Embedded Systems – Design Verification and Test

Dr. Santosh Biswas
Prof. Jatindra Kumar Deka
Dr. Arnab Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

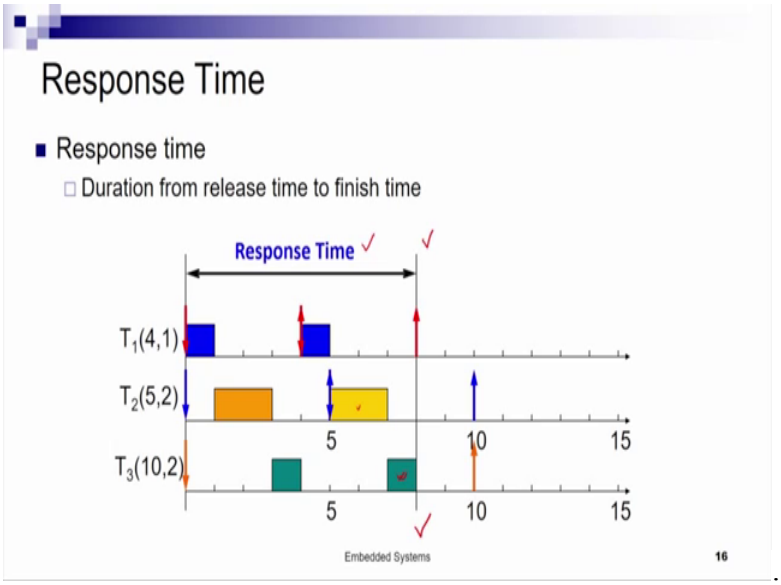
Lecture – 16
Real-time Uniprocessor Scheduling

You let us continue our discussion with the rate monotonic algorithm. We said that rate monotonic is an online static priority algorithm, in which priorities are decided by the periods of the tasks; shorter the periods of the tasks higher becomes its priority. And we said that to understand whether a set of tasks will be schedulable using the RM algorithm, we need to perform schedule ability analysis and schedule ability tests. We saw one such sufficiency based schedule ability test based on upper bound, which said that if the summation of utilization of the task set is less than n into 2 to the power 1 by n minus 1, then the task set will assure to be schedulable under RM in this n determines n is the number of tasks in the task set.

However this is a sufficiency based test and is a bit pessimistic; in the sense that if I have a task set for which this bound is met I am care it is guaranteed that RM will be able to schedule this task set. However, if this bound is not satisfied; that means, the summation of utilization is greater than n into 2 to the power 1 by n minus 1, it is still possible that the task set will may be schedulable by RM. So, therefore, it is not as necessary test.

So, we do not get both the necessary and sufficient conditions for schedulability of task set using that algorithm using that bound. However, why do we use that bound? Because the time complexity is just be go often, we just need to you just need to sum the utilizations of the tasks and then if when we have performed the summation of utilizations we can perform this bound based test. However, if this condition is violated we still need to understand more carefully whether the task set will be schedulable under RM. Therefore, we need to have a test which is also which is both and which provides both a necessary and sufficient condition for schedulability and this test is based on the response time of the tasks in the system. So, what do we mean by the response time of a task?

(Refer Slide Time: 03:02)



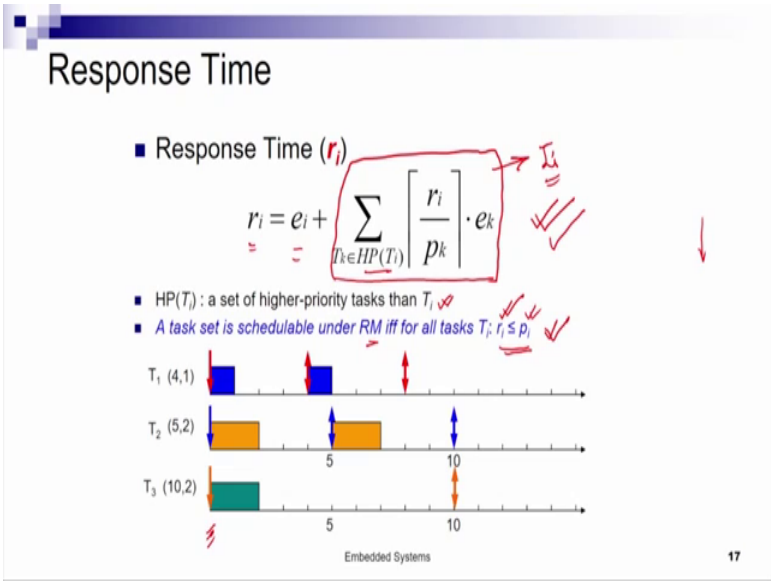
The response time of a task is a duration between its release time and its finish time. For example, for the task T 3 here if you want to further the response time of the first job of T 3 is given as follows can be seen as follows.

Now, if you see the order of executions what will happen? T 1 has the highest priority because it has the shortest period, its execution time is 1 period 4, T 2 has an execution time of 2 period 5, T 3 has an execution time of 2 period 10. So, the highest priority is T 1 because it has the shortest period so, it will get a chance to execute first. So, T 1 execute first and after T 1 executes T 2 will get a chance to execute because it has the second the second shortest period length and only after both T 1 and T 2 have got a chance to to execute their jobs T 3 can get a chance to execute.

Now, if we see the execution order under RM it will be something like this, T 1 execute T 2 execute and then the first part of T 3 execute then the first; then T 1 execute again and after T 1 execute T 2 executes at 5 completes its job and finally, the last part of T 3 execute and completes execution at time 8. So, the response time of the first job of T 3 is at time 8

Now, what is the mathematical what is the mathematical expression that can be used to determine what the response time of a task will be?

(Refer Slide Time: 04:52)

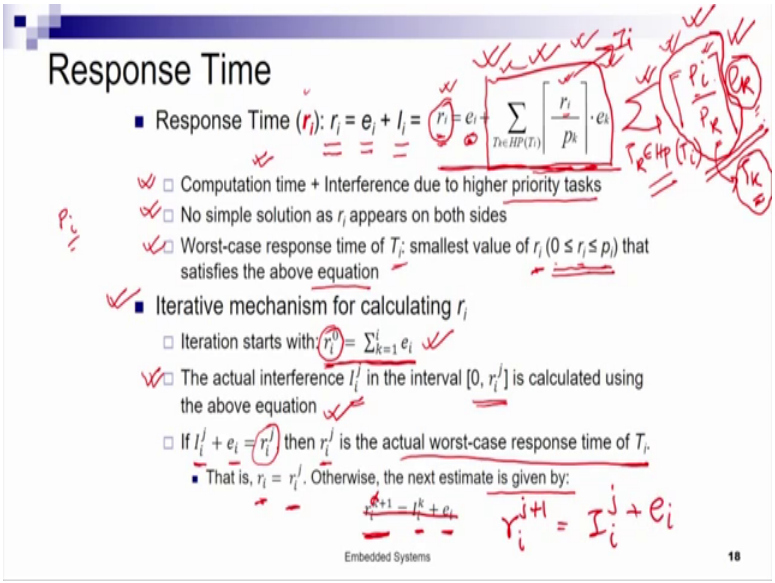


It is given by this equation here. So, what does this equation tell? It says that is that the response time of a task is equal to its execution time plus this part. This part is called the interference created by higher priority tasks. So, interference I i for tasks I it is the interference created by higher priority tasks. So, where H here HP i is the interference created by the set of higher priority tasks, which have higher priority than T i we will look at this in a bit more detail. And we say that if a task set is schedulable under RM if and only if for all tasks its response time is less than its period, when all tasks start at their critical instance; that means that everybody all tasks start at the same time.

So, tasks in the system can start at different times; however, if that all tasks start at the same time, we call that task set has started at the critical instance. And it can be proved that if a task set is schedulable when all tasks start at the critical instant, then for all other instances the tasks will always be schedulable ok.

So, then this condition this condition for schedulability, says that a task set will be schedulable under RM if and only if for all tasks the response time of their first jobs is less than the period of the task when all these tasks start at the critical instance. So, therefore, we now need to understand deeper in deeply more deeply what response time is and how such a response time can be determined.

(Refer Slide Time: 06:49)



So, as we said the response time has 2 parts. The response time of the ith task is given by the execution time of the task itself plus the interference created by higher priority tasks. If you look at RM we will see that during any period of time, let us say the execution time has to complete before its period P i. So, e i is his execution time and P i is its period. So, within this period P i it has to execute itself it has to complete its own execution e i, and we said that whenever a task with a shorter period comes, I must allow it to execute. So, I must allow the execution of all instances of higher priority or shorter period tasks, that come within P i ok.

So, in this expression if we replace if this is the if I we said this is the response time I i this is I i right this is I i. In this I i if we replace this one T k belongs to H p of T i if we replace this by P i and then P k into e k what does this tell us? This part of the expression this part of the expression tells that how many times a higher priority task T k. So, for this task T k P k is its period and e k will be the execution time. So, this part tells how many times a higher priority task gets executed within the period of T i. So, T i has a period of P i T k is somebody which is which has a higher priority than P i and hence has a shorter period.

So, within this period of P i t T k is going to come multiple times how many times? At most ceiling of P i by P k times. This many times T k will come for execution within the period P i within the time period P i and each time it comes it will take an execution time

of e k. So, e k is the execution time of T k, each time it comes for execution it will consume an execution time e k within this period P i. So, it comes for P i by P k ceiling times and each time it comes it consumes e k. So, therefore, this total time is given by P i by P k into e k and this is the interference that is caused by task T k.

Now, we have to add up the interference is caused by all higher priority tasks. So, therefore, we sum up we obtain this sum we sum up for all higher priority tasks T k, we find that and we add that up. This is the total interference caused by higher priority tasks. So, why did not we use this expression what is the difference between this expression and this expression? Here this is r i and here this is P i what is the meaning? So, P i is the.

So, if we use this bound instead of this bound this is also a sufficiency based bound, it is a good bound, but it is a sufficiency based bound meaning that at the time the task T i may not complete exactly at the end of P i. So, it may not complete exactly at P i it may complete at a time which is earlier than P I, then the inter then the response time before its response time we need to calculate the response we need to calculate the interference caused by the higher priority task before the response time of P i comes. We do not need to calculate the interference cost caused by higher priority task after the completion of T i itself after the completion of the first job of P i itself.

Hence this bound here is a sufficiency based bound, it is not the actual necessary bound; the actual necessary bound is given by this one. So, what does this calculate? It says that the response time of the task of our task of the ith task T i is given by the summation of the execution time of T i plus the interference caused by higher priority tasks within its response time not within its period, but within its response time and hence we have this equation. But the problem with this bound is that this r i appears both on the LHS and RHS which makes it complicated to calculate the value of this response time ok.

So, let us see how do we calculate the response time? So, we see that the response time is the summation of the computation time plus the interference caused due to higher priority tasks. There is no simple solution as r i appears on both sides. The worst case response time of T i however, is obtained by the smallest value of r i that satisfies the above equation. So, the catch is to be able to find out the smallest value of r i that satisfies this above equation and that is the worst case response time.

So, how do we calculate? The smallest value of r i; obviously, r i will be something which is within this interval. R i will be greater than 0 and is less than equal to P i beyond P i we need do not need to calculate anything because the period is violated period or deadline is violated, we are considering here implicit deadline tasks where deadlines of the task is equal to the length it is equal to its period ok.

So, r i is something which is between 0 and P i and we need to find out the smallest value of r i within this interval. How do we calculate the value of r i through an iterative mechanism? Let us understand this iterative mechanism now. The iteration starts with r i 0 equals to summation k equals to 1 to i e i. Here r i 0 is the value of the response estimate of the response time at the 0 with iteration. So, r i 1 will be the value of the estimate of the response time at the first iteration, r i 2 will be the estimate of the response time at the second iteration and so on.

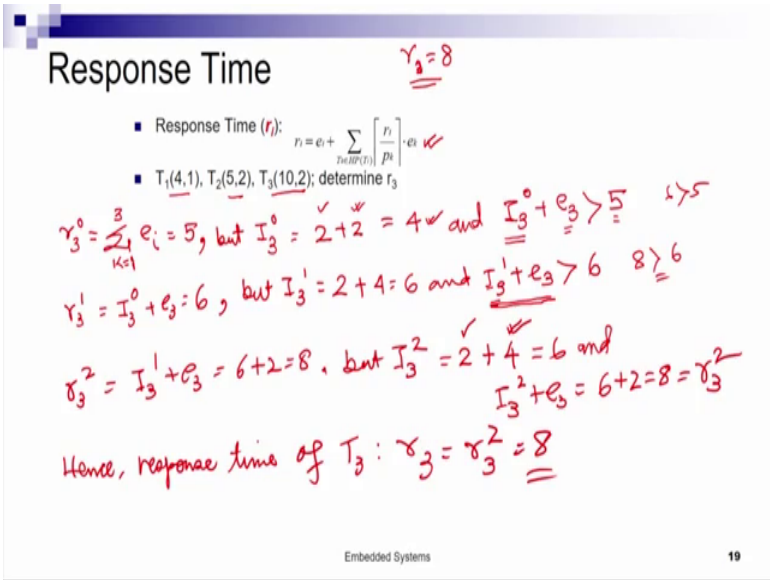
Similarly, we will have I i 0 which is the estimate of the interference cost at the 0 at iteration which then r i 1 is the estimate of sorry I i 1 is the estimate of the interference caused at the at iteration 1. I i 2 is the estimate of the interference caused at iteration 2 and so on.

So, what do we start with? We start with this we start with r i 0 equals to summation k equals to 1 to i e i ok. One assumption what we have made here is that higher priority tasks have a lower index. So, k equals to 1 to i, k equals to 1 to i minus 1 are the higher priority tasks than T i. So, T 1 T 2 T 3 up to T i minus 1 have a higher priority than T i we want to find here the response time of T i and what do we start with at least all the higher priority tasks will execute at least once and the task itself has to execute once.

So, the response time by which the output will be produced. When will the output will be produced? I have to allow at least one execution it will be more than that possibly, but at least one execution will always be there of all the higher priority task, and the task itself also must execute. So, we will start first with the response time with the response time estimate as summation k equals to 1 to 1 e i. Then the actual interference between these between this for in this in this interval r i 0 will be calculated ok. So, then we will calculate I i 0 which is the rest which is the interference caused within the response time I i 0 estimated response time r i 0.

Now, the actual interference I i j i in the interval 0 r j i is calculated using the above equation ok. So, for r i 0 we estimate the interference I i 0, for r i j we estimate the interference I i j. Now when do we stop if I i j plus e i the execution time so, the total actual interference plus the execution time of the task itself if that becomes equal to the response time, then we say that I r i r j i is the actual worst case response time of T i, that is r i equals to r j i. Otherwise we go to the next iteration and the next estimate of the response time becomes r k plus 1 i equals to r i k plus I i sorry not k plus 1 this is j r using a response time analysis method we just saw let us calculate the response time of the first job of T 3 here.

(Refer Slide Time: 17:45)



We saw this example of 2 slides back and we know that the response time r 3 of T 3 is actually equal to 8. Let us now see how this value 8 is obtained. So, we do this in iterations. So firstly, we have the first estimate of the response time r 0 which is equals to the summation k equals to 1 to 3 e i why k equals to 1 2 3 because T 3 is the lowest priority task and which is equals to 5. But I 3 0 for this becomes what equals to 2 plus 2 equals to 4; why? Because it is the summation this value of summation if you see what does it become for I 3 0 r i 5. So, 5 by 4 ceiling becomes 2, 2 into 1. So, therefore, the interference caused by T 1 is 2 and r i is 5. So 5 by p 2 that is again one ceiling 5 r i by r i by r 3 by p 2 that becomes 1 so, 1 into 2 equals to 2.

So, the interference caused by task T 2 is 2. So, the total interference caused by higher priority tasks is 4. However, what do we have I 3 0 plus e 3. So, this is the total is greater than 5 hence within this response time 5, I cannot have the; I cannot accommodate all the interferences of higher priority tasks and also the execution time of T 3. So, it is 4 plus 2 which is equals to 6 ok. So, 6 is greater than 5.

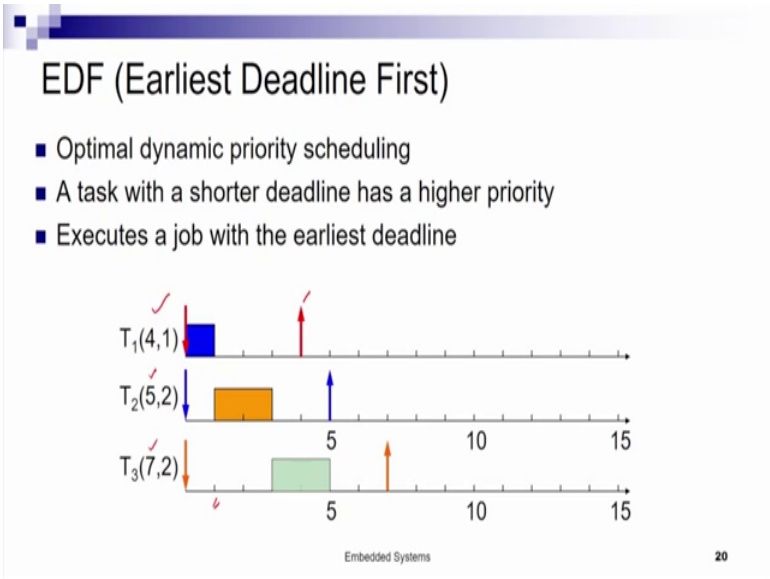
Now, we have the current. So, the next estimate of the next estimate r 3 1 becomes what equals to I 3 0 plus e 3 which is equals to 6, but what happens? I 3 1 becomes what equals to the interference caused by T 1 what is the interference caused by T 1? If it is 6 it is again 6 by 4 which is 2. So, 2 into 1 is 2 what is the interference now caused by the task? T 2 it is 6 by 5 which is 2 into 2 which is 4. So, the total interference caused is 6 and therefore, I 3 1 plus the execution time e 3 that becomes greater than 6. So, I 3 1 is 6 plus e 3. So, it is 8 so, 8 is greater than 6.

Now, so the total inter this is the total actual interference by higher priority tasks within the estimated response time 6, that is actually 6 and plus I add up the execution time of the task itself and I get the total time that is necessary. So, the response time it cannot be 6, response time has to be something higher than 6. Hence we move to the next iteration and we get r 3 2. So, r 3 2 equals to I 3 1 plus e 3 equals to 6 plus 2 equals to 8 ok, but I 3 2 I 3 2 equals to 2 plus 4 why because I 3 2 is r 3 2 is 8. So, 8 by 4 ceiling of 8 by 4 is 2, 2 into 1 is 2. So, interference caused my T 1 is 2, the interference caused by T 2 is 8, ceiling of 8 by 5 which is 2 into 2, 2 multiplied by 2 which is 4. So, the total interference caused by task T 1 plus and T 2 is equals to 6 and we have I 3 2 plus e 3 equals to 6 plus 2 equals to 8 equals to r 3 2.

Hence response time of T 3 becomes r 3 equals to r 3 2 equals to 8 hence we obtain the response time of r 3 as 8, this we already saw that this is actually. So, what happens when we ran the schedule on the natural Gantt chart that we saw. So, hence we saw that the response time of task 3 is 8, and we can see that this response time is less than its period which is 10. Similarly in order to conduct the schedule ability test, we also need to find out the response time of T 2 and T 1 and if their response times are also less than their deadlines, that is for all 3 task their response times are less than their respective deadlines, we can say that this task set will be feasibly schedulable by the by the RM algorithm.

However the time required to calculate this schedulability test, if pseudo polynomial in nature and hence it is costly ok. With this discussion of response time we now move to the next algorithm which is earliest deadline first.

(Refer Slide Time: 24:07)

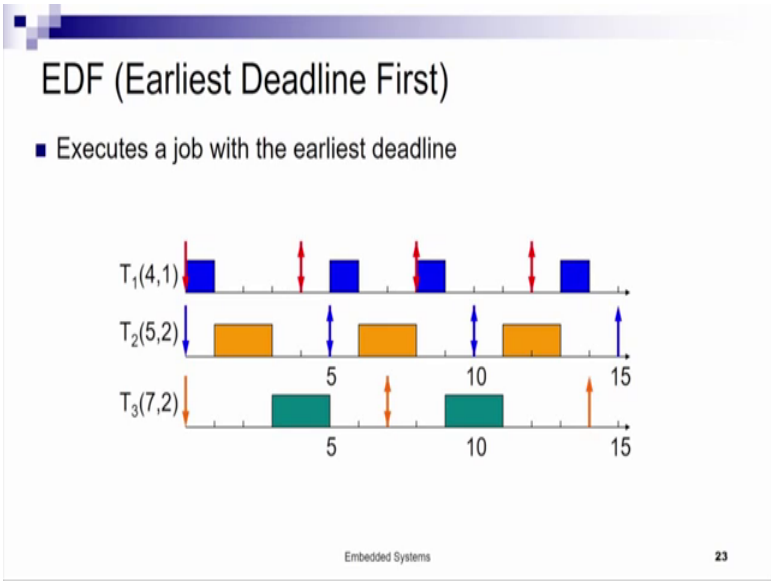


Earliest deadline first is an optimal dynamic priority scheduling algorithm. So, we have to see why it is optimal and what do we mean by being dynamic priority. It is an online algorithm; that means, the decision for the order of task execution is taken at runtime, it is a dynamic it is a job level dynamic priority algorithm; that means, the priority of the jobs can vary during execution and is determined during at runtime and what is the priority, how is the priority determined? A task with a shorter deadline has a higher priority.

Now, so, therefore, earliest deadline first at any time executes the ready job with the shortest deadline oh sorry with the earliest deadline first. So, in this task set we see that the deadline or period of the first task is 4 deadline or period of the second task is 5, deadline or period of the third task is 7.

Now, let us see for the jobs how the scheduling is conducted. So, T 1 first executes being the highest priority task execute first, now after T 1 completes at time one at this time the second job of T 1 will arrive only at 4 and so, the remaining jobs that are active are that of T 2 and T 3 and T 2 gets a chance to execute because it has the next earlier deadline T 3.

(Refer Slide Time: 26:04)



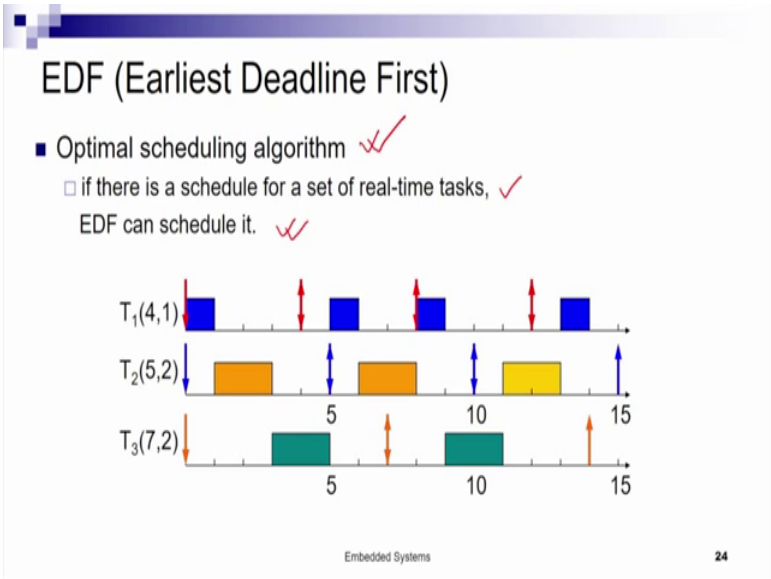
T 3 executes the first part of T 3 execute next executes next and we at this time the second job of T 1 arrives, but for the second job of T 1 the deadline is going to be 8, the first for the first job of T 1 the deadline is 4. So, this for the second job of T 1 the deadline is going to be 8 which is here, but the deadline for the first job of T 3 here is 7. So, the first job of T 3 has an earlier deadline than the than the deadline for the second job of T 1 and therefore, this job of T 3 now has a higher priority than the second job of T 1 now, let us see what happens.

So, therefore, although T 1 is there, but T 1 second job is there, but T 3 having the higher priority completes execution at 5. Now at 5 the second job of T 2 arrives for the second job of T 2 the deadline is going to be 10; for the first job of T 2 the deadline was 5. So, the sec for the second job of T 2 the deadline is going to be 10. Now become between T 2 and T 1 we see that for the second job of T 1 the deadline the deadline is 8 and for the second job of T 2 the deadline is 10.

So, the second job of T 1 has a higher priority here and hence this execute the second job of T 1 executes likewise the other jobs in the tasks are scheduled and if we see that for RM this particular task set became unschedulable, we could not schedule we had a deadline means at this position. That was because at this position if we see T 3 was became a lower priority task compared to T 1 at this point in time; however, for when the second job of T 1 right

However in earliest deadline first we saw that because the first job of T 3 had a deadline of 7, while the second job of T 1 had a deadline of 8, the first job of T 3 remained at a higher priority than the second job of T 1 and hence all the deadlines could be successfully satisfied by earliest deadline first, where RM could not satisfy them.

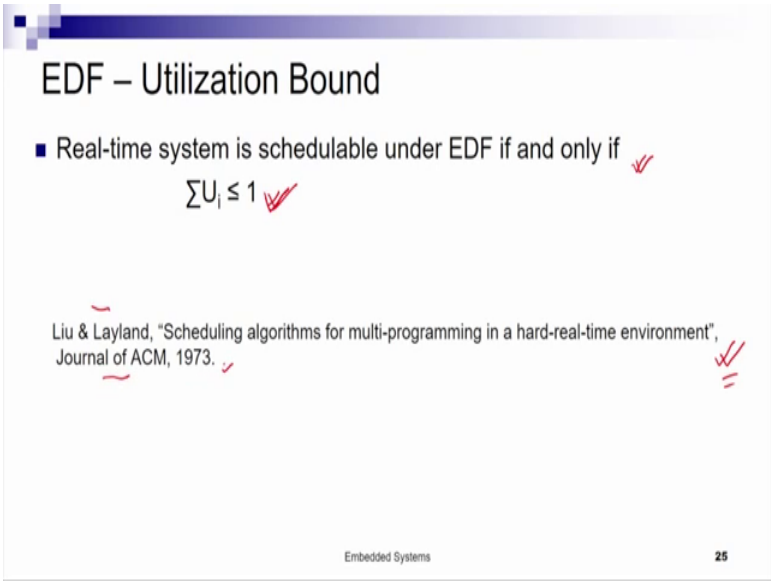
(Refer Slide Time: 28:51)



So, we say that earliest deadline first is an optimal scheduling algorithm, which means that if there is a schedule for a set of real time tasks EDF can schedule it. So, if there is any algorithm that can schedule a set of tasks periodic tasks on a uniprocessor system, then EDF can also schedule that task set ok.

So, what is the bound?

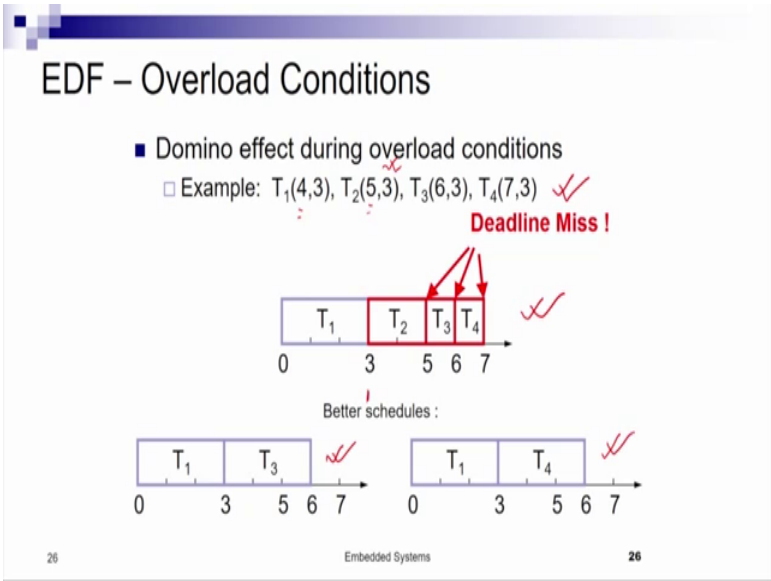
(Refer Slide Time: 29:26)



A real time system is shaped set to be schedulable under EDF if and only if the summation of its utilization is less than equals to 1. So, for RM we saw that in the worst case for a large number of tasks, if in the worst case in summation U i is less than equals to 69 percent there can be cases, where that task set will not become will become unschedulable. So, if the summation of U i is less is greater than 69 percent then it may become unschedulable. But for EDF, but for EDF the system remains schedulable if the total summation of utilization is less than 1. So, EDF is able to harness or utilize the complete capacity of the processor, but RM cannot.

So, both the RM and the EDF scheduling algorithm was first presented in a seminal paper by Liu and Leyland in published in Journal of ACM way back in 1973.

(Refer Slide Time: 30:34)

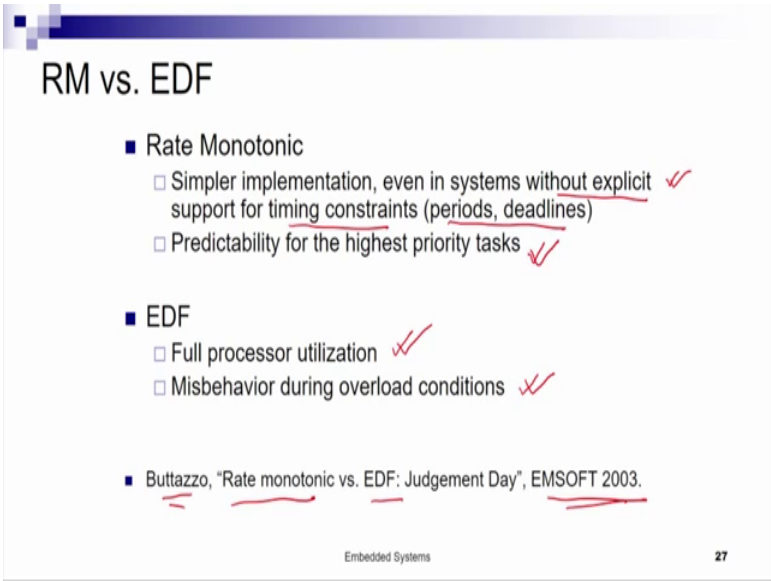


However EDF has its own problems for example, during transient overload conditions EDF may fail to meet the deadlines of any task. For example, let us say we have this task set and the summation of utilizations is greater than 1 and hence we have an overload situation here. In this overload situation we see that when T 1 having the shortest the first job of T 1 having the shortest deadline is going to be executed first and it takes 3 time steps to execute and completes at 3.

Now, after this T 2 will be executed and we see that T 2 has a deadline of 5, but we do not have sufficient time to complete its 3 units of execution requirement, and we have a deadline miss at 5. Similarly T 3 is also going to miss its deadline at 6 not being able to complete its entire execution requirement and similarly for T 4. In comparison RM has the advantage that only the lower priority tasks will be affected if for affected by a given task by any given task T i. So, the higher priority task will always get a higher priority during execution and due to a lower higher priority task is never finalized by due to misbehavior of a lower priority task. So, RM is better in this case.

Now, in this situation where everybody misses deadlines we possibly can have better schedules. For example, we just remove say T 2 and T 3 from the system and execute only T 1 and T 2, T 1 and T 3, then both are able to missed meet their deadlines or we can execute T 1 and T 4.

(Refer Slide Time: 32:37)



So, comparing between RM and EDF, the most important advantage of RM is that it has a very simple implementation being a static priority scheme suppose if we keep ready queues which are segregated classified based on priorities. So, I have a queue of for priority 1 I have a queue for priority 2 queue for priority 3, and we put the tasks in the in the queue in their respective queues based on the priority of the tasks, then it can be easily implemented the decision of which task to execute when it is just an one affair it can be done in constant time.

So, the implementation becomes simple it can be applied even in systems without explicit support for timing constraints such as periods and deadlines. Why because although the priorities are decided based on periods of the tasks, we do not actually require them during execution. We know its static priority and if we can fix its relative priority we will just push the task into its relative ready queue into its respective ready queue, and we do not need to bother about anything else.

RM also has this advantage that it provides better predictability for higher priority tasks. We saw that in the case of EDF deadlines may be missed due to a misbehaving task at the beginning. If a transient overload is caused by a misbehaving task which more time to execute than it is stipulated to take then all tasks suffer due to that, everybody else afterwards misses its deadlines.

In case of RM we saw that higher priority tasks whenever they arrive must be given the processor in preference over the lower priority tasks. So, higher priority tasks never suffer due to a lower priority task. A higher priority task obviously, may have to suffer due to further high priority tasks than this task, but it never suffers due to a lower priority task.

EDF on the other hand has the advantage that it allows full resource utilization; however, we saw that it means behaves during overload conditions. A much deeper analysis of a rate monotonic versus EDF is provided in this paper by Buttazzo ok. With this implementation we complete our discussion on uniprocessor scheduling, the real time scheduling algorithms for embedded systems.