# Embeded Systems - Design Verification and Test

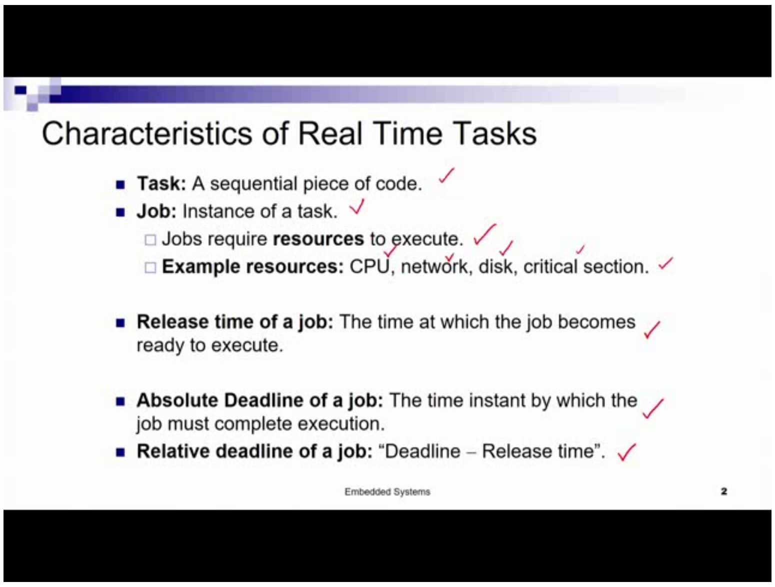
**Dr. Santhosh Biswas**
**Prof. Jatindra Kumar Deka**
**Dr. Arnab Sarkar**
**Department of Computer Science and Engineering**
**Indian Institute of Technology, Guwahati**

## Lecture - 14
## Real-time Uniprocessor Scheduling

Welcome. In this lecture we continue with our discussion on the system level performance analysis of embedded systems. In the last lecture, we discussed at techniques for estimating the worst case execution time of a single task. Now, after knowing the worst case execution time of a single task, we must embark on seeing the issues other issues which influence the end to end performance of an embedded system.

So, these will be extra task issues which are external to a single task. One such important issue is the contention among tasks which are allocated on the same processor. Now, we saw in the last lecture that such a contention is resolved through techniques known as uniprocessor scheduling techniques. In this lecture, we will go through an overview of two important uniprocessor scheduling techniques, the rate monotonic algorithm and the earliest deadline first algorithm. However, before going into the discussion of these algorithms we will first recap a few definitions.

(Refer Slide Time: 01:46)



### Characteristics of Real Time Tasks

- **Task:** A sequential piece of code. ✓
- **Job:** Instance of a task. ✓
  - Jobs require **resources** to execute. ✓
  - **Example resources:** CPU, network, disk, critical section. ✓
- **Release time of a job:** The time at which the job becomes ready to execute. ✓
- **Absolute Deadline of a job:** The time instant by which the job must complete execution. ✓
- **Relative deadline of a job:** "Deadline – Release time". ✓

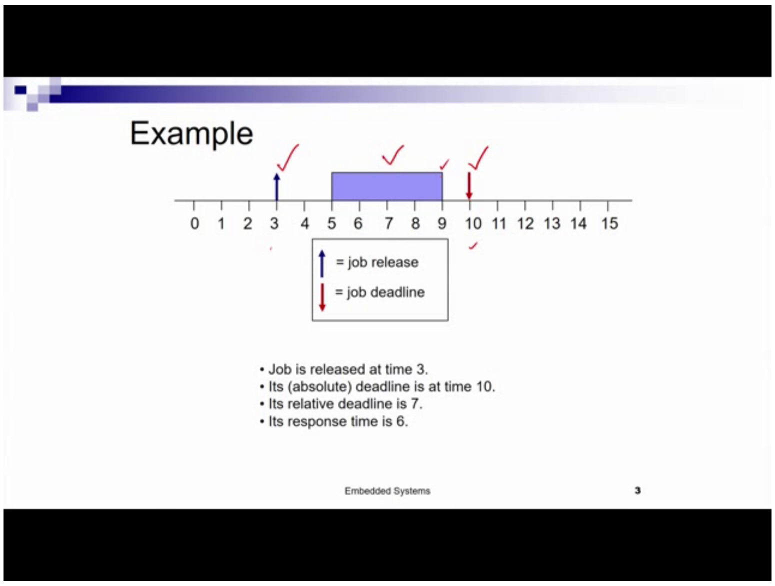
Embedded Systems 2

So, the definition of a task: A task is a sequential piece of code, it can be any sequential piece of code a c code, we saw in the introductory lecture that these tasks could be control tasks which are executed repeatedly to produce actuation outputs in the embedded system. And such each such repeated execution of that task is called a Job. So, job is an instance of the execution of the task.

So, once one time the code is executed, it is said that the job of the task is executed once. Jobs require resources to execute. For example, jobs will require CPU, their CPU or the processor to execute the instructions in the job. It may require the network to pass messages who other components in the system. It may be it may require disk script sections etc. So, jobs requires resources to execute, and they have a distinct release time, the release time of a job is the time at which the job becomes ready to execute. So, in the case of a periodic task that we saw in the introductory lecture, we saw that the job is released every time period t.

So that time period so, if it starts at 0, then 0 is a release time of the job, 0 plus t is a release time of the second job 0 plus 2 t is a release time of the third job in the case of that periodic task. Absolute deadline of a job; the time instant by which the job must complete execution is called the absolute deadline of the job. And the relative deadline of the job which is the deadline minus it is release time; it is the absolute deadline minus it is release time that is called the relative deadline of the job.

(Refer Slide Time: 03:45)



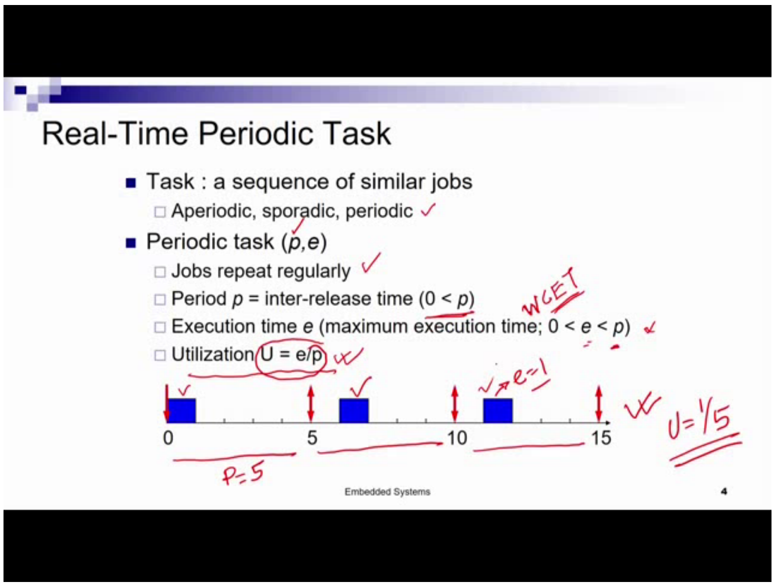
## Example

- Job is released at time 3.
- Its (absolute) deadline is at time 10.
- Its relative deadline is 7.
- Its response time is 6.

Embedded Systems 3

Now, here we take an example of a single job. So, here is a single job, the release time of this job is 3, the deadline of this absolute deadline of this job is 10. The relative deadline of this job is 7, which is 10 minus 3; the response time of the job is 6; which is 9 minus 3. So, response time is a time at which the output of the job is produced or the job complete subsequent to it is release.

(Refer Slide Time: 04:16)



## Real-Time Periodic Task

- Task : a sequence of similar jobs
  - Aperiodic, sporadic, periodic
- Periodic task ($p$,$e$)
  - Jobs repeat regularly
  - Period $p$ = inter-release time ($0 < p$)
  - Execution time $e$ (maximum execution time; $0 < e < p$)
  - Utilization $U = e/p$

Now with these definitions, we now come to a few more characteristics of a task. And important characteristic of a task depends on the nature of the inter arrival time of it is jobs. So, depending on the nature of the internal rival times of jobs of a task, a task can be Aperiodic sporadic or periodic. Task is considered to be a periodic if we have no idea between if we have no idea about the inter arrival times of consecutive jobs. So, the next job can arrive immediately after the completion of the current, job or it can time or it can arrive anytime afterwards, we have no idea, then we call this job to this task to be an Aperiodic task.

So, we do not have any idea about the inter arrival time of it is jobs. Aperiodic tasks are typically even driven tasks meaning that they these tasks are triggered after a certain event as is that certain event occurs. For example, one such task could be the controller for lifting the seat of a car safe. So, when the driver of the car presses a button in order to initiate lifting of the seat this task is triggered. And before and after that this task is not triggered. So, this task is even driven and hence these are these are Aperiodic tasks which are typically even driven. Tasks can be sporadic, in sporadic tasks we at least have an idea about the minimum internal rival time between jobs.

So, when the jobs are separated by a minimum inter arrival time, they are called sporadic. But we do not have any idea about the maximum inter arrival time. So, we know that the next job will not arrive before a minimum time is elapsed after the completion of the current job, but it can arrive anytime after this minimum internal rival time is passed. The other type of task is periodic tasks are the most pronoun predominant among control tasks, they are typically they typically occur as an infinite sequence of jobs, and such sequence of jobs and such sequence of jobs arrive periodically at fixed intern arrival times.

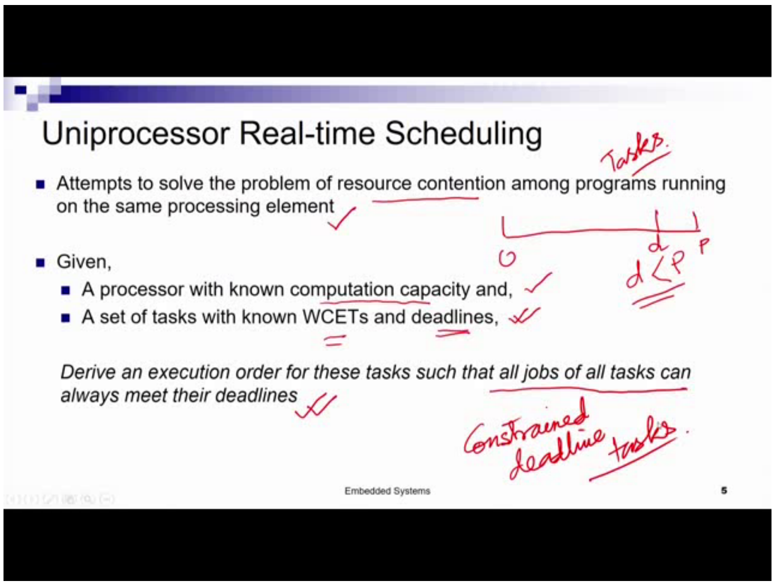
So, as we as we told that let us say this inter arrival time is p here and is called the period of the job. So, what we said to be t in the in the in the introductory class? We just renamed it as p and nothing more this t and p is same. So, this so, if the if the first job arrives at time 0, the next job is going to arrive at time 0 plus p thus, the third job is going to arrive at time 0 plus 2 p and likewise afterwards. So, jobs repeat regularly, the period p is the inter arrival time, and the inter arrival time is a value which is greater than 0. The execution time e is the maximum execution time, that is the WCET which we discussed in the last class. So, this e and WCET is same here and it is something which is between this e and p. So, it is greater than 0 and it is less than p.

So, if the execution time of a task on a certain processor is exactly equal to p; that means, that the whole capacity of this processor must be dedicated for the execution of this task. Because the execution time is equal to p and jobs repeat every p. So, the entire capacity of the processor can be has to be devoted for the execution of this task. And with this idea we come to the utilization of a task. The utilization of a task U is defined as e by p; which is the part of the capacity of a processor which must be dedicated for the execution of this task.

So, let us say if your e by p is half; that means, that half of the time on the processor must be must be dedicated for the execution of this task. And we can also say that through this through without a lot of deep analysis, we can easily understand that if we have 2 such tasks of size half, we cannot have any more tasks on this processor. Because half of the time will be used by tasks by the by one task, and the other half which is also has a need which also has a utilization of half will be taken by the other task.

So, no more tasks can be scheduled on that processor. So, in this case we have a periodic task here. The periodic task has a p value of 5, p equals to 5, because jobs arrive every 5 time units. The execution of this task e v c is equal to 1, we see that the time required is 1. So, the utilization of this task is U equals to 1 by 5. So, jobs arrive every 5 time units, and within the within with within this 0 to 0 to 5, we must execute this job within 5 to 10, we must execute the second job, between 10 to 15 we must execute the third job, likewise.

(Refer Slide Time: 10:34)



## Uniprocessor Real-time Scheduling

- Attempts to solve the problem of resource contention among programs running on the same processing element
- Given,
  - A processor with known computation capacity and,
  - A set of tasks with known WCETs and deadlines,

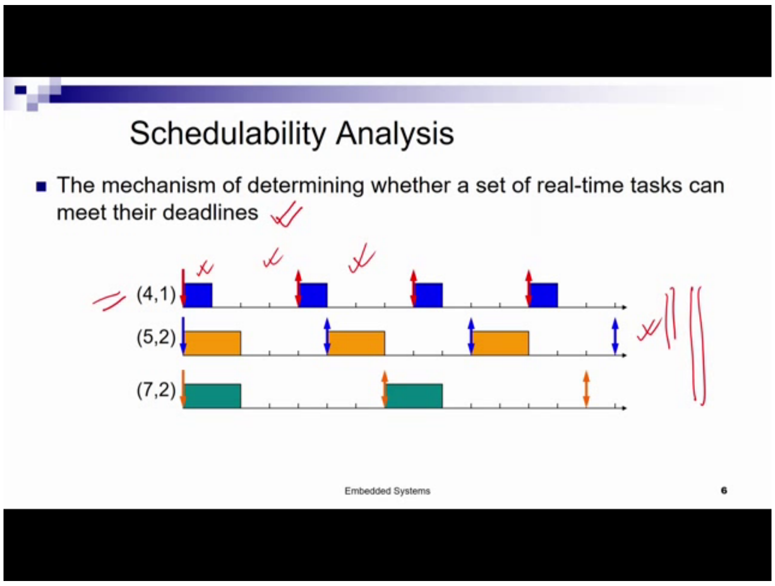
*Derive an execution order for these tasks such that all jobs of all tasks can always meet their deadlines*

So, with this idea we come to we will start our discussion on uniprocessor scheduling. So, as we said uniprocessor scheduling attempts to solve the problem of resource contention or processor contention here among programs running on the same processing element. Your programs and tasks are same. So, given a processor with known computation capacity and a set of tasks with known worst case execution times and deadlines, derive an execution order of these tasks such that all jobs of all tasks can always meet their deadlines.

So, this is the definition of what we want to achieve. We want to have a schedule means an order of execution of the tasks on this processor; such that, all jobs of all the tasks which execute on this processor will always meet their respective deadlines. Obviously, for this we must know the computation capacity of our processor and we also must know the deadlines of our jobs or the deadlines of the task if we have we know the relative deadlines of the jobs, and WCET is the worst case execution time that also must be known.

One implicit assumption that we are making here is that, the deadlines off of each job is equal to it is period; so, which may not always be the case. When this is the case that the deadline is equal to the period the relative deadline is equal to the period of a task of a job, then we saw when then we call such tasks as implicit deadlines tasks; however, we can also have the case that the deadlines d is less than period. So, if I if the task starts at 0, we it is p the deadline could be somewhere here. And such tasks are called constraint deadlines tasks, constraint deadlines tasks. So, here in our analysis, today we will look only at implicit deadlines tasks in this overview.

(Refer Slide Time: 13:05)



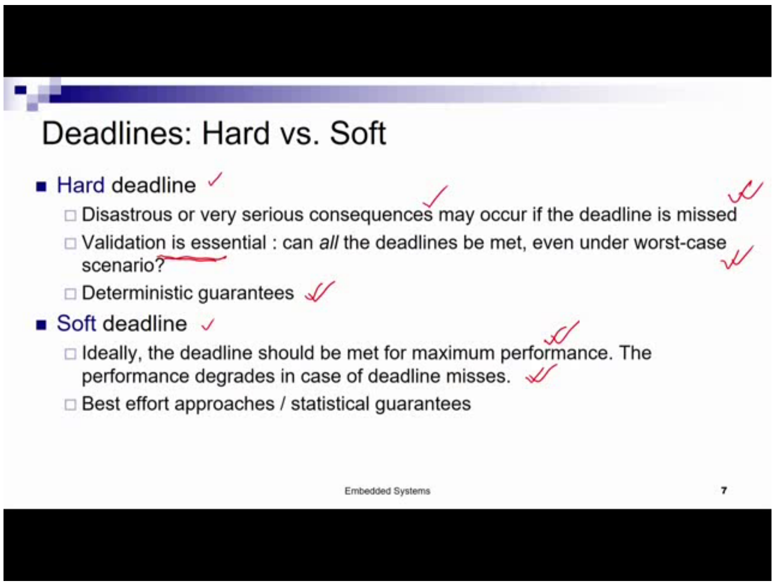
## Schedulability Analysis

- The mechanism of determining whether a set of real-time tasks can meet their deadlines

So, then what is shredder ability analysis? The mechanism of determining whether a set of real time tasks can meet their deadlines: the mechanism of determining whether a set of real time tasks can meet their deadlines. So, we have a set of tasks which needs to be scheduled on a processor. Can will I be able to schedule all the tasks on this processor such that all jobs of all tasks will always meet their deadlines? This analysis this mathematical analysis is called schedulability analysis. For example, suppose first task one comes. We can always understand that yes, this can be scheduled don that processor, on the processor. Every 5 time units I have just one task, job which has an execution requirement of one is going to come and I will be able to execute.

But let us say when a second task comes, will I be execute both these tasks together? How do I analyze? So, we do it through scheduled ability analysis. Suppose all 3 tasks come, will I be able to schedule all 3 tasks for a given scheduling algorithm? How do I know? We know it through this process of schedule ability analysis. And such analysis differs varies for different scheduling algorithms.

(Refer Slide Time: 14:22)



## Deadlines: Hard vs. Soft

- Hard deadline ✓
  - Disastrous or very serious consequences may occur if the deadline is missed
  - Validation is essential : can *all* the deadlines be met, even under worst-case scenario?
  - Deterministic guarantees
- Soft deadline ✓
  - Ideally, the deadline should be met for maximum performance. The performance degrades in case of deadline misses.
  - Best effort approaches / statistical guarantees

Embedded Systems 7

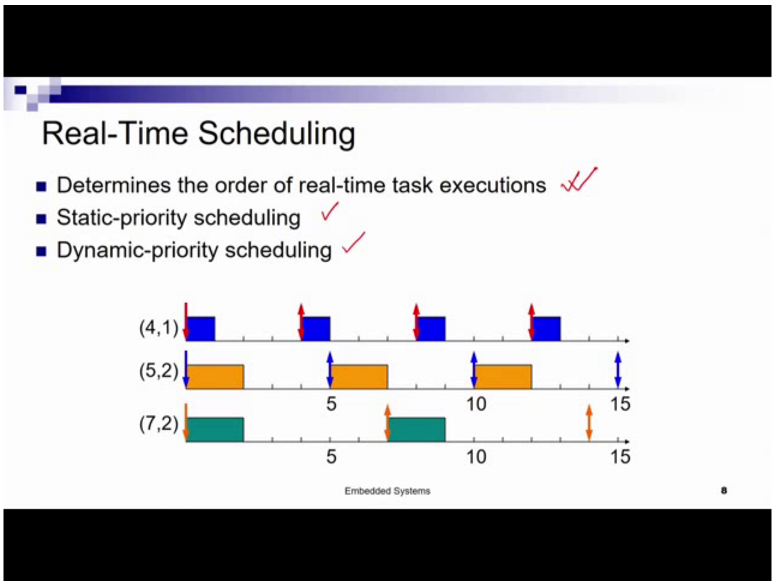
Another important criteria is with the deadlines of tasks. Deadlines of tasks can be hard or soft. So, when we have tasks with hard deadlines, the scheduling algorithms are called hard real time scheduling algorithms; when we have soft deadline tasks, we have soft real time scheduling algorithms. So, what are hard deadlines tasks? The tasks are said to have hard deadlines, when disastrous or very serious consequences may occur if the deadline is missed.

For example, let us say tasks deadlines within a nuclear reactor. If we miss the deadlines by certain or by even a very small amount, the reactions may result in an explosion. So, such tasks are hard deadlines tasks. Therefore, validation is extremely essential, we must very nicely do schedule ability analysis and hand guarantee that all tasks will always meet their deadlines even in the worst case scenario. So, we need deterministic guarantees. If the tasks are not that critical they may consider to have softer deadlines.

So, ideally the deadlines should be met in met for the maximum performance. We are allowed to miss deadlines; however, if we miss deadlines the performance. For example, an example of a soft real time task would be that let us say we are opening a web page. If the web page opens very quickly then we are very happy, but if the web page takes a certain it delays, if the web page delays in it is opening we slowly get frustrated.

So, the performance of the system in producing this web page to us degrades if degrades with the time with bit more as it takes more time to open the page ok. So, such tasks where the deadline has the essence of a reward missing deadlines do not have extremely serious consequences; however, has a sense of lower rewards as the latency beyond deadlines becomes higher, we call such tasks as soft deadline tasks. So, we use best effort approaches or statistical guarantees for such systems. Today in our discussion we will only look at hard deadline tasks and hence hard real time scheduling algorithms.

(Refer Slide Time: 17:10)



## Real-Time Scheduling

- Determines the order of real-time task executions ✓✓
- Static-priority scheduling ✓
- Dynamic-priority scheduling ✓

So, real time scheduling as we said determines the order of real time tasks executions. And there can be different types of scheduling algorithms depending on the types of tasks and the environment. For example, if that the tasks can be independent, where there is no dependency between them or tasks could be precedence constraint. Where the execution of a task the start of execution of a task could be or the completion of it the execution of a task could be dependent on the output produced by another task. Hence, there is a precedence constraint one the execution of one task must precede the other against independent tasks. Today we will consider independent tasks.

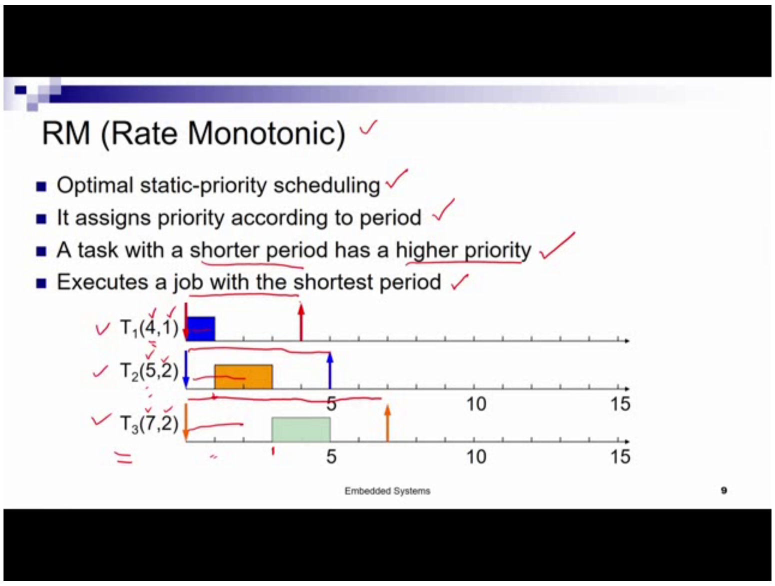
So, in a subsequent class we will also look at precedence constraint task sets. The scheduling algorithms could be online or offline; meaning that, the entire schedule of the tasks, the execution order of the tasks can be determined offline before putting the system into operation. So, I know what will be I run the algorithm I know the exact execution order. I have stored that execution order in the system, and use that execution order that determined execution order to run tasks during run of the system. That is that is an offline scheduling algorithm.

Against online scheduling algorithms where the order of tasks executions is actually determined when the system is executing; because enough information may not be available offline in such cases. We will look at a few offline algorithms for precedence constraint tasks. Today we will look at online scheduling algorithms that the scheduling algorithms can have static priority or dynamic priority. Meaning that: if the priority of the task does not change at runtime, the priority or the other relative importance of a particular task with respect to other tasks.

Relative importance in terms of getting the processor or getting the processor for execution; so, if the relative importance of a task remains fixed during the execution of that during execution at runtime it is called a static priority scheduling algorithm. So, a schedule algorithm can be online and still have static priorities. The scheduling algorithm can be online and can have dynamic priorities. So, the execution order of the task is decided online, but the relative importance of the tasks with respect to getting the processor will remain static.

Then it is a static priority scheduling algorithm; against dynamic priority scheduling algorithm, where the relative importance of the tasks can vary. So, we will look at the RM algorithm which is a static priority online scheduling algorithm. And we will look at e d f earliest deadline first which is a job level dynamic priority scheduling algorithm.

(Refer Slide Time: 20:30)



## RM (Rate Monotonic)

- Optimal static-priority scheduling
- It assigns priority according to period
- A task with a shorter period has a higher priority
- Executes a job with the shortest period

So, with this discussion we now look at our first scheduling algorithm the rate monotonic algorithm. As we said, it is a static priority algorithm and it is an optimal static priority algorithm; which means that given static given a set of static priority given all the static priority algorithm. If there is a task set which can be executed feasibility a scheduled; meeting all deadlines by any static priority scheduling algorithm RM can also schedule it therefore, RM is optimal.

Why is RM and optimal scheduling algorithm? Because if there exists any static priority scheduling algorithm, that can execute a task set feasibly meeting all deadlines on an uniprocessor system, then rate monotonic the RM algorithm can also schedule it. So, it at list schedule all scheduling algorithms that any other scheduling algorithm can execute hence it is optimal.

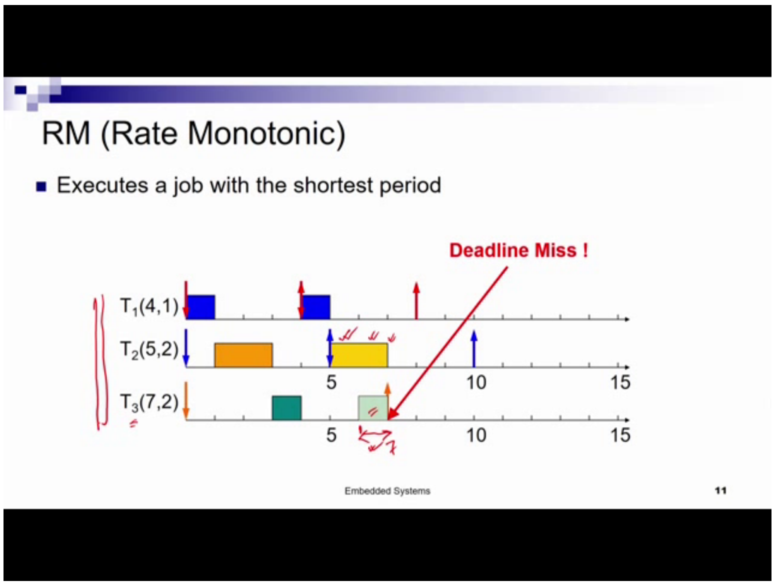
So, it assigns how does it execute, it is a it has a very simple static priority assignment strategy. What is the priority? The it assigns priority according to period. So, the period of the task is the frequency at which it is jobs arrive, lower the period or higher the frequency the jobs arrive higher is the priority. So, lower the period value higher is the priority of the task. Higher the period value or lower the frequency at which jobs of the task arrives lower is the priority. So, a task with shorter period has higher priority. And it executes a job with the shortest period first. So, at any point in time when whichever it gets whenever it gets a task whichever task it finds with the first period it executes that.

So, it has a set of tasks ready to execute, among that it will choose that task which has the shortest period at any point in time. This is a scheduling policy. For example, here we have 3 tasks T 1, T 2, T 3. And the period of T 1 is for, it is execution time is 1, period of T 2 is 5 execution time 2, period of T 3 is 7, execution time 2.

So, execution time is this length, and period is this length, ok. Now how does scheduling proceed? So, first at time 0 we all have these T 1, T 2, T 3 all these 3 tasks ready to execute now among them T 1 is the task with the shortest period 4. So, T 1 gets chance to execute, ok. After T 1 completes execution so, the first job of T 1 completes execution, and I know that the second job is only going to arrive at time 4 ok.

So, at time 1, at this time I only have 2 tasks T 2 in 2 jobs T 2 or one of T 2 one of T 3. And I know that among T 2 and T 3 the job of T 2 has higher priority because it has a lower period. So, T 2 completes it is execution the first job of T 2 completes it is execution. At this point in time, at time 3 what happens ah? T 1 there is no job of T 1, there is no job of T 2 and therefore, I can execute T 3.

(Refer Slide Time: 24:35)



## RM (Rate Monotonic)

- Executes a job with the shortest period

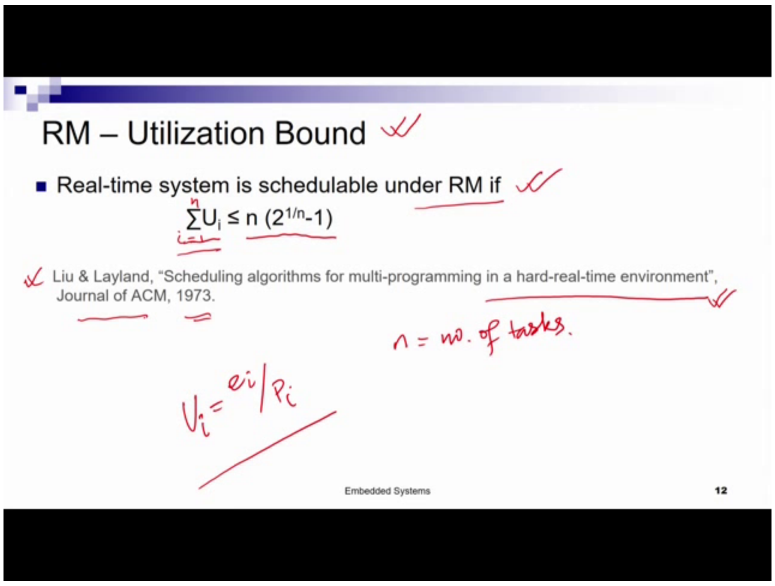
But after o execute the first part of T 3. So, T 3 has an execution requirement of 2 time units, out of it, the first time unit for the first time unit it can execute. And then what happens is that the second job of T 1 arrives.

Now, whenever the second job of T 1 arrives, we know that T 1 having a lower period has higher priority. And hence it pre-empts the execution of T 3. So, T 3 is sent from the processor to the ready queue, and T 1 is brought onto the processor for execution at this time. So, T 1 executes at time bit between 4 and 5, and it completes it is second job at time 5. Now at time 5 what happens? The second job of T 2 arrives.

So now, because T 2 has T 2 arrives at file and T 2 has a shorter period than T 3, the execution of the first job the remaining execution of the first job of T 3 has to be further postponed. Hence, what happens? Hence we see that we see that the first part of this the this t to execute and then what happens? At 6, we must we must give T 3 a chance to execute. Otherwise, it is deadline is 7 is going to be violated 43; however, RM decides the priority on the based on period.

So, T 2 still has higher priority at time 6, and hence T 2 completes it is execution at time 7. And then at this point in time, T 3 has no more time to complete the execution of it is first job. And therefore, we have a deadline miss 43 at time 7. So, therefore, the rate monotonic algorithm could not schedule this task set. Now the natural question therefore, is how do we understand whether this rate will know how will we understand whether this rate monotonic scheduling algorithm will be able to schedule an arbiter arbitrary tasks set or not. This as we said is done through a mechanism called schedule ability analysis.

(Refer Slide Time: 27:05)



## RM – Utilization Bound

- Real-time system is schedulable under RM if

$$\sum_{i=1}^{n} U_i \le n(2^{1/n}-1)$$

Liu & Layland, "Scheduling algorithms for multi-programming in a hard-real-time environment", Journal of ACM, 1973.

$n$ = no. of tasks.

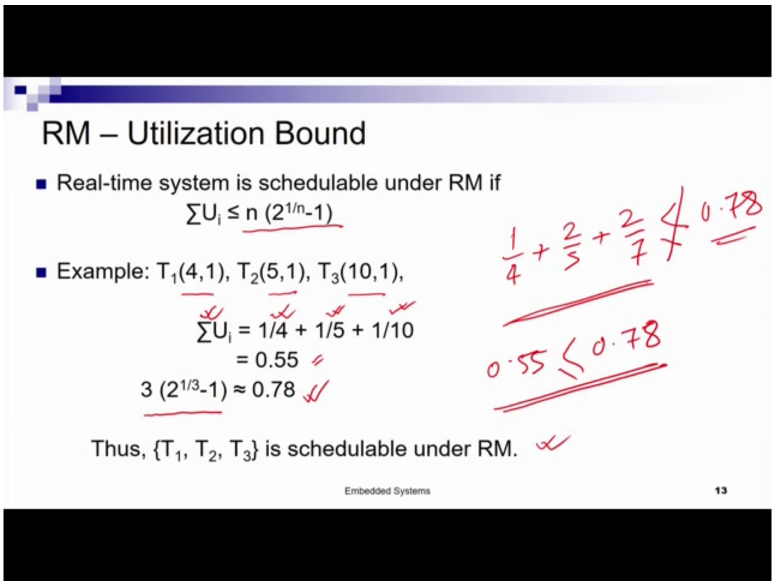
$U_i = e_i / P_i$

Embedded Systems 12

And the easiest she reliability analysis that we have for atom is the utilization bound. So, it says that a real time system is schedulable under RM on a uniprocessor real time system; obviously, and uniprocessor real time system is schedulable under the rate monotonic algorithm. If the sum of utilizations of it is tasks is less than equals to n into 2 to the power 1 by n minus 1; where n is the number of tasks.

So, n is the number of tasks in the system. And U as I told you, U is e by so, U i is e i by p i, ok. So, this is a guarantee, it says that if I can guarantee this, then I will always be able to schedule my task set. And this utilization bound and the RM algorithm as for that matter the e d f algorithm as well, was discussed first in a seminal paper by Liu and Leyland way back in 1973. The paper has the name scheduling algorithms for multi programming in a hard real time environment. And it was published in journal of a c m in 1973.

(Refer Slide Time: 28:41)



## RM – Utilization Bound

- Real-time system is schedulable under RM if
  $\sum U_i \leq n\,(2^{1/n}-1)$
- Example: $T_1(4,1)$, $T_2(5,1)$, $T_3(10,1)$,

  $\sum U_i = 1/4 + 1/5 + 1/10$
  $= 0.55$
  $3\,(2^{1/3}-1) \approx 0.78$

  Thus, $\{T_1, T_2, T_3\}$ is schedulable under RM.

$\frac{1}{4}+\frac{2}{5}+\frac{2}{7} \nleq 0.78$
$0.55 < 0.78$
Embedded Systems
13

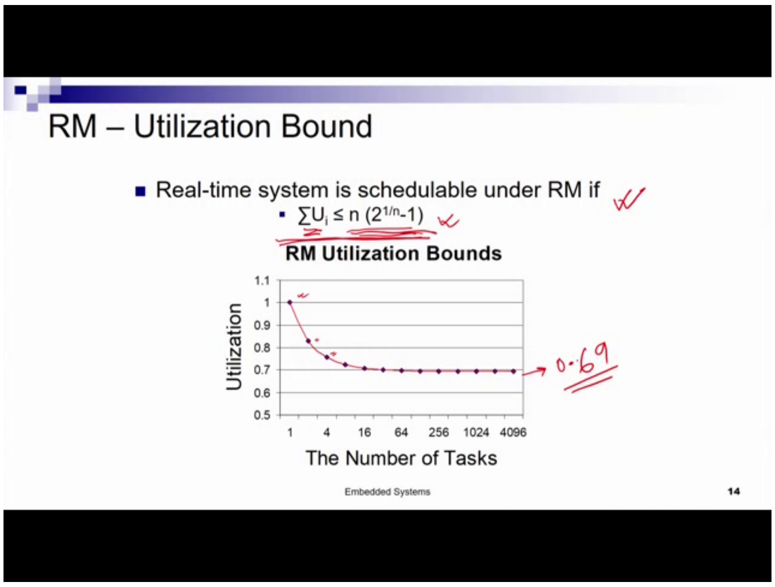
So, let us take the example of 3 tasks here. We have 3 tasks, where 4 1 p is equals to 4 execution time 1; period 5 execution time one for task T 2, and period 10 execution time one for task T 3. For these 3 tasks, we have summation of utilization is what? 1 by 4 plus 1 by 5 plus 1 by 10 which is equal to 0.55.

Now if we calculate this n into 2 to the power 1 by n minus 1 when we have 3 tasks, this becomes 3 into 2 to the power 1 by 3 minus 1. And which is equals to which is equal to

0.78. So, this ensures that these 3 tasks are scheduled able under RM, why? Because 0.55 is less than 0.78.

So, because this condition is satisfied, I know that these 3 tasks will always be schedule able under RM and will never miss a deadline. But we saw that the earlier task that we saw here missed a deadline. If you go into that, we will see that for those 3 tasks for those 3 task. So, the 3 tasks that we had here is 1 by 4 2 by 5 and 2 by 7. If we do this calculation, we will see that 1 by 4 plus 2 by 5 plus 2 by 7 is not greater less than equal to 0.78. You will see this, if you just calculate it you will find this.

(Refer Slide Time: 30:33)



## RM – Utilization Bound

- Real-time system is schedulable under RM if
  - $\sum U_i \leq n\,(2^{1/n}-1)$

Embedded Systems

14

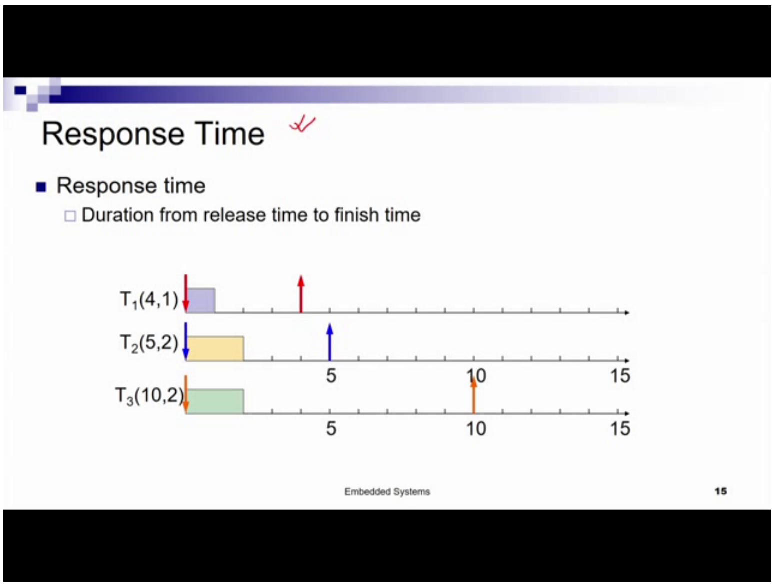
So, see that this happens in the worst case. This is a sufficiency bound not a necessary condition for RM to schedule. Meaning that there could be task sets whose utilization is greater than 0.78 for 3 tasks, but they are still scheduled level; however, if we have they are still scheduled level. However, if it is less than 0.78 it will always be schedulable we do not need any other analysis. If it is greater than 0 point if the if the utilization of the task set is greater than 0.78 we need other means mechanisms for analysing.

So, more complex mechanisms for analysing whether that task set will be schedulable under RM. This always considered for 3 tasks in general which we have if this condition is not satisfied that is summation U i is greater than this value, then it is not necessary that this task set will not be schedulable. This task set can still be available, but we need for the test to understand whether it will be actually schedulable or not under RM.

However, if we have this bound, if this bound is satisfied we have a very simple test and we can be ensured that the task set will be schedulable. Now when we have one task this bound is one. So, this value is 1, when we have 2 tasks, this value is around 0.8283. We saw for 3 task this value is around 0.78. And likewise for n task when then it went when n tends to infinity, this 0 is it goes to around 0.69.

So, we can say that for any arbitrary task set of any big size will be schedulable under RM. So, any arbitrary task said meaning the task set can contain any arbitrary number of tasks, but for that task set we can ensure that that task set will be schedulable under RM on a uniprocessor system, if the total execution, if the total utilization of that task set is less than about 0.69, ok.

(Refer Slide Time: 32:55)



## Response Time

- Response time
  - Duration from release time to finish time

Embedded Systems 15

But, if that is not so we have to look at other mechanisms of schedulability analysis. And one such schedulability analysis depends on the response time of the jobs of the tasks. We now discuss this analysis method.