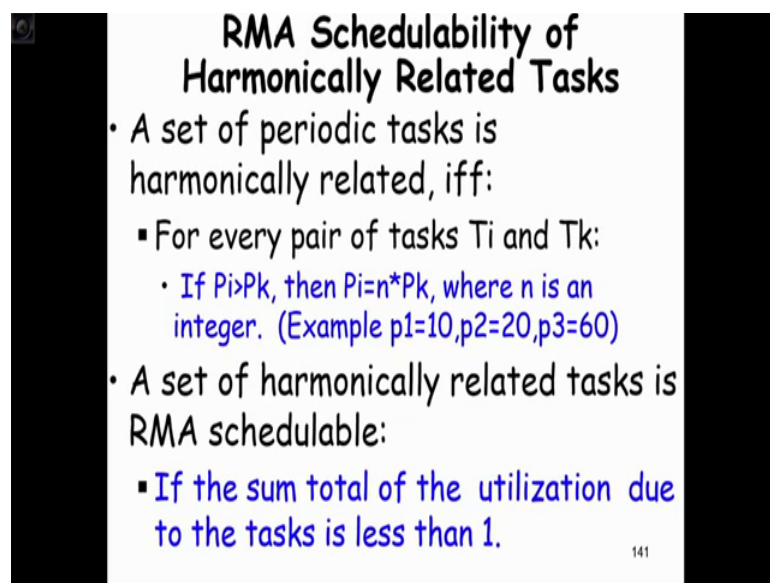


Real Time Operating System
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 11
RMA Generalizations

Welcome back. We were looking at the rate monotonic scheduler and we have looked at some issues.

(Refer Slide Time: 00:23)



RMA Schedulability of Harmonically Related Tasks

- A set of periodic tasks is harmonically related, iff:
 - For every pair of tasks T_i and T_k :
 - If $P_i > P_k$, then $P_i = n * P_k$, where n is an integer. (Example $p_1=10, p_2=20, p_3=60$)
- A set of harmonically related tasks is RMA schedulable:
 - If the sum total of the utilization due to the tasks is less than 1.

141

The schedule ability analysis and also looked at the tangent overload handling and so on. Now, let us look at some specific points that we need to understand regarding the rate monotonic scheduler. One is that how does the rate monotonic scheduler work with harmonically related tasks.

We say that a task set is harmonically related if that if given that any task has higher period than another task then it must be a multiple of the lower period task or in other words if we take any 2 tasks in the task set T_i and T_k let us say and we find that the period of T_i which is P_i is greater than P_k , then P_i is sum multiple of P_k and that holds for every task in the task set.

Whenever we take two tasks if one task has higher period than other task then it must be multiple of the other period. Just to give an example of a harmonically related task set let

us say for the p1 the task 1 the period is 10, for task 2 the period is 20 and for task 3 the period is 60.

In this case you can check that if we take 2 arbitrary tasks t 1 and t 2, t 2 is greater than p2 is greater than p1, but then it is a multiple of p1. If we take p2 and p3, p3 is greater than p2, but again p3 is a multiple of p2. So, that holds for every pair of tasks. So, this is an example of a harmonically related setup tasks and this is our special interest to us because something very unique happens here. If a task set is harmonically related then the schedule ability criterion becomes that the utilization even if the utilization is 1 still the task set becomes schedulable.

Let us say, we have 10 tasks with task periods that are harmonically related and if you use the Liu Layland criterion you will get something like 0.7, less than 0.7 utilization task set will not be schedulable that will be the application of the Liu Layland to any task. But, then here in the harmonically related task set even if the utilization is up to one still the task set remains schedulable, but how do we show this.

(Refer Slide Time: 03:30)

Schedulability of a Harmonic Task Set

- By completion time theorem:

$$e_i + \sum_{k=1}^{i-1} \left\lceil \frac{p_i}{p_k} \right\rceil * e_k \leq p_i$$
- For harmonically related tasks:
 - Ceiling can be removed since the periods are integral multiples.
 - So, $\frac{e_i}{p_i} + \sum_{k=1}^{i-1} \frac{e_k}{p_k} \leq 1$
 - or, $\sum_{i=1}^n \frac{e_i}{p_i} \leq 1$

Now, let us through a simple mathematics, let us show that any harmonically related tasks set which schedule able under the rate monotonic scheduler as long it is utilization is less than or equal to one how do we show it let us try the completion time theorem. We know that the completion time theorem for every task set we must have e i and the time taken for the task t i we need to check whether it is execution time plus the time taken for

executing all its higher priority tasks that arrive before it is period is less than equal to its period.

But, for a harmonically related task set we know that p_i and p_k are multiples of each other. So, p_i is a multiple of p_k and therefore, we can get rid of the ceiling because it perfectly divides e_i is divisible by p_k therefore, we can take out the ceiling and we can write e_k by p_k , so, e_i by p_i .

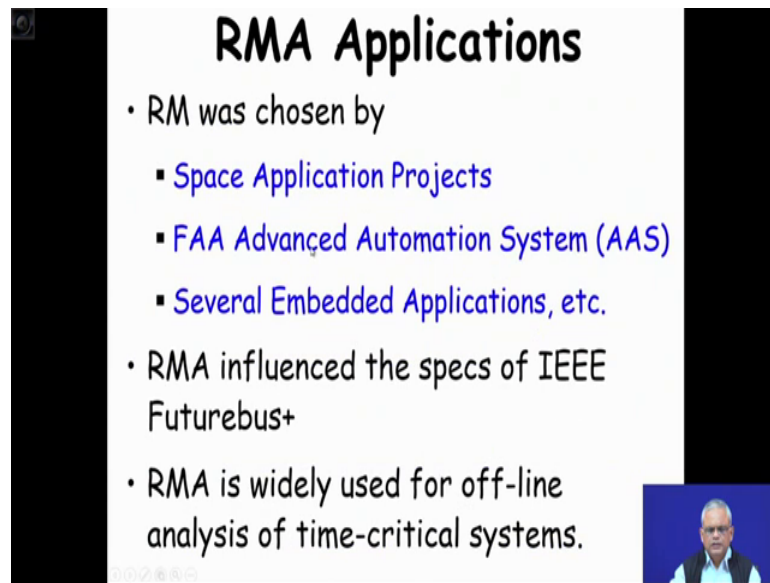
So, we have divided p_i by all through. So, we write e_i by p_i and $\sum_{k=1}^{i-1} e_k$ by p_k the p_i had come out from the ceiling and therefore, p_i is cancelled by the division and therefore, we write e_i by p_i $\sum_{k=1}^{i-1} e_k$ by p_k is less than 1 or we can re-write this expression as $\sum_{k=1}^{i-1} \frac{e_k}{p_k}$ is less than 1, less than equal to 1. So, by the completion time theorem we can show that as long as the sum of the utilization of the task set is less than 1, then it will meet its deadline.

(Refer Slide Time: 05:53)

RMA vs. EDF	
• Implementation multi-level priority queue, $O(1)$	• Heap, $O(\log n)$
• Processor utilization 0.69 (expected 0.88)	• Processor utilization ---full utilization
• Context switches -- many	• context switches few
• Guarantee test nontrivial	• simple

If we compare whatever we learnt about the rate monotonic scheduler and the EDF we can see that the rate monotonic scheduler has lot of advantage. Its implementation is more efficient in a multi level priority queue of course, the processor utilization is slightly lower than the EDF and we will look at the context switches slightly more context switches than the EDF and the guarantee test is slightly non trivial need to check the Lius key or the Liu Layland criterion whereas, the EDF is simple utilization sum of utilization is less than 1.

(Refer Slide Time: 06:45)



RMA Applications

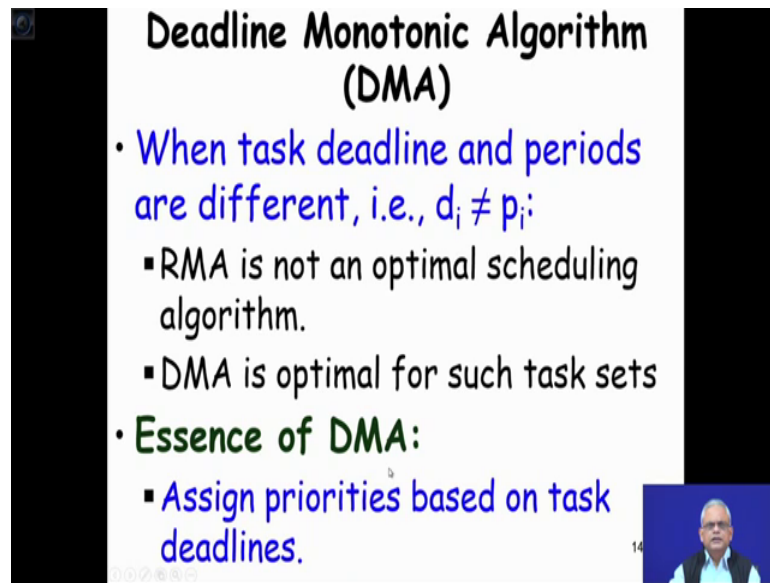
- RM was chosen by
 - Space Application Projects
 - FAA Advanced Automation System (AAS)
 - Several Embedded Applications, etc.
- RMA influenced the specs of IEEE Futurebus+
- RMA is widely used for off-line analysis of time-critical systems.

00:28:30

The rate monotonic scheduler is overwhelmingly popular, used in many applications; space applications, advanced automation systems, large number of embedded applications, even as influence the specification of the IEEE future bus and in time critical applications the rate monotonic scheduler is used and is supported in almost every operating real time operating system.

Now, let us look at certain rate monotonic scheduler generalizations because it does not handle certain conditions the simple algorithm that we looked at, we need to have certain other enhancements that will cater to these situations.

(Refer Slide Time: 07:42)



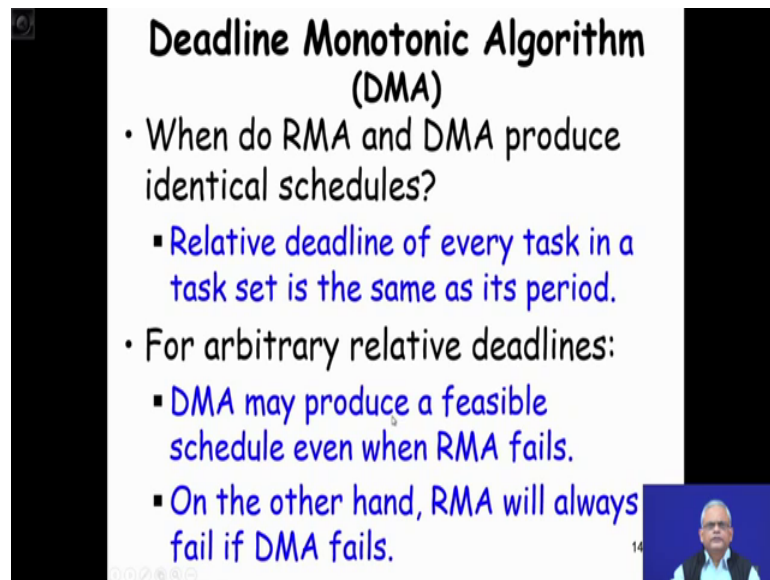
Deadline Monotonic Algorithm (DMA)

- When task deadline and periods are different, i.e., $d_i \neq p_i$:
 - RMA is not an optimal scheduling algorithm.
 - DMA is optimal for such task sets
- Essence of DMA:
 - Assign priorities based on task deadlines.

The first one we look at is the dead line monotonic algorithm just a small variation in the rate monotonic algorithm and this we need to use when the task deadline and periods are different. If you would have noticed in all our examples so far, we considered the task period and dead line to be the same, but there can be situations in practical applications where the task deadline and period are different for example, deadline may be less than the period or dead line may be more than the period. Of course, dead line being more than a period is extremely rare whereas, there are many cases where the deadline is less than the period.

So, in cases where deadline is not equal to the period rate monotonic algorithm is not really the optimal scheduling strategy for these cases the deadline monotonic algorithm is optimal, but what is the main idea behind the deadline monotonic algorithm it is very similar to the rate monotonic algorithm excepting that the priorities 2 tasks are assigned based on their deadlines rather than their periods. So, even if a task has higher period than another task, but it has a lower deadline then that gets higher priority.

(Refer Slide Time: 09:23)



Deadline Monotonic Algorithm (DMA)

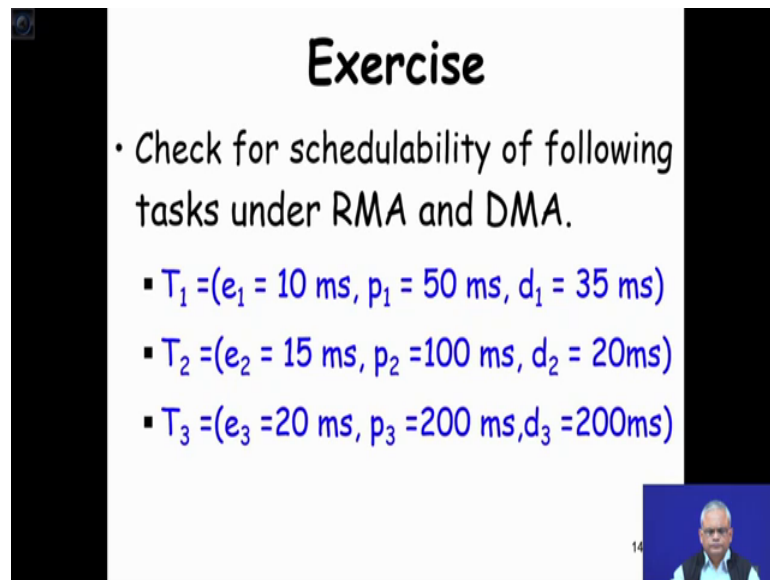
- When do RMA and DMA produce identical schedules?
 - Relative deadline of every task in a task set is the same as its period.
- For arbitrary relative deadlines:
 - DMA may produce a feasible schedule even when RMA fails.
 - On the other hand, RMA will always fail if DMA fails.

First let us answer this question that do RMA and DMA, Rate Monotonic Algorithm and the Deadline Monotonic Algorithm both of them do the produce identical schedule under some situations. With little thinking, we can say that if the period and deadline are the same, then of course, the deadline monotonic algorithm it simplifies into the rate monotonic algorithm.

But, for arbitrary relative deadlines where the deadline may be different from the period they may produce different schedules. For example, even when the rate monotonic scheduler fails the deadline monotonic scheduler may come up with a feasible solution. But, on the other hand whenever the deadline monotonic scheduler fails the rate monotonic scheduler will definitely fail.

We will not really look into the mathematical theorem proof behind this statement we just assume at it is phase value, but then in the books that we refer proofs are available you can go through, but for our purpose we will just use it at its phase value, that the deadline monotonic algorithm can produce a feasible schedule when the rate monotonic scheduler may not. Whereas, whenever the deadline monotonic scheduler fails the rate monotonic scheduler will also fail.

(Refer Slide Time: 11:15)



Exercise

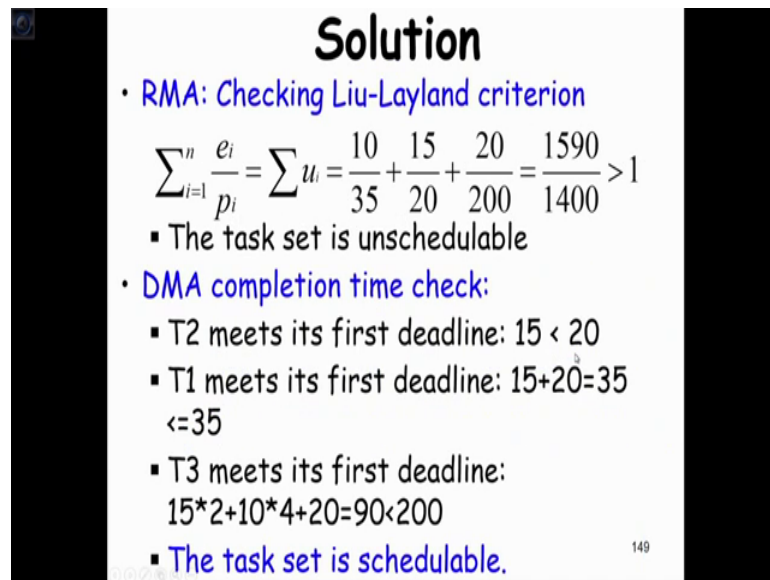
- Check for schedulability of following tasks under RMA and DMA.
 - $T_1 = (e_1 = 10 \text{ ms}, p_1 = 50 \text{ ms}, d_1 = 35 \text{ ms})$
 - $T_2 = (e_2 = 15 \text{ ms}, p_2 = 100 \text{ ms}, d_2 = 20 \text{ ms})$
 - $T_3 = (e_3 = 20 \text{ ms}, p_3 = 200 \text{ ms}, d_3 = 200 \text{ ms})$

14

Now, let us look at one example we have a task set consisting of 3 tasks T_1 , T_2 , T_3 , their execution time period and deadline is given. Now, we need to check for their schedule ability using the rate monotonic algorithm and the deadline monotonic algorithm. If you apply the rate monotonic algorithm the task T_1 is the highest priority task. Task T_2 is the second higher priority and T_3 is the lowest priority task.

But, in the deadline monotonic algorithm task T_2 is the highest priority task because it has a lowest deadline, task T_1 is the second highest priority task and task T_3 is the lowest priority. This indicates that the priorities assigned to the tasks are different and the difficulty here is that the results that we got for the rate monotonic analysis that is completion time theorem Liu Layland criterion etcetera, would not apply to the deadline monotonic algorithm because here the deadline assignment is different than the rate monotonic algorithm.

(Refer Slide Time: 12:46)



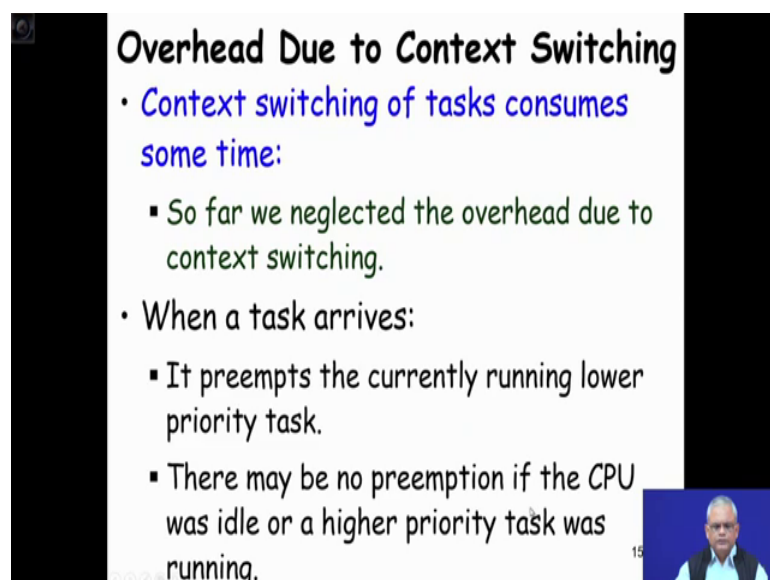
Solution

- **RMA: Checking Liu-Layland criterion**
$$\sum_{i=1}^n \frac{e_i}{p_i} = \sum u_i = \frac{10}{35} + \frac{15}{20} + \frac{20}{200} = \frac{1590}{1400} > 1$$
 - The task set is unschedulable
- **DMA completion time check:**
 - T2 meets its first deadline: $15 < 20$
 - T1 meets its first deadline: $15 + 20 = 35 \leq 35$
 - T3 meets its first deadline: $15 * 2 + 10 * 4 + 20 = 90 < 200$
 - **The task set is schedulable.**

149

But, then we can check it manually, so, you can check the Liu Layland criterion for the rate monotonic analysis we find that it is un schedulable and we can do a completion time check for the deadline monotonic algorithm and check whether all the tasks sets meets their first deadline and we see that the task set is schedulable under the deadline monotonic algorithm.

(Refer Slide Time: 13:23)



Overhead Due to Context Switching

- **Context switching of tasks consumes some time:**
 - So far we neglected the overhead due to context switching.
- **When a task arrives:**
 - It preempts the currently running lower priority task.
 - There may be no preemption if the CPU was idle or a higher priority task was running.

15

Now, let us look at another aspect which is also practically important is the context switching. The context switching time for operating systems is substantial sometimes

compared to the small task execution times. The task execution times are of the order of few milliseconds may be 50 milliseconds or 20 milliseconds, maybe 10 milliseconds whereas; the context switching time is typically 1 millisecond or so on. So, when we have task deadlines and the task execution times so close, so comparable to the task to the context switch times of the operating system we cannot really ignore them in our analysis. But, then if we really want to include that in our analysis how do we do it? Let us look at that point.

The crux of our solution here about how to handle context switching in the rate monotonic analysis is to consider the following situation that when do the task preemptions occur and whenever task preemption occurs we need to consider the context switch time. One task preemption can occur when there is a lower priority task running and then there is a higher priority task which has arrived then the higher priority task preempts the lower priority task and also when the highest priority task completes again the lowest priority task resumes its execution.

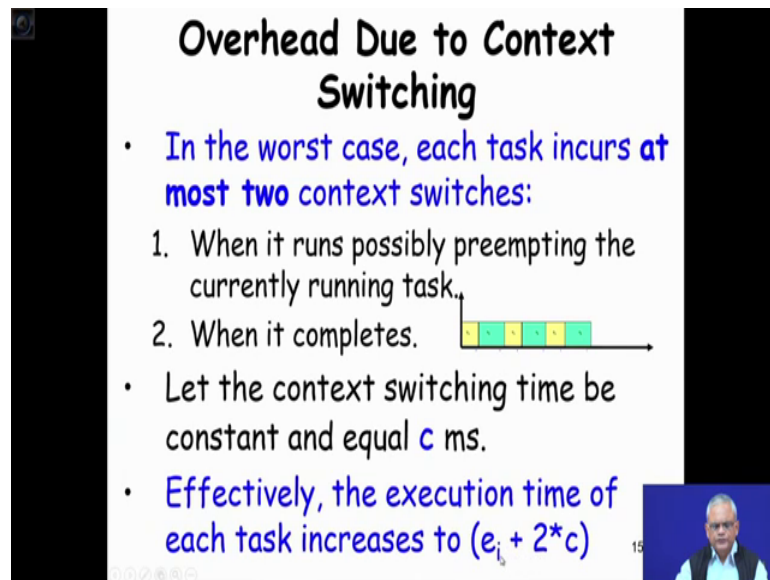
So, there is again a context switch. So, in the worst case we can assume that each time there is a task that arrives it may at most cause one context switch because the task that is that may be running is context switched and again when it is on it is completion there may be a context switch, but then as you can easily look through this that this is a very conservative analysis because you are assuming that each time a higher priority task is running sorry a lower priority task is running and higher priority task arrives and there is a context switch, but we are over looking that sometimes higher priority task may be running and a lower priority task arrives and there is no context switch, but then in the real time systems all our analysis are conservative analysis. We look at the worst case and do a worst case analysis.

So, from that point this is quite ok that we assume that whenever a task arrives it causes 2 context switches; one the running task is context switched and the arriving task is taken up a running, in the second on completion of the task the one that was already context switch resumes its run, so, there are 2 context switches for every task arrival.

(Refer Slide Time: 17:10)

Overhead Due to Context Switching

- In the worst case, each task incurs at most two context switches:
 1. When it runs possibly preempting the currently running task.
 2. When it completes.
- Let the context switching time be constant and equal c ms.
- Effectively, the execution time of each task increases to $(e_i + 2*c)$



And, once we understand that then the answer about how to take context switch overheads becomes extremely simple. So, we assumed that each task instance that is each job incurs at most 2 context switches when it starts to run preempts the currently executing tasks and when it completes. Now, let us assume that the context switch time is constant see that is the worst case context switch time see let us say c millisecond.

So, we can take the effect of context switching by just adding $2c$ to the execution time of every instance of the tasks. So, in effect we just transform our task set into one where execution time of every task is increased by $2c$, but then of course, you may agree or you may argue that see the lowest priority tasks the never contest switches any task why do we consider $2c$ for that? It can be $1c$ like nothing was running and it started to run may be $1c$, but then we are doing a worst case analysis and for every task we increase it by $2c$.

(Refer Slide Time: 18:45)

Example

- Assume 3 periodic tasks:
 - T1: $e_1=10\text{mSec}$, $p_1= d_1= 50\text{mSec}$
 - T2: $e_2=25\text{msec}$, $p_2= d_2= 150\text{mSec}$
 - T3: $e_3=50\text{mSec}$, $p_3= d_3= 200\text{mSec}$
 - Assume context switching time = 1msec
 - Determine whether the task set is schedulable.

152

So, this is an example we have a task set of 3 tasks T 1, T 2, T 3 and their execution times are 10, 25 and 50 millisecond periods are 50, 150 and 200 and we assume that the context switch time is 1 millisecond and we want to check whether the task set is schedulable. To take the context switching time into account we need to increase the execution time of every task by 2 millisecond. So, our new task set becomes 12, 27 and 50 milliseconds. So, that was 10, 25 and 50, it will become 12, 27 and 52.

(Refer Slide Time: 19:40)

Answer

- Effect of context switch:
 - Execution time of each task increases at most by 2 msec.
 - **Task T1:** $12\text{msec} < 50 \text{mSec}$
 >>Schedulable
 - **Task T2:** $27 + 12*3=63 < 150 \text{msec}$
 >>Schedulable
 - **Task T3:** $52 + 12*4 + 27*2 = 154 < 200\text{msec}$ >>Schedulable

T1: $e_1=10\text{mSec}$,
 $p_1= d_1= 50\text{mS}$

T2: $e_2=25\text{msec}$,
 $p_2=d_2= 150\text{msec}$

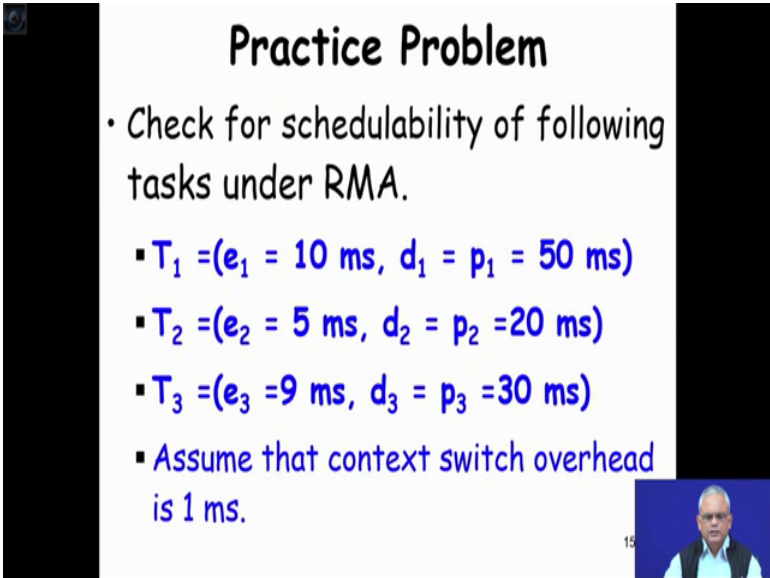
T3: $e_3=50\text{mSec}$,
 $p_3= d_3= 200\text{mSec}$

153

So, we check whether 12 millisecond is less than 50 milliseconds. So, that is task T1 has the highest priority find that it is schedulable. Similarly, we check for the second task 27 after taking 2 context switch the every execution instance. The higher priority task can arrive at most 3 times. So, this is also schedulable for task T 3, we need to check that fifty 2 millisecond after taking the context switching into account and it is too higher priority tasks that take this is the execution time means period and therefore, it is also schedulable.

So, considering the context switch time under the simplified assumptions that we consider the worst case context switch time and also we will assume that every task arrival for this 2 context switches. So, under that we can do an analysis for a practical situ situation.

(Refer Slide Time: 20:54)

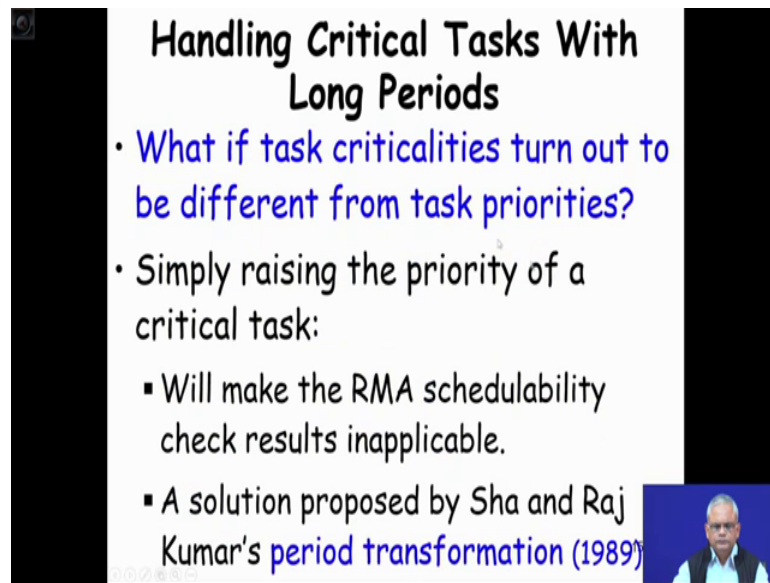


Practice Problem

- Check for schedulability of following tasks under RMA.
 - $T_1 = (e_1 = 10 \text{ ms}, d_1 = p_1 = 50 \text{ ms})$
 - $T_2 = (e_2 = 5 \text{ ms}, d_2 = p_2 = 20 \text{ ms})$
 - $T_3 = (e_3 = 9 \text{ ms}, d_3 = p_3 = 30 \text{ ms})$
 - Assume that context switch overhead is 1 ms.

There is some practice problems please do it. One is, check whether the following task set is schedulable we have T 1, T 2, T 3 different tasks with different execution times and periods and we assume that the context switch overhead in the worst case is one millisecond the effect will be to increase the execution by 2 millisecond. So, this will become 11, 7 and 12, but what about the priorities of the tasks which task will have the highest priority. So, we look through the period and find that T 2 has the lowest period and that has the highest priority a priority one for T 2 priority 2 for T 3 and the lowest priority for T 1.


(Refer Slide Time: 21:54)



Handling Critical Tasks With Long Periods

- What if task criticalities turn out to be different from task priorities?
- Simply raising the priority of a critical task:
 - Will make the RMA schedulability check results inapplicable.
 - A solution proposed by Sha and Raj Kumar's period transformation (1989)

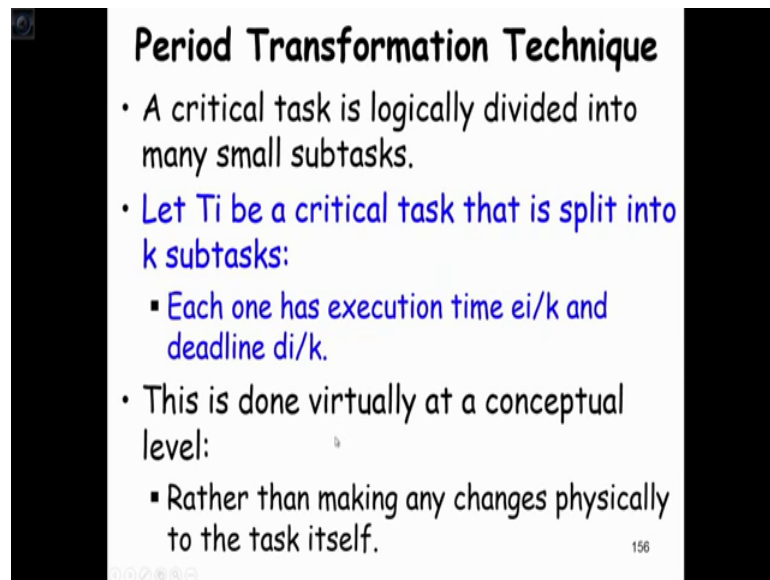
© 2024



Now, let us examine another situation, that what happens if task is extremely critical, but then it is not the shortest task its period is larger than some tasks, but this task is more critical than the other tasks, how do we handle this situation? Because, in a simple rate monotonic priority assignment we just assign priorities based on the frequency of occurrence or the period of repetition and then, in that case the shortest period will get the highest priority, but we know that some task with higher period has most criticality. How do we handle this situation? Because, simply giving it a highest priority that will make our all the schedule ability results unusable.

If a task period is 100 and we assign it priority one when there are other tasks like 20, 50 etcetera and they are given lower priority then all our Liu Layland, Liu (Refer Time: 23:18) criterion etcetera will be difficult to use. So, solution proposed by Sha and Raj Kumar known as the period transformation method 1989. So, this is the standard technique to handle situations where a critical task has a long period or its frequency of occurrence is lower.

(Refer Slide Time: 23:51)



Period Transformation Technique

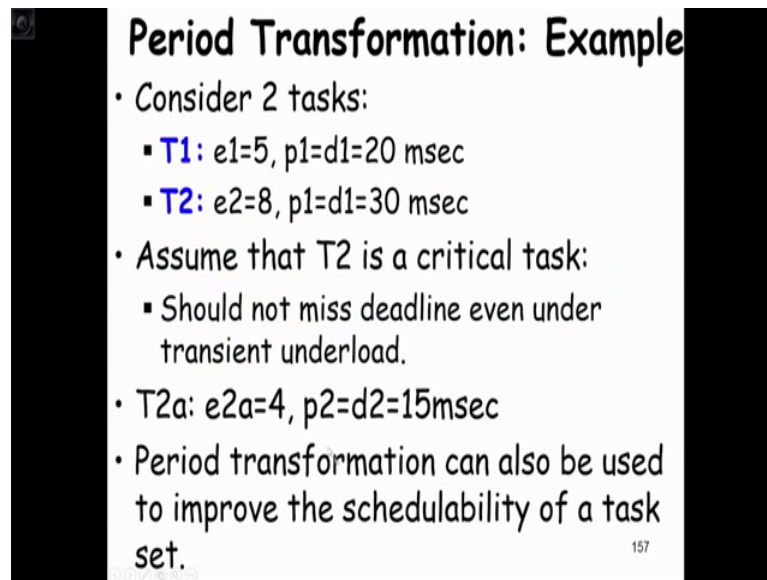
- A critical task is logically divided into many small subtasks.
- Let T_i be a critical task that is split into k subtasks:
 - Each one has execution time e_i/k and deadline d_i/k .
- This is done virtually at a conceptual level:
 - Rather than making any changes physically to the task itself.

156

The main idea here is that we divide the critical task into smaller sub tasks. We do not really physically subdivide, do not really take out the code and start dividing it is only for analysis. So, what we do is we just consider its execution time and we want to split it into k parts we assume that each part runs for e_i by k and deadline d_i by k . So, if you want to split into 2 parts, and the task execution time was let us say 50 and period was 100 we assume that it is actually 2 tasks. Each one execution time 25 and period is 50. So, that way we have increased it is priority and still the Liu Layland and Liu (Refer Time: 24:56) results become applicable.

So, the task splitting is done at a conceptual level rather than making change to the code itself.

(Refer Slide Time: 25:11)



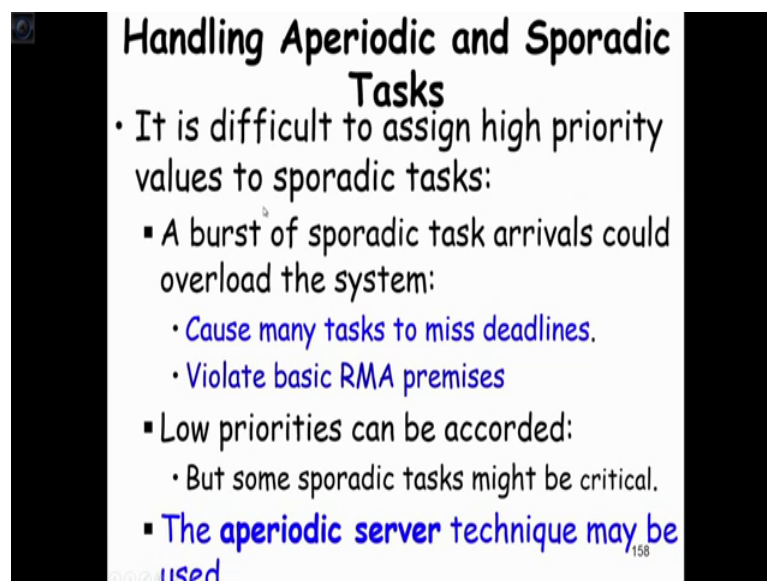
Period Transformation: Example

- Consider 2 tasks:
 - **T1**: $e_1=5$, $p_1=d_1=20$ msec
 - **T2**: $e_2=8$, $p_1=d_1=30$ msec
- Assume that T2 is a critical task:
 - Should not miss deadline even under transient underload.
- T2a: $e_{2a}=4$, $p_2=d_2=15$ msec
- Period transformation can also be used to improve the schedulability of a task set.

157

Let us take some example. Let us assume that we have 2 tasks T1 requires 5 milliseconds for every 20 millisecond and T2 it requires 8 millisecond. Every 30 millisecond and T2 let us assume that it is the critical task. So, it should not miss dead line on account of the T1 we need to keep a higher priority to T2, in that case what we do is, we assume we consider that T2 consists of 2 tasks T2a and T2b each one having execution time 4 and period of 50 and then we can use the rate monotonic analysis results.

(Refer Slide Time: 26:08)



Handling Aperiodic and Sporadic Tasks

- It is difficult to assign high priority values to sporadic tasks:
 - A burst of sporadic task arrivals could overload the system:
 - Cause many tasks to miss deadlines.
 - Violate basic RMA premises
 - Low priorities can be accorded:
 - But some sporadic tasks might be critical.
 - The **aperiodic server technique** may be used.

158

The next question comes is that how do you handle aperiodic and sporadic tasks. If you remember that aperiodic tasks are tasks that arrive randomly, but then they do not have a deadline whereas, this sporadic task which arrive randomly, but they have some deadline before we reach they must be complete.

If we give priorities to sporadic tasks then it would make our real time scheduler unusable because many of the slots will go empty because this sporadic tasks arrives randomly and it may lead to a situation where the system is cannot schedule the regular periodic tasks. And, also the sporadic tasks may occur randomly in close succession and therefore, it will make the lower priority tasks meet miss their dead line. So, the solution to this is a aperiodic server technique.

Now, we are almost at the end of the lecture hour. So, we will just look at the aperiodic server that is how to handle the sporadic and aperiodic tasks within the rate monotonic framework in the next lecture.

Thank you very much.