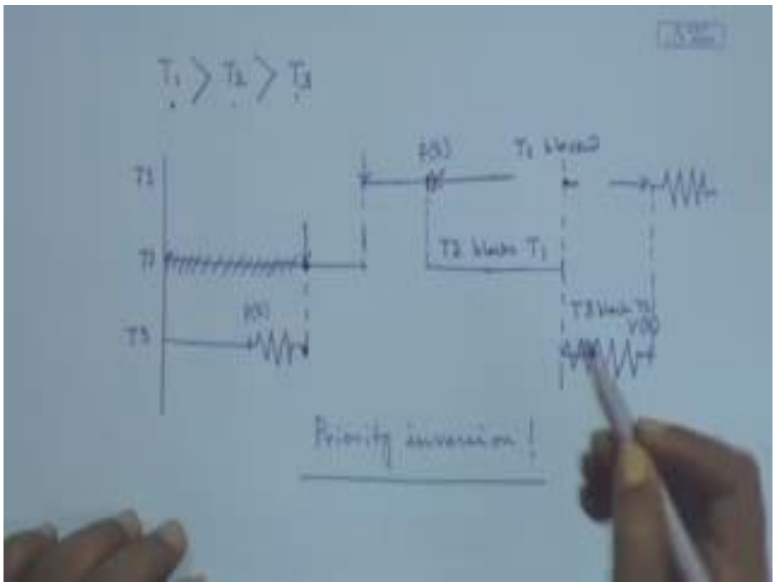
Embedded Systems Design

Prof. Anupam Basu
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 30
Priority Inversion and Priority Inheritance Protocol

(Refer Slide Time: 00:32)



Good morning. In the last class, we were discussing about a phenomenon called priority inversion. We were showing a scenario where there are three tasks say T 1 having a higher priority than T 2 and T 2 having a higher priority than T 3. And, the scenario is this that T 3 starts first, and then performs a critical section entry at this point. So, suppose this is the critical section in which P s; I mean in which T 3 is working on.

Now, at some point here while this is working, T 2 comes in. So, this one is preempted. So, T 2 starts from this point, T 2 arrives here. So, T 2 starts from this point and goes for a while. When we find the T 1 is arriving; T 1 arrives at this point. And while T 1 is arriving, T 1 makes a request for the same resource that is held by T 3, which is a lower priority task. And, so, this one actually cannot proceed any further. It is blocked here. It is not being able to proceed.

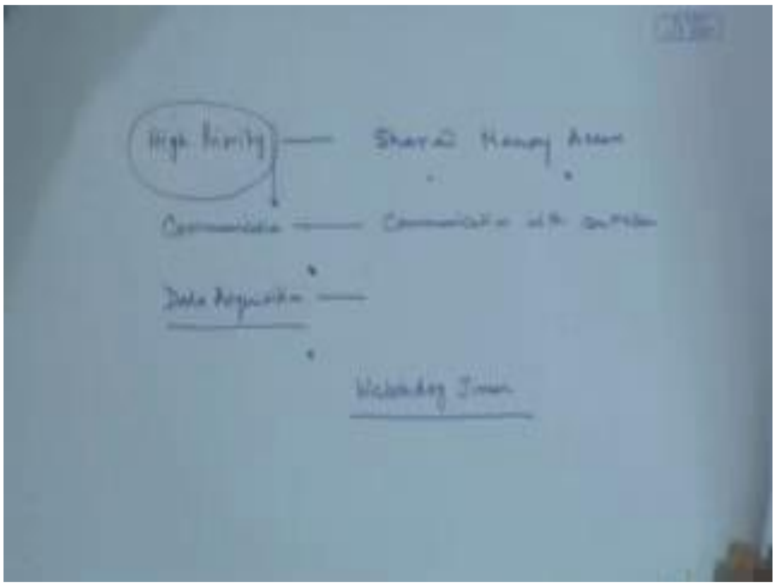
So, who will continue? Now this one, therefore, then the priority will come down to the next one; that means T 2 will continue for quite some time. And when T 2; because T 2 is having the priority next to T 1, so when T 1 halts, then T 2 will continue. And when T

2 ends here, suppose T 2 ends here, then also T 1 cannot resume although it is higher priority; because the critical section is already entered by T 3. Therefore, T 3, as long as T 3 is not doing, not releasing the resource, T 1 cannot get in. So, T 3 starts from here. Sorry, the T 3 actually executes the critical section from here. And, when it ends the critical section, then only T 1 which was waiting for. So, here T 3 performs the release through the V s signal. Then only from here, this one can enter its critical section.

So, you see, during so much time you can see T 1 was blocked. And, T 1 was mostly blocked by tasks having lower priority. So, here this is T 2 was blocking T 1 and here T 3 blocks T 1. Therefore, although T 1 was having priority higher than T 2 and T 3; because of this scenario, the priority has been inverted T 3 was having in higher priority than T 1. And, similarly T 2 was having higher priority than T 1. So, this is known as priority inversion.

On intentionally; it was not intentional, not intended. But, the priority got changed. So, one anecdote that we were discussing was that of the Mars; the pathfinder that went to Mars.

(Refer Slide Time: 05:26)



High Priority — Shared Memory Access

Communication — Communication with [illegible]

Data Acquisition —

Watchdog Timer

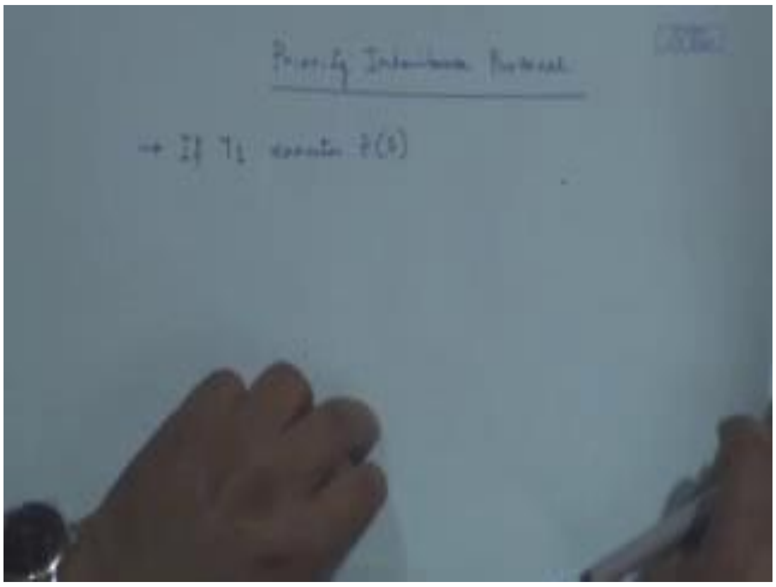
And, there I was talking about that there were three tasks. One was the high priority task of some communication or shared memory access, say. This was being done by the different components of the Mars system. The second one was the communication. I am not too sure; I think it was communication with the Earth station or something like that.

May be with some controller and, the third was data acquisition from Mars. So, these were the three tasks.

Now, it was somehow happening that it was found that there is a system reset that is taking place at different point; I mean intermittently. The system is getting reset, so all the data that was acquired is being lost and all those. So, ultimately what was found is that while the data acquisition was going on somehow just like this, if we assume just like this, the data acquisition was going on. And, during that time the higher priority task because of some interrupt, some interrupt came and the communication tasks started.

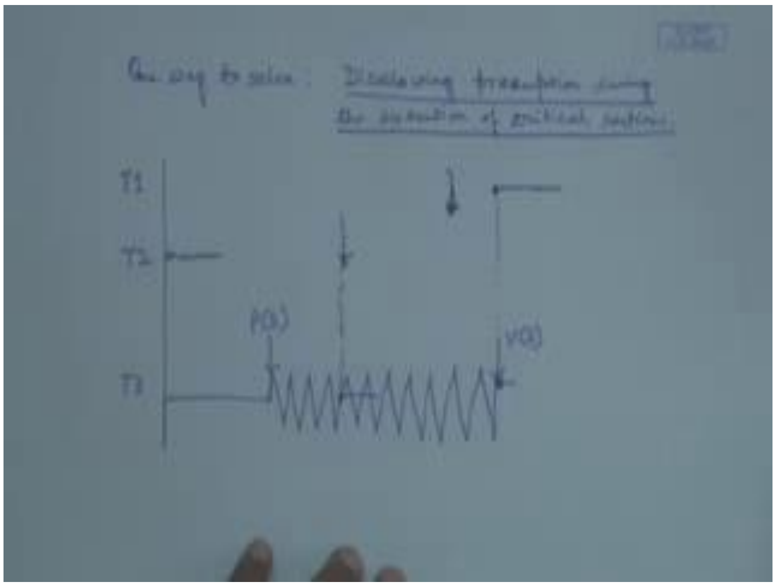
In this diagram, I have drawn it a little. This part you should not consider. The T 2 starts from here, T 2 starts from here, I inadvertently had drawn this line. So, this line is not there. T 2 just started from here. So, somehow there was an interrupt (Refer Time: 07:54). And, consequently the higher priority task that was on; highest priority task, that is, a shared resource was on was being delayed for a long time than that is expected. Since, this task is being delayed beyond the time that is expected, there was a watchdog timer. And, you know what watchdog timers do. That keep on watching for some regular events and when that does not happen that raise an alarm signal. So, this watchdog timer found that this high priority task is not being active for a long time. So, there must be something wrong. And, it was resetting the system. And, therefore because of this resetting which was not intended by design, the data acquisition was being affected. Time and again the data loss was occurring.

(Refer Slide Time: 09:19)


Priority Inheritance Protocol
→ If T1 executes P(S)

So, what is the; and this whole thing was happening because of the priority inversion phenomenon. So, what is the way out of priority inversion? The way out from priority inversion is the priority inheritance protocol. And, what is that? One option, even before that I can solve this problem by disallowing interrupts. Say, I wait for a second for this before I give the priority inheritance protocol.

(Refer Slide Time: 10:15)


One way to solve: Disallowing preemption during the execution of critical sections
T1
T2
T3
P(S)
V(S)

.

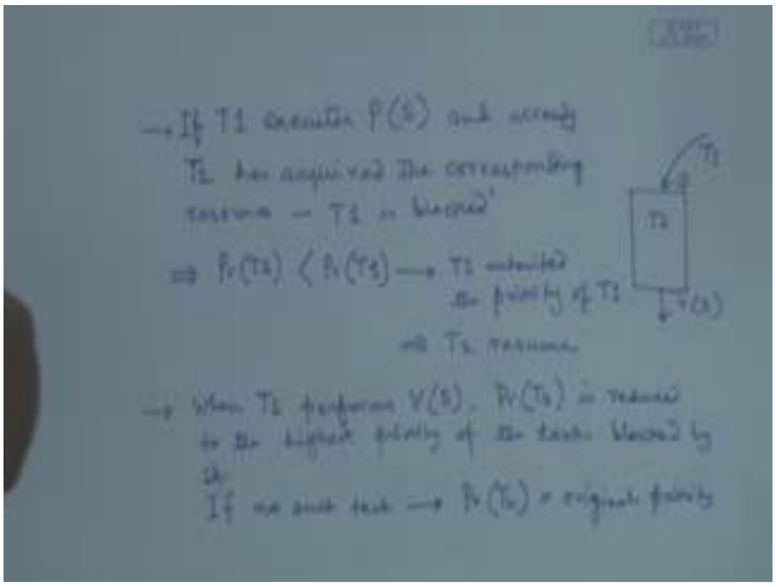
I can solve this. One way to solve this, the priority inversion problem would be by disallowing preemption during the operation, during the execution of critical sections. So, as a result what can happen if I just skipped this case in view? So, T 3 starts and makes a request to enter a critical section here. So, let us do that. T 3 starts from this point, makes a P s here, P s request here and enters the critical section.

Now, earlier what happened is T 2 arrived at some intermediate point. But, here what I am saying is the P 2, say, P 2 has arrived here. Sorry, T 2 has arrived here. Again, I am drawing this by inertia. So, T 2 has arrived at some point here. T 2 has arrived. But, I am not allowing this to start. I am not allowing this to preempt T 2 and T 3. And, instead I am carrying on T 3 till T 3 completes and performs a V s at this point. Now, at this point may be T 1 has again arrived. T 1 has arrived at some point like this. Now, obviously T 1 has got higher priority than T 2. So, when it performs the V s, at this point T 1 will resume.

T 1 will resume here. In that way, I can reduce the extent to which T 1 was delayed earlier are blocked because T 1 has arrived here. And, we expect this critical section to be not that big. But, one problem with this that since I am now here, say T 1. This T 1 was not actually asking for s, might be. It was not interested in the shared resource, but because of my principle that I would not allow preemption during the execution of the critical section, this was blocked. Although, it was not interested in the resource, so that is one disadvantage of taking the measure of disallowing the interrupts.

Now, next I come to the priority inversion protocol. Sorry, priority inheritance protocol. Here, the rules are. Please, note carefully that the tasks were scheduled according to their priorities. If task T 1 executes say P s; that means, when I write if task T 1.

(Refer Slide Time: 15:00)



→ If T1 executes P(S) and already T2 has acquired the corresponding resource — T1 is blocked

⇒ Pr(T2) < Pr(T1) → T2 inherited the priority of T1

⇒ T2 resumes

→ When T2 performs V(S), Pr(T2) is reduced to the highest priority of the tasks blocked by it.

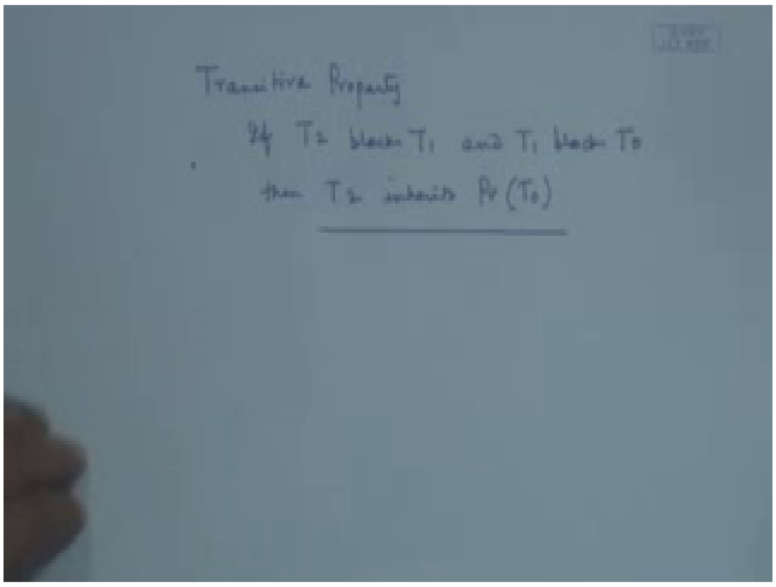
If no such task → Pr(T2) = original priority

Let me write it clearly. This is not being very clear. When I write if T 1 executes P s, what does it mean? That there is some resource, shared resource, which is being accessed by T 1 through the semaphore s; through this gate s. So, if T 1 executes P s and already say exclusive access; if already T 2 has acquired the corresponding resource, T 1 is blocked. That means if T 1 is trying, but before that already mister T 2 is already here is executing this. Therefore, T 1 is blocked.

Now, if here priority of T 2 is less than priority of T 1, if that is the case, then if priority of T 2 is less than priority of T 1, then actually what happened in this case? T 2 has inherited the priority of T 1. Is not it because T 1 is having higher priority than T 2. But since T 2 is here, T 1 is being blocked. Therefore, T 2 is inheriting the priority of T 1. So T 2; now according to our preemption, according to our preemptive policy, T 2 was being preempted. But then, we will stop a while and see that T 2 already acquired s. Therefore, let T 2 inherit the priority of T 1 and T 2 resumes; implies the T 2 will resume. And, T 1 will remain blocked. So, the rule is that the tasks that inherit the highest priority of the tasks that are blocked by it. So, T 2 is blocking; the highest priority task the T 2 is blocking is T 1. Therefore, T 2 gets the priority of T 1.

Now, when T 2 performs V s; that means, T 2 is coming out from here by doing V s. T 2 is coming out. Priority of T 2 is reduced or decreased to the highest priority of the tasks blocked by it. If no other tasks are blocked by T 2, then if no task, no such task is there, then priority of T 2 is the original priority, the original priority of T 2. Now, if T 2 blocks T 1 and T 1 blocks T 0, then T 2 inherits the priority of T 0. Suppose now, so is it clear? Will give an example; that will be clear.

(Refer Slide Time: 20:26)



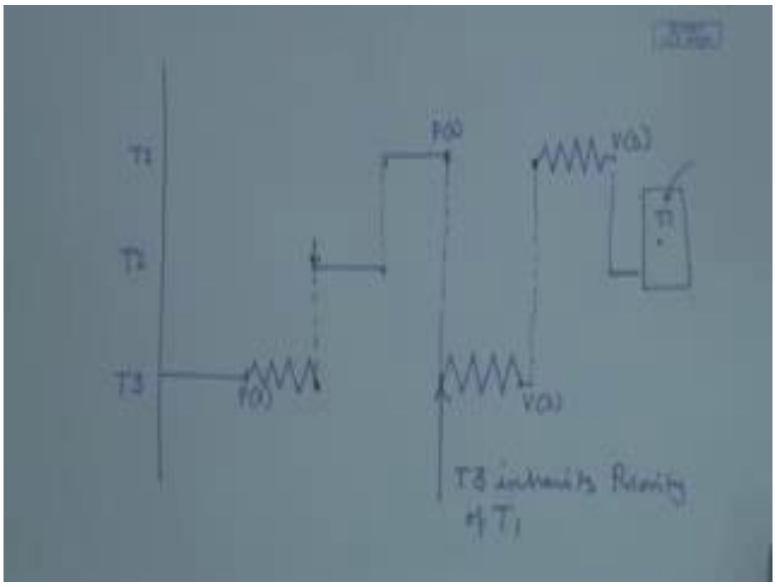
Transitive Property

If T_2 blocks T_1 and T_1 blocks T_0

then T_2 inherits $Pr(T_0)$

Now, it is this property is transitive. In the sense, this inheritance is the transitive property because if T 2 blocks T 1 and T 1 blocks T 0, then T 2 will inherit priority of T 0 because T 0 was the highest priority task. Now with this, we will come back to this with an example and see how it works. Say, we come back to our earlier example.

(Refer Slide Time: 21:28)



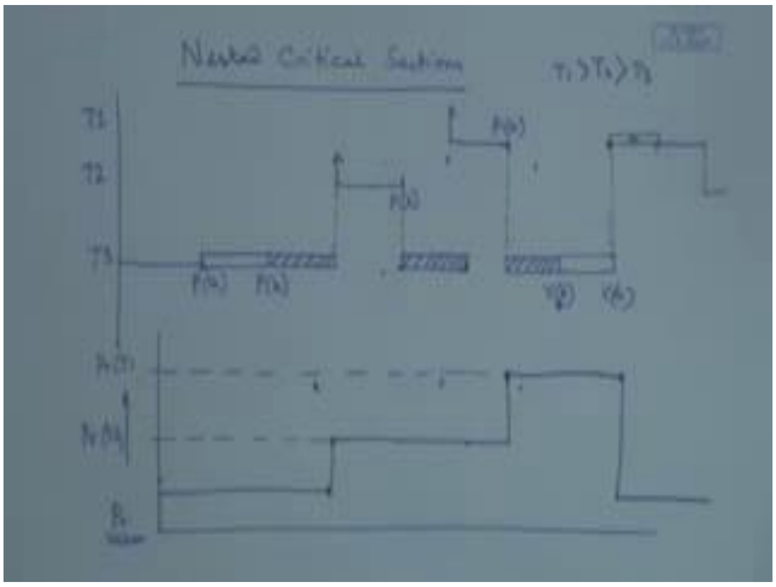
So, I have got T 3 starting here. And, after while it performs P s. So, some shared resource is here. That has been acquired by T 3, T 3 has performed P s and has got in. So, at this point T 3 is performing the critical section. Now, at this point T 2 arrives. So according to normal preemption, there is no conflict over the critical section. So, T 2 will start. T 2 is started. Now, at this point T 1 arrives, does something and executes a P s. Now what happens? This goes to sleep.

Now, the question comes. If T 1 executes P s and already T 2 has acquired the corresponding resource, T 1 is blocked and T 2 inherits the priority of T 1. Now what will happen? Who is? Which task is blocking? Yes, T 3 therefore, T 3 will inherit the priority of this and will start executing its; resume executing its critical section from here. So, at this point T 3 inherits priority of T 1.

So, now it completes, performs a V s here. And then, what will happen? Immediately this one, T 1, will get back its priority because T 3 actually have been pushed out T 1 because of inheriting the higher priority. So, this one will resume. Does the critical, it was waiting for the critical section and does something and does V s here. And, after that T 2 will continue. T 2 will then continue. T 2 was waiting here because T 2 preempted T 3. You see T 2 is preempting T 3. But, something happened here because T 1, again T 1 preempted T 2. Now T 1 is contending for a resource, which is already grabbed by T 3. Therefore, T 3 is inheriting the priority of T 1 and completing and releasing the resource first. Releasing the resource, look at it from this angle. It is releasing the resource to the highest priority task that is asking for it. I know that you are a highest priority person, so let me finish off my task and quickly hand it over to you. That is what is happening here. And therefore, this highest priority task is being delayed less (Refer Time: 25:47) completing and this one is continuing. So by this, this is the priority inheritance protocol.

Now, what happened in the Mars story is somehow the vent was sent to Mars. The priority inheritance bit was set to zero. So, priority inheritance was not working. So, priority inversion was taking place. So, from the earth station they reset. They set the bit to one, and after that the problem was solved. So, you see that all these thing that we are discussing are not merely academic. They are real life and very costly consequences.

(Refer Slide Time: 26:40)



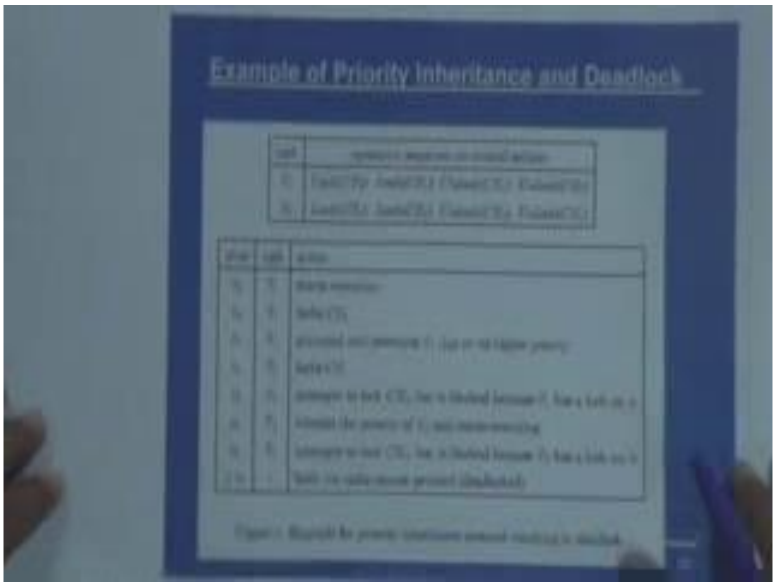
Now, we can have some nested; let us look at nesting of the critical sections. So, you will see here I am depicting a situation where T 3 starts. And, here T 3 requests for some resource a. Starts accessing a. At this point, it is asking for another resource b. It gets b for a while. But, please note that it has not released a; nesting. While it was in a critical section, it is asking for another critical section. And at this point, T 2, at this point T 2 arrives. And, T 2 starts execution. And, here T 2 asks for b, but b cannot be granted here because it is captured by this. Therefore, at this point what will happen? A priority inheritance will take place. How? T 2 is of higher priority than T 3, although I did not mention it in this example.

So, the priorities are like this; T 1 greater than T 2 is greater than T 3. So, this is having the higher priority than this. But, this will inherit. This will inherit that priority and complete T 3 first. T 3, although this was having higher priority, it was asked to wait and let it complete by inheritance. And here while this was being executed, T 1 appears here. T 1 appears somewhere here. Now, T 1 again executes and asks for P a. Now, what will happen? Now, it has not yet completed. But, here there is a request for P a. So, who will have the highest priority now? T 3 only. T 3 will have the highest priority. So, T 3 will first complete this part, b part, then will complete the a part. So after this, at this point it does V b, but although it does V b here, this cannot preempt. Why not? Because the priority of T 3 is that of T 1. And, T 1 is of higher priority. I repeat. Although, it is V b here and T 2 was waiting for P, but still it cannot get this because by this time the priority of T 3 is that of T 1; because T 1 is having the higher priority than T 2. So, it completes; this a part does V a. And then, at this point T 1 will complete this a part. And then, whatever it does. And, when it completes this a part, when it completes the whole thing, then only T 2 can start.

So, if I just plot here, say what happens to the priorities? Say, I say priority values of T 3. We can see I should have aligned it with this. It is going on here, fine up to this and here it gets the priority of T 2. Right. Goes on up to this and here it gets the priority of T 1, more higher priority, goes on and then when it completes this V a, it will come back to its old priority. So, if I plot the priority values here, this is the priority value of T 1, this is the priority value of T 2. So, the priority value here in this diagram are merely showing the priority value of T 3, how it is changing because of priority inheritance.

So, there can be scenarios where this can also raise to deadlock. Although, this one is very nice to see that I can solve the priority inversion problem because of using priority inheritance, but this is also not without side effects. Let us take one example.

(Refer Slide Time: 32:55)



Example of Priority Inheritance and Deadlock

task	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]

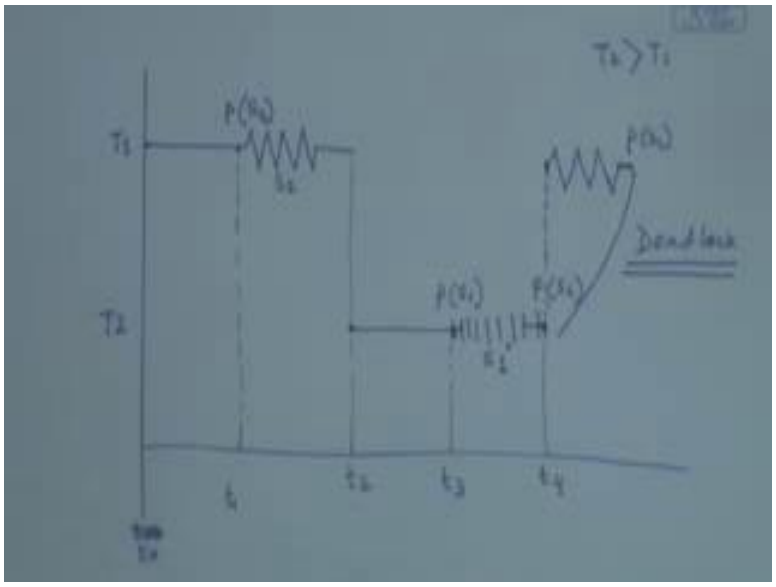
time	task	action
[illegible]	[illegible]	starts execution
[illegible]	[illegible]	locks [illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	locks [illegible]
[illegible]	[illegible]	attempts to lock [illegible], but is blocked because [illegible] has a lock on it
[illegible]	[illegible]	inherits the priority of [illegible] and resumes executing
[illegible]	[illegible]	attempts to lock [illegible], but is blocked because [illegible] has a lock on it
[illegible]	[illegible]	[illegible] (deadlock)

[illegible]

Let us take this example. I do not know whether it is very visible or not? Visible? So, here what is shown is at time T 0, T 1 is starting execution. At T 1, T 1 is locking some critical section. Now in this example, priority of T 2 is higher than that of T 1. Let us assume that. If you like you may take it down. T 0 is starting the execution, then T 1, at T 1, I am sorry, at T 0 the task T 1 is starting execution; the low priority task. Then, it is locking on some critical section. At that time, the higher priority task is activated and preempts; the higher priority.

Now this, there are two critical sections here. You can see that this T 2 is locking this C s 1. Let me draw this. I think that will be easier for you to understand.

(Refer Slide Time: 34:14)



See, initially now here note that priority of T 2 is greater than priority of T 1. So, initially at this is T zero, from here T 2 starts. At time T 1, I am sorry this T 1 has started. Let me redraw it.

Priority of T 2 is greater than priority of T 1. So, this is the time line t equal to zero. Initially, task T 1 starts execution; normal execution up to time T 1. And, here it performs P s 2. So, there are two critical sections here; P s 2. And, nothing else has happened. So, this is executing the critical section. Which critical section? S 2. On S 2, it is working on the resource. So, they are synonymous.

Now at this point T 2, higher priority task T 2, at time t 2, higher priority task T 2 starts. So, it is activated. And obviously when it is activated, it is preempting the lower priority task T 1 (Refer Time: 36:31). Now, here at this point at time t 3, at time t 3, it asks for P, another resource S 1. Now, it performs locks s 1. After that, immediately after that say at t 4, so it locks S 1. And, let me give some other sign. This is the sign when it is working on s 1. And at this point at t 4, it requests s 2. Now, what will happen? It requests s 2.

Now by priority inheritance, which one will get the priority? Because here it has, it is requesting for the same thing. So, T 1 will now get the priority of this. Now at t 4, therefore T 1 has a lock on it. So at t 5, so it has got made; this one is has started executing again. It was doing the critical section. Now, at this point if it requests for P s 1, but it is blocked because T 2 has already got s 2. Now, it is asking for P s 1, but that is already; now if I already grabbed by T 2, now if the priority inheritance also occurs, then this also cannot proceed. Therefore, here it is a deadlock situation. So, deadlock is a side effect of priority inheritance protocol that cannot be completely avoided.

There is an algorithm called the priority ceiling algorithm, which I will do in the next class before I start moving to some other topic.