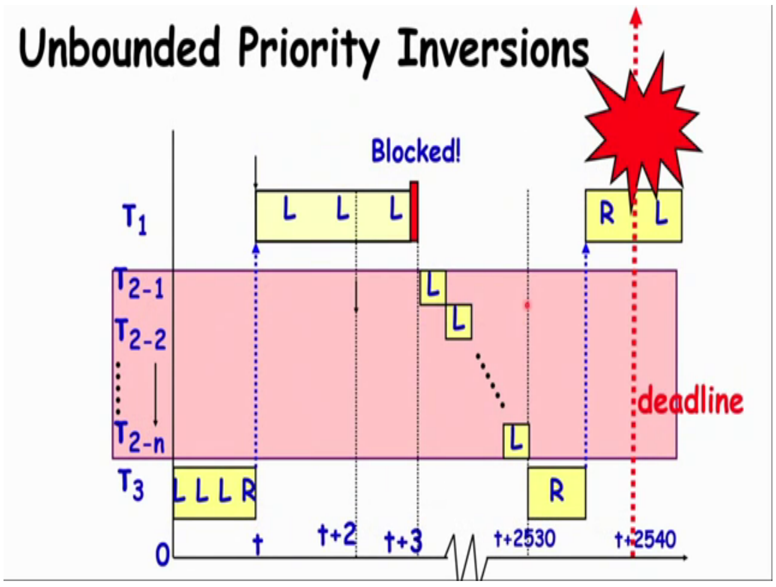
And since the resource is already held by T 3 the task T 1 gets blocked, and once it gets blocked T 3 becomes active uses a resource and after sometime at t plus 4 it completes the execution, and then T 1 gets the resource starts executing does local processing, and then after it completes then T 3 gets the CPU and starts executing. So, this is an example of a single priority inversion, and the time for which the task T 1 gets blocked is between t 3 and t 4 where the task t 3 completes uses of its critical resource. Then let me just ask this question that what is the maximum duration for, which task T 1 can suffer priority inversion. The answer to this question is that its equal to the largest duration for which the task T 3 would need to use the critical resource R.

(Refer Slide Time: 07:14)



Now let us look at the unbounded priority inversion problem due to the same resource R. The task T 1 can suffer 2 priority inversion, when the task T 2 which does not need the resource starts executing and preempts T 3 from using the CPU, and therefore, T 1 suffers 2 priority inversion 1 on account of T 3 and the other on account of T 2, this T 2 is executing and T 3, T 1 would be waiting because it does not have the resource and T 3 is holding the resource and not being able to execute because the T 2 has started executing. So, there are 2 priority inversions as you can see, 1 is an account of T 3 for this duration, and another on account of T 2 for this duration.

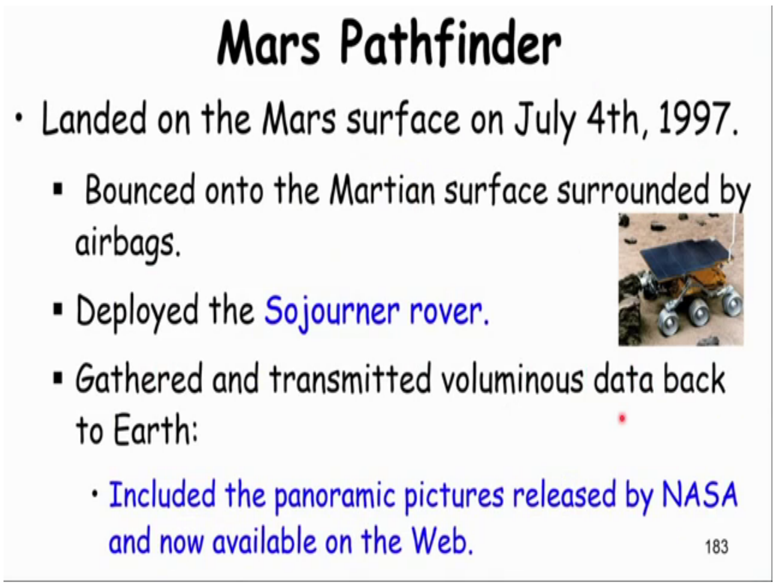
(Refer Slide Time: 08:23)



If we extend this example we come to the example of a unbounded priority inversion here, we are considering multiple intermediate tasks here, intermediate priority task T 3 is the lowest priority T 1 is the highest priority, but there are T 2 1, T 2 2, T 2 n etcetera these are intermediate priority tasks and once T 1 blocks for the resource for T 3 the intermediate priority task keep on executing 1 after the other.

And T 1 keeps on waiting for T 3 to be able to get the CPU complete its execution, but by that time T 1 misses its deadline. So, there is a application failure.

(Refer Slide Time: 09:20)



Mars Pathfinder

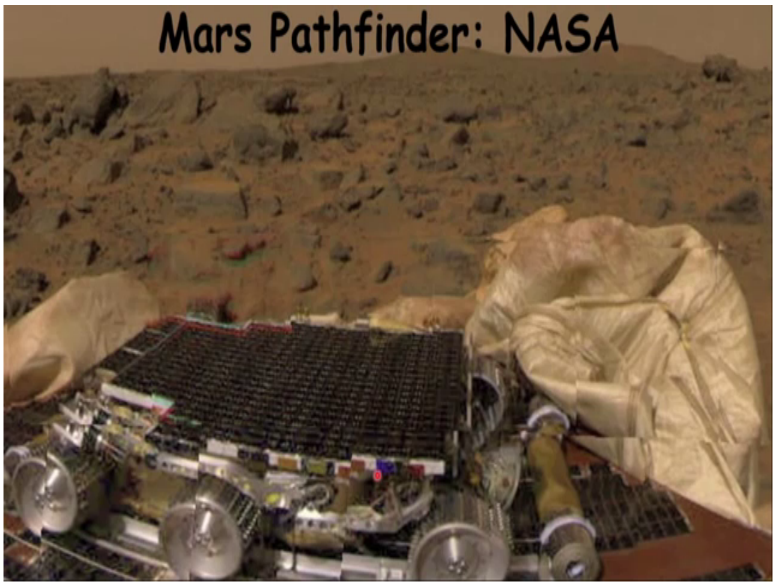
- Landed on the Mars surface on July 4th, 1997.
  - Bounced onto the Martian surface surrounded by airbags.
  - Deployed the Sojourner rover.
  - Gathered and transmitted voluminous data back to Earth:
    - Included the panoramic pictures released by NASA and now available on the Web.

183

Unless the unbounded priority problem is handled adequately a real time application can fail there are many examples in the past where application failure was caused by this problem of unbounded priority inversion, but possibly the most celebrated example is the mars path finder, let us see what happened, there mars path finder landed on the mars surface on 4th July 1997.

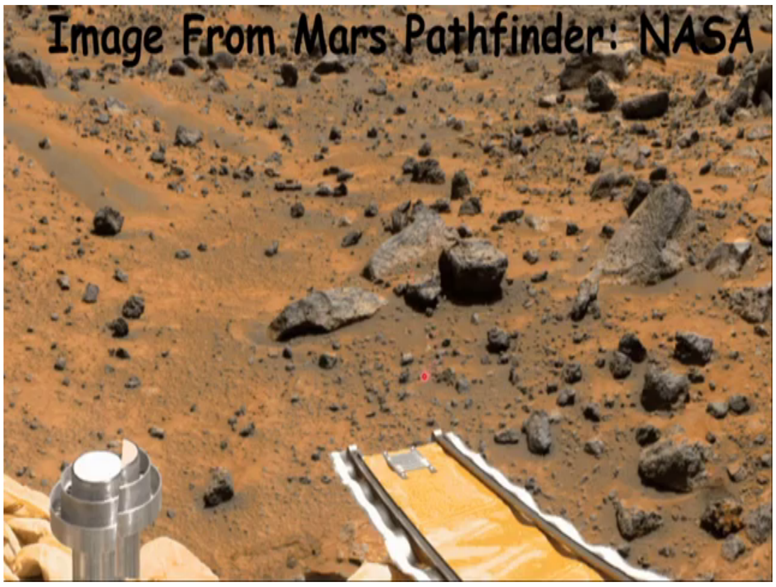
And as it enter the mars surface the vehicle which is the sojourner rover it was surrounded by air bags slowly it dropped down to the mars surface, it moved out from the airbag on to the mars surface, and started transmitting to the earths station large amount of data. And this data included pictures of the mars surface, and which were released by NASA and their publicly available on the web.

(Refer Slide Time: 10:48)



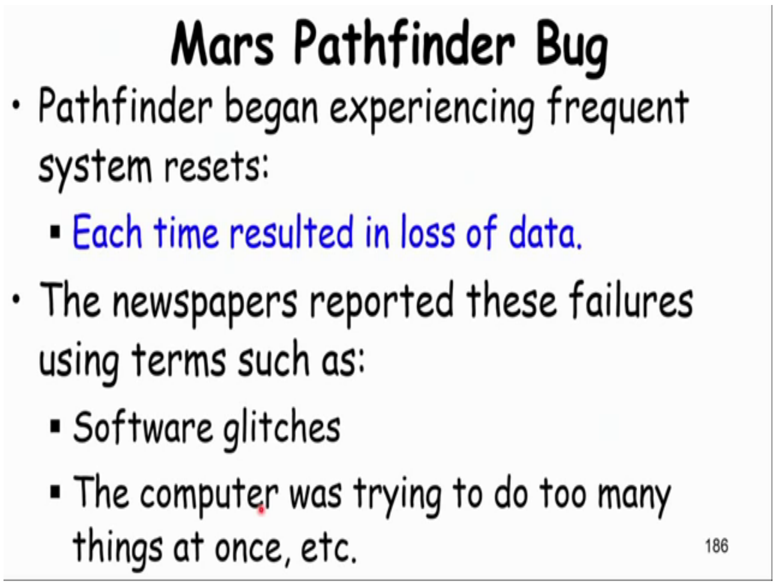
A few of the pictures here, a large number pictures were transmitted by the rover 1 picture of the mars surface the rover has landed on the moon surface surrounded, by bags and started taking pictures.

(Refer Slide Time: 11:10)



This is another image from the mars path finder courtesy NASA released to the public.

(Refer Slide Time: 11:22)



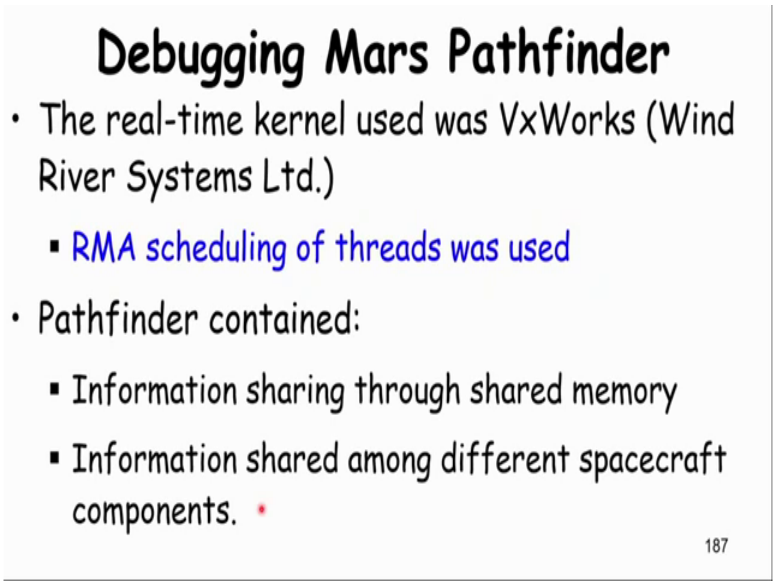
Mars Pathfinder Bug

- Pathfinder began experiencing frequent system resets:
  - Each time resulted in loss of data.
- The newspapers reported these failures using terms such as:
  - Software glitches
  - The computer was trying to do too many things at once, etc.

186

But then the pathfinder began experiencing system resets. The system after every few seconds started resetting itself, and as a result data could not be transmitted only a part of the data would be transmitted and it reset, and again it starts transmitting and resets so, complete data could not be transmitted. Newspapers at that time they reported that the mars path finder is experiencing failures, and the terminologies they used to describe the failure is there are software glitches, which is causing the pathfinder to fail some other newspapers reported that the computer in the pathfinder was trying to do too many things at once and therefore, was failing and so, on.

(Refer Slide Time: 12:32)



Debugging Mars Pathfinder

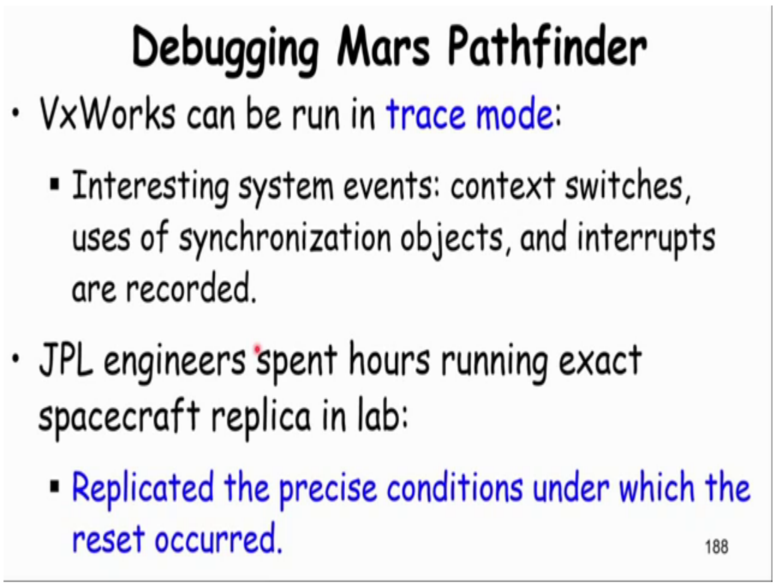
- The real-time kernel used was VxWorks (Wind River Systems Ltd.)
  - RMA scheduling of threads was used
- Pathfinder contained:
  - Information sharing through shared memory
  - Information shared among different spacecraft components.

187

But then in the laboratory in NASA they were desperately trying to fix the probe. In the mars pathfinder V x works which is a real time operating system, from the Wind River systems was being used, and the rate monotonic scheduling of threads was used.

And in the pathfinder different tasks the shared resource sorry the shared information through shared memory 1 task would write to the memory, and other task would read it and so, on. So, the different components of the spacecraft could communicate with each other through shared memory.

(Refer Slide Time: 13:30)



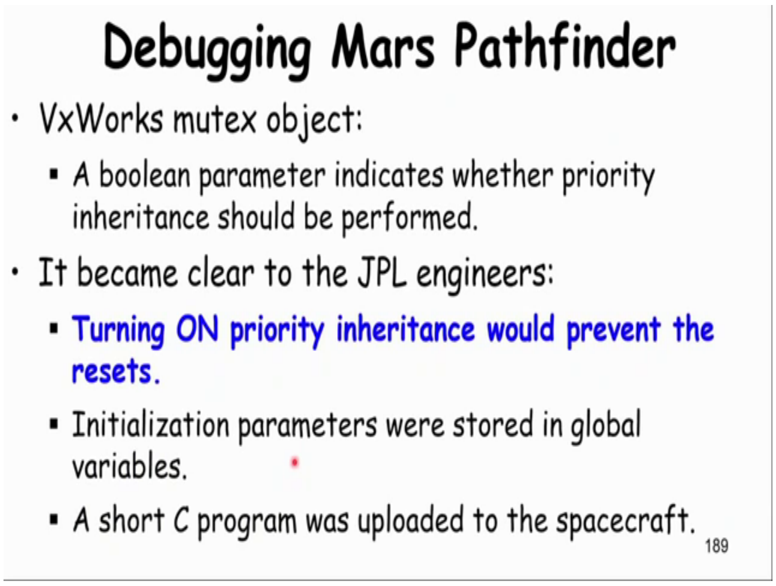
Debugging Mars Pathfinder

- VxWorks can be run in trace mode:
  - Interesting system events: context switches, uses of synchronization objects, and interrupts are recorded.
- JPL engineers spent hours running exact spacecraft replica in lab:
  - Replicated the precise conditions under which the reset occurred.

188

To debug the pathfinder the NASA scientists they had a replica of the pathfinder in their lab, they started doing a simulation of what used to happen in the mars surface, but they set the debug mode on where they can find out where context switches occur synchronization among objects take place. When interrupts occur all these they could see and finally, after several hours of working with the replica of the pathfinder they could converge and precise conditions under which the reset occurred.

(Refer Slide Time: 14:25)



Debugging Mars Pathfinder

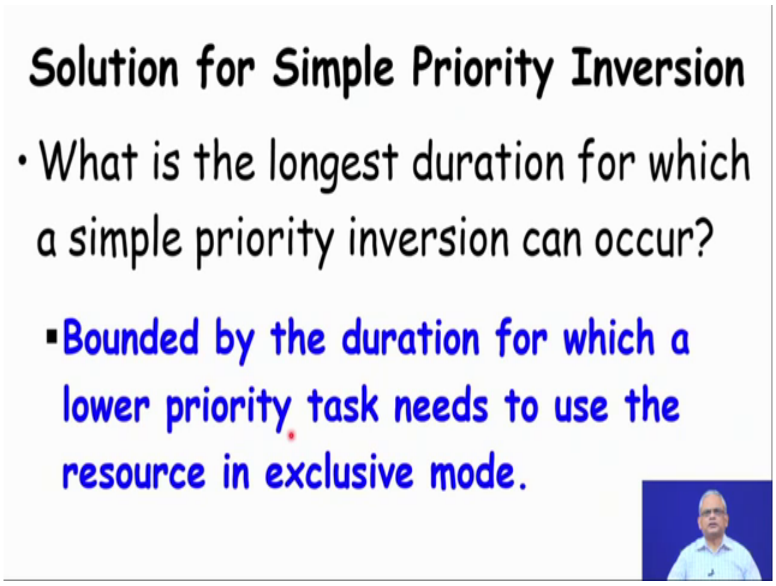
- VxWorks mutex object:
  - A boolean parameter indicates whether priority inheritance should be performed.
- It became clear to the JPL engineers:
  - **Turning ON priority inheritance would prevent the resets.**
  - Initialization parameters were stored in global variables.
  - A short C program was uploaded to the spacecraft.

189

They found that there was a mutex a semaphore that was being used, and a Boolean parameter a global variable where is used which was a Boolean global variable, and depending on whether it is the value of the Boolean variable whether the priority inheritance is to be performed or not that 2 was indicated by the Boolean parameter. We will see what is the what is meant by priority inheritance, but then they found that the Boolean variable had set the priority inheritance of, and they could on their replica of the mars pathfinder found that the problem can be solved if the priority inheritance could be turned on.

And then they uploaded a short c program on to the mars path finder, and then the unbounded priority inversion that was occurring could be tackled. In summary I can say that the mars fight pathfinder was experiencing problem, because the solution to the unbounded priority inversion in the form of a priority inheritance procedure was disabled by the Boolean variable. So, the enabled the priority inheritance procedure, and then it could solve the unbounded priority inversion problem.

(Refer Slide Time: 16:18)

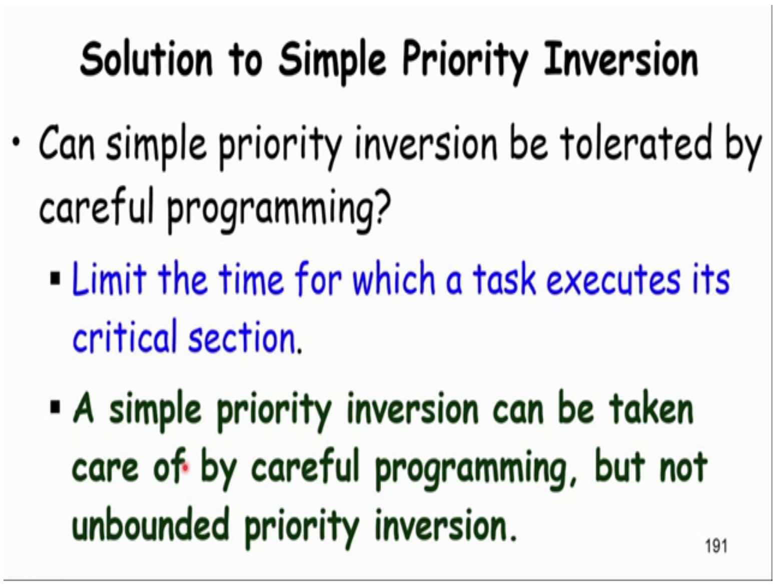


Solution for Simple Priority Inversion

- What is the longest duration for which a simple priority inversion can occur?
  - Bounded by the duration for which a lower priority task needs to use the resource in exclusive mode.

Now let us see what are the solutions for the priority inversion and multiple priority inversions, first let me ask this question that if we a task is a high priority task is under go in priority inversion on account of a lower priority task, what is the longest duration for which the priority inversion can occur, if you think of it we had actually a mentioned it only few minutes back we had said that a single priority inversion can occur for a duration that is bounded by the duration for which lower priority task needs the resource n exclusive mode. Or in other words the maximum time for which a high priority task can suffer priority inversion on account of a lower priority task. Is the maximum time for which the lower priority task needs to execute its critical section.

(Refer Slide Time: 17:40)



Solution to Simple Priority Inversion

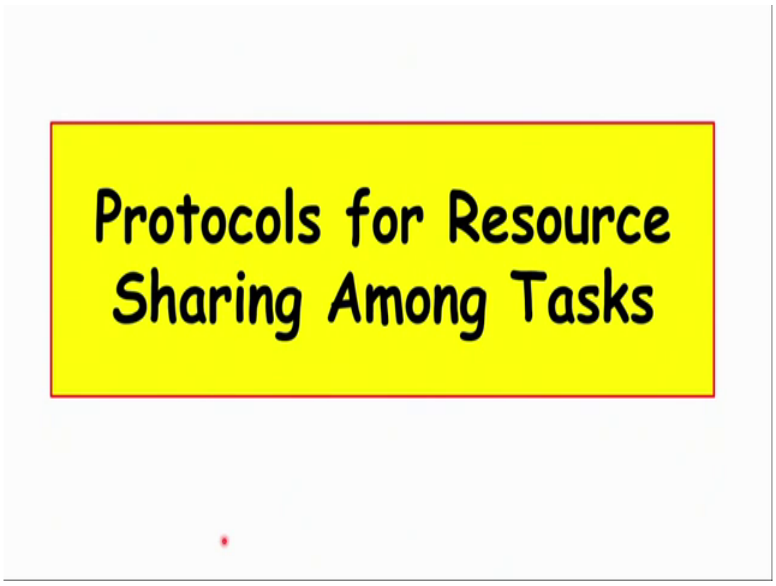
- Can simple priority inversion be tolerated by careful programming?
  - Limit the time for which a task executes its critical section.
  - A simple priority inversion can be taken care of by careful programming, but not unbounded priority inversion.

191

Given that the time for which a lower priority task needs to use the critical section is the time for which a higher priority task, suffers priority inversion can we solve single priority inversion problem, through careful programming. The answer is yes we can solve the simple priority inversion problem through careful programming, we can restrict the time for which the low priority task uses its critical section. Each time there is a consistent point, in its uses a critical section it is just releases the resource.

So, through careful programming the duration for which a low priority task needs to use the non preempt able resource can be made very small. A simple priority inversion can be taken care by careful programming, but we will see that the unbounded priority inversion problem cannot be handled in similar way through careful programming. Let us see how to solve the unbounded priority inversion, because this is the difficult problem, simple priority inversion can be solved through careful program.

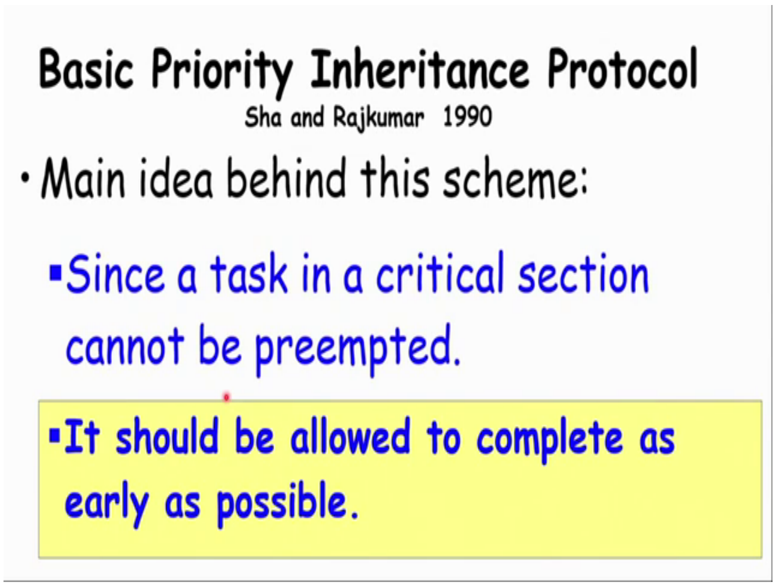
(Refer Slide Time: 19:21)



Protocols for Resource Sharing Among Tasks

The solutions to the unbounded priority inversion are called as protocols for resource sharing among tasks.

(Refer Slide Time: 19:34)



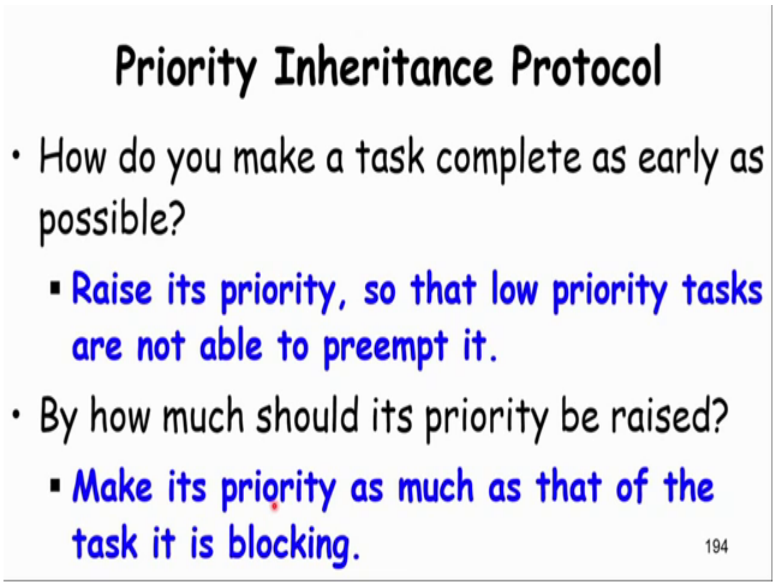
Basic Priority Inheritance Protocol

Sha and Rajkumar 1990

- Main idea behind this scheme:
  - Since a task in a critical section cannot be preempted.
  - It should be allowed to complete as early as possible.

The simples of these protocol is called as the basic priority inheritance protocol. This proposed by Sha and Rajkumar 1990, the main idea behind the basic priority inheritance protocol is that once a lower priority task is executing its critical section, it cannot be preempted. So, the only thing that can be done is to make it execute as fast as possible. The shortest duration where which can release the critical section the higher priority task can be get to use the resource and the inversion time will be small, but how do we really make a low priority task complete its execution as early as possible.

(Refer Slide Time: 20:40)



Priority Inheritance Protocol

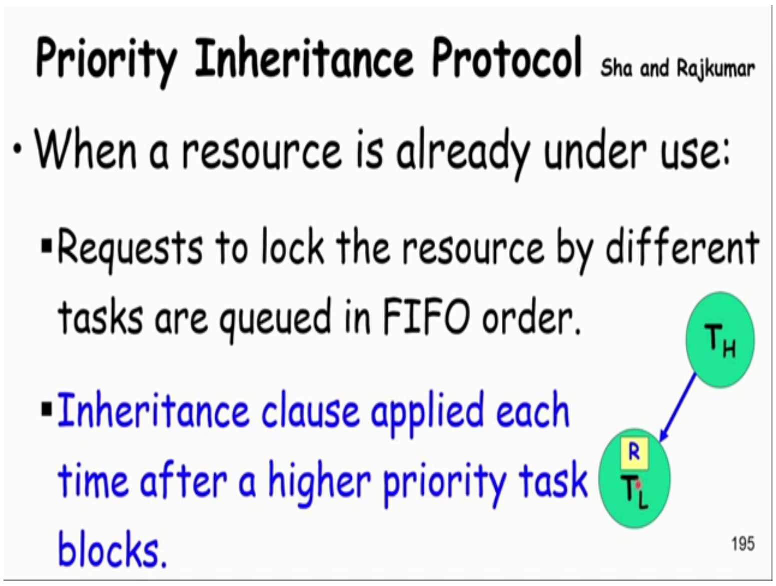
- How do you make a task complete as early as possible?
  - **Raise its priority, so that low priority tasks are not able to preempt it.**
- By how much should its priority be raised?
  - **Make its priority as much as that of the task it is blocking.**

194

Given that the protocol is simple, the main idea is that whenever a low priority task is executing make it complete, as early as possible. If you remember a low priority task which was executing in the critical section was getting delayed by intermediate priority tasks which was started to use the CPU, and the low priority task could not make progress. So, the solution here is to raise the priority of the low priority task so, that the intermediate priority task cannot preempted.

But then by how much to raise the priority of the low priority task executing its critical section, let us try to answer that question by how much should the priority of a low priority task once it is executing its critical section will be raised. The answer is that make the priority of the low priority task as much as the task that is waiting for the resource. And therefore, intermediate priority task cannot preempt this low priority task any more.

(Refer Slide Time: 22:13)



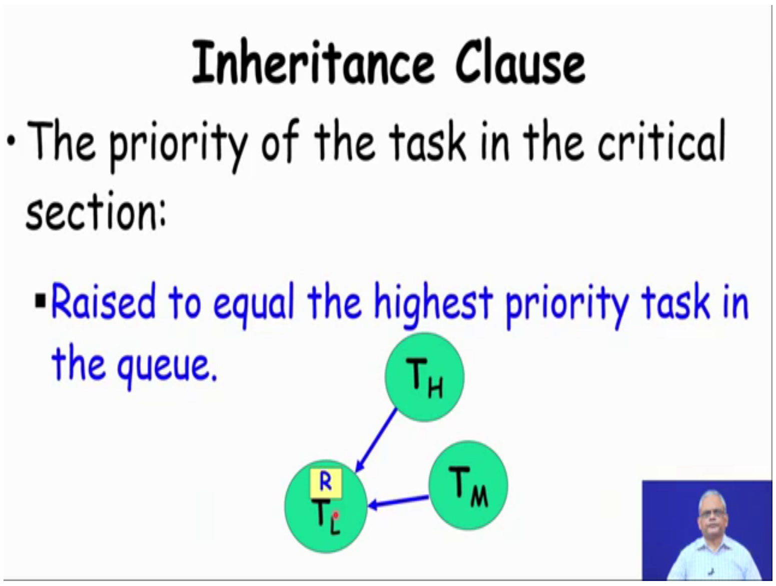
Priority Inheritance Protocol Sha and Rajkumar

- When a resource is already under use:
  - Requests to lock the resource by different tasks are queued in FIFO order.
  - Inheritance clause applied each time after a higher priority task blocks.

In this basic priority inheritance protocol when a resource is already under use. A high priority task that needs the resource makes the request and all the resources sorry all the request are queued in the FIFO order. And the inheritance clothes is applied to each time a high priority task requests a resource, and is waiting the low priority tasks priority is made equal to the highest priority task, that is waiting for the resource that is the simple protocol whenever a higher priority task, blocks for the resource.

The lower priority task inherits the priority of the highest priority task that is blocking on the resource. So, this is an example here, low priority task holding the resource high priority task waiting for the low priority task to complete uses of the resource, and the priority of the low priority task is raised to be equal to the high priority task. So, that the intermediate priority task cannot preempt the low priority task and this is called as the priority inheritance scheme.

(Refer Slide Time: 23:46)

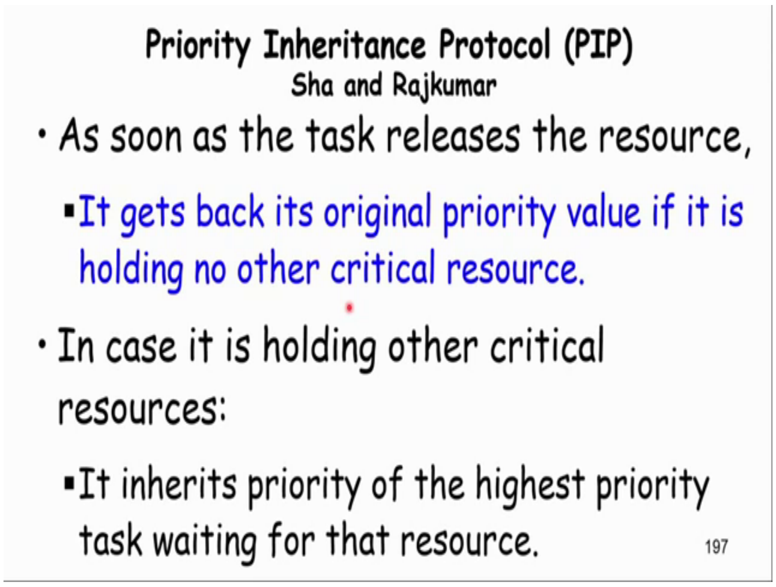


Inheritance Clause

- The priority of the task in the critical section:
  - Raised to equal the highest priority task in the queue.

Let us look at some more details of the scheme the nifty gritty of the scheme here, once the high priority task blocks for the resource, the low priority task priority gets raised to the highest priority task in the queue. If there are 2 tasks which are waiting 1 is medium priority, 1 is high priority for the resource then the lowest the priority task that is executing using the resource gets the priority of the highest of these 2 so, which is H.

(Refer Slide Time: 24:33)



Priority Inheritance Protocol (PIP)

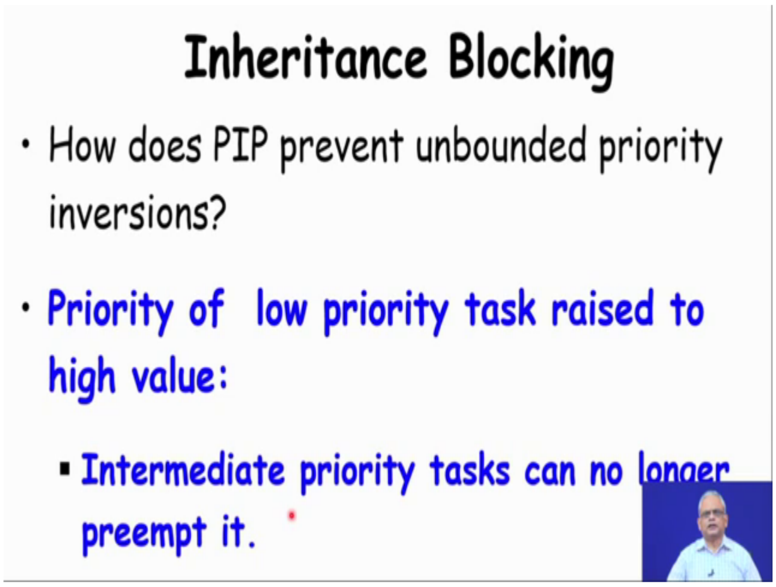
Sha and Rajkumar

- As soon as the task releases the resource,
  - It gets back its original priority value if it is holding no other critical resource.
- In case it is holding other critical resources:
  - It inherits priority of the highest priority task waiting for that resource.

197

But then once the low priority task completes its uses of the resource, it gets back to its older priority the low priority task again becomes low priority, its priority value changes once it release the resource unless it is holding other critical resources. Critical section and high priority task is waiting, it inherits the priority of the high priority task that is waiting and as soon as it releases the resource, it gets back its original priority.

(Refer Slide Time: 25:15)

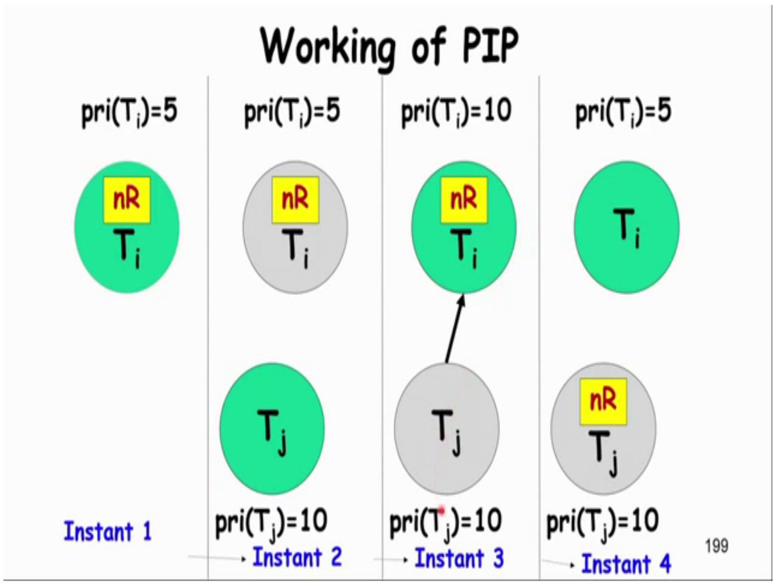


Inheritance Blocking

- How does PIP prevent unbounded priority inversions?
- Priority of low priority task raised to high value:
  - Intermediate priority tasks can no longer preempt it.

But then let us look at the problem it creates, and the problem it solves. It solve the problem of unbounded priority inversion, but how does it solve unbounded priority inversion problem it solved the unbounded priority inversion problem, because in the inheritance clause. The priority of the low priority task is raised to the priority of the high task that is waiting and therefore, the intermediate priority tasks that were preempting the low priority task from CPU uses cannot do so and therefore, the unbounded priority problem is solved. The intermediate priority tasks can no longer preempted and the unbounded priority problem is solved.

(Refer Slide Time: 26:24)



This is just a representation schematic representation of the walking of the priority inheritance protocol. Let us look at it a low priority task is holding a critical resource, and the priority value is 5 that is a time instance 1, after sometime that is time instance 2. A high priority task started executing and its priority level is 10. After sometime it needed the resource and it blocked, because the resource is already being held by the low priority task.

Started executing shown by green, but then its priority changed, because high priority task is waiting the priority changed from 10 to 5 due to the inheritance clause sorry, once it starts waiting its priority changes from 5 to 10, and it keeps n executing as high priority task with priority 10, and then as it releases resource it gets back its on priority value. We are at the end of this lecture we will continue from this point in the next lecture.

Thank you.