

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

Lecture – 12
Further RMA Generalizations

Welcome to this lecture. So far, we have looked at some basic aspects of real time operating systems and then we said that real time scheduler, task scheduler is a vital part of the real time operating system. And we have been looking at some of the real time tasks scheduling algorithms. The event driven schedulers are sophisticated schedulers compared to the clock driven schedulers and rate monotonic algorithm to be to for being useful in different practical situations.

In the last lecture we are looking at the deadline monotonic algorithm, where the task period and deadline are not identical. Today we will look at some more generalizations before starting to discuss about some of the issues so far, we have simplified assuming that tasks are independent and so on. They do not share resources. So, that under this simplified assumption we have been discussing so far. So, that assumption will relax after discussing the rate monotonic algorithm generalizations. So, let us get started.

(Refer Slide Time: 01:39)



**Further RMA
Generalizations**

(Refer Slide Time: 00:43)

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 was proposed by Sha and Raj Kumar --- **period transformation (1989)**.

156

One important thing that occurs frequently in designing and implementing real time applications is that some of the tasks which are very critical tasks, may not have the highest repetition rate, or the lowest period. There may be other tasks which are less critical or maybe non-critical which have high repetition rate and due to the rate monotonic priority assignment which says that assign highest priority to the task that has the highest frequency of repetition, or the lowest period; by that assignment even a very critical task may get assigned a low priority and therefore, whenever a low priority task sorry low critical task gets delayed the high critical task may miss its deadline.


One simple solution to this is to just rise the priority of the critical task, but then that makes all the results that we discussed Liu Lehoskies etcetera become inapplicable now let us look at how to handle this situation. So, there are often situations where the task criticalities are different from the task priorities and if we simply adjust the priorities and violate the rate monotonic priority assignment then the results that we so far heard on rate monotonic analysis would not hold.

To overcome this situation a solution was proposed by Sha and Rajkumar in 1989. The name of the technique is period transformation technique.

(Refer Slide Time: 04:11)

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 .




Let us look at the period transformation technique; the main idea here is very simple. A critical task having high period long period is a split into many smaller subtasks depending on to what extent we want to raise it is priority. If you want to really raise it is priority then we split it into many subtasks such that; it becomes the new period becomes lower than the other tasks.

Now, let us assume that we have a task T_i and we know that it is a highly critical task and then we split it into k sub tasks. So, the execution time for each of the sub tasks become e_i/k . Where e_i is the execution time for the task T_i and d_i is the deadline and the deadline for each of these sub tasks becomes d_i/k .

(Refer Slide Time: 05:47)

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 physical task itself.




But one thing we need to understand is that we are not changing anything in the code the task etcetera. This we are doing only for helping our with analysis without violating the rate monotonic constraint so that we can apply the rate monotonic analysis techniques that we have discussed so far.

We are just virtually considering the tasks to be split into k sub tasks, and then the priority of this task increases and, but the total execution time of all the sub tasks together hard up to the actual time.

(Refer Slide Time: 06:32)

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




Just to give an example let us say we have task T 1 whose period is 20 millisecond requires 5 millisecond execution time. The task T 2 has requires 8 millisecond execution time, but has a period 30 millisecond. In the rate monotonic priority assignment T1 will be assigned highest priority and T 2 a lower priority. Now assume that T 2 is a critical task and T 1 is not so critical. So, we want that even if T 1 gets delayed due to some reason, T 2 should not miss its deadline, in that case we split T 2 virtually into 2 sub tasks. T 2 a we can consider it to which has one sub task which has high repetition rate, but low execution time so; that means, the we have split T 2 into T 2 a whose execution time is 4 and deadline is 15 millisecond.

Whose execution time is 4 and deadline is 15 millisecond. So, the overall task characteristics has not changed, it still it is 8 millisecond execution time every 30 millisecond. Only thing is that for our analysis purposes we have assigned a priority to the task T 2 that is higher than T 1 and we have considered that the execution time is 4 and deadline is 15. And with this assumption we can still apply the rate monotonic analysis results.

(Refer Slide Time: 08:49)

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.
 - **Periodic server technique may be used.**



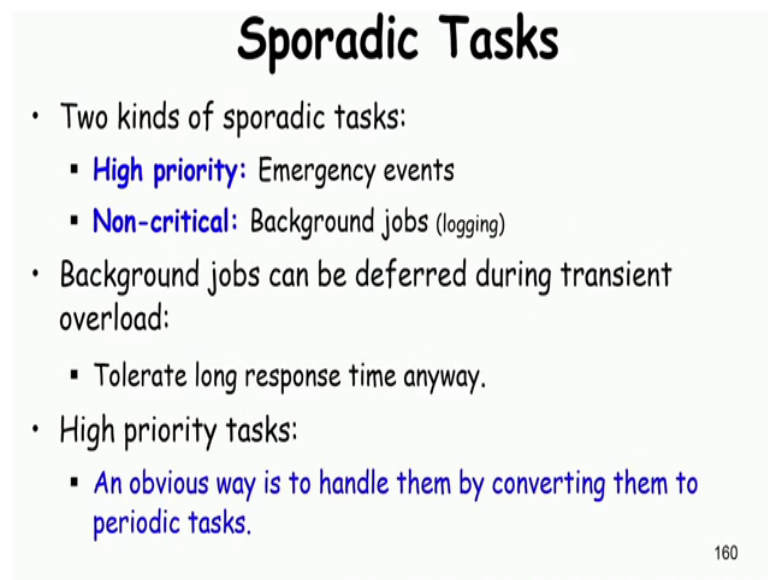
Now let us look at another issue which is handling aperiodic and sporadic tasks in the rate monotonic priority assignment.

We had so far looked at only to periodic tasks. And based on their period we assign priorities during that design time, but what if there are aperiodic and sporadic tasks. The aperiodic and sporadic tasks they do not occur periodically rather they occur randomly we cannot really approximate them to be periodic tasks with some period, because there can be bursts of

sporadic tasks, and once we get bursts of this task if you assign we have a assigned higher priority to these tasks then the task that are lower than this priority would miss deadlines.

And also, when the sporadic aperiodic tasks do not occur then we are unnecessarily under loading the system. To overcome this situation the periodic server technique has been proposed let us look at the periodic server.

(Refer Slide Time: 10:26)



Sporadic Tasks

- Two kinds of sporadic tasks:
 - **High priority:** Emergency events
 - **Non-critical:** Background jobs (logging)
- Background jobs can be deferred during transient overload:
 - Tolerate long response time anyway.
- High priority tasks:
 - **An obvious way is to handle them by converting them to periodic tasks.**

160

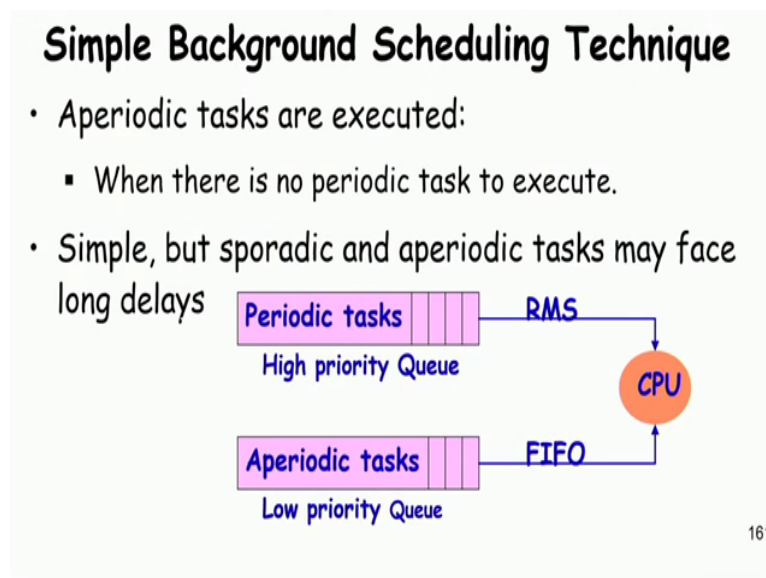
Before that let us look at some aspects of the sporadic tasks. So, there are 2 types of sporadic tasks some are high priority for example, in emergency event like let us say there is a fire detection reported and the task needs to be started to report fire conditions and also to start the water sprinkler. So, this is a very random event occurs very rarely, but then this is a high priority event.

It should not get delayed, but then there are non-critical tasks which can be sporadic. For example, let us say logging of the results. So, this also once in a while need to log the results and such non-critical tasks like logging results etcetera. They can be they can accommodate some delay in execution they can tolerate long response times. So, during a over load situation the background jobs can be differed, but for high priority tasks like emergency events we should be able to execute in the rate monotonic framework and should be able to high priority occurred high priority tasks.

So, that these meet their deadline a very bears way is to convert this high priority sporadic tasks into aperiodic tasks. For example, if it is a fire alarm handling task then we find out what is the maximum deadline that will be permitted in handling this task and then we consider it to be a periodic tasks with the period is equal to the deadline that is acceptable, but then there may be many types of such tasks many tasks which occur very rarely, but are high priority.

And also, if we use this simple solution then there can be a design problem that we have the system that is highly under loaded. Can we do can we handle this sporadic tasks in a better way

(Refer Slide Time: 13:24)



Before looking at that the periodic sever technique. Let us look at first handling the non-critical tasks which are the we call as background tasks. We can easily handle the background tasks in the rate monotonic framework by keeping the real time tasks in a queue and apply the real time rate monotonic scheduling under, and whenever there are no real time tasks high priority tasks to execute the CPU idle and then we have another FIFO scheduler.

where the aperiodic tasks have been queued it just takes from the queue in a first in first out and executes this low priority tasks and see that there is no guarantee when will this tasks gets executed, but as and when the CPU becomes available the other high priority tasks are not executed the FIFO scheduler becomes active and starts executing a aperiodic tasks and as


soon as periodic tasks come the aperiodic tasks are preempted. So, handling background tasks in simple.

In addition to the rate monotonic scheduler we use a FIFO scheduler for handling the aperiodic tasks.

(Refer Slide Time: 15:13)

Periodic Server

- A periodic server is a:
 - High priority periodic task
 - Created to handle multiple sporadic tasks that have deadlines associated with them.
 - The period and execution time is decided based on the characteristics of the sporadic tasks.




Now let us see how to handle high priority sporadic tasks more efficiently. Then just considering each high priority sporadic task as a periodic task with period is equal to the acceptable deadline. First let us look at the periodic server a periodic sever is a high priority tasks and it and handles multiple sporadic tasks, that have deadlines associated with them and the period.

This periodic server is decided based on the different sporadic tasks that it handles the assumption here is that the sporadic tasks occur very rarely. For example, fire handling events and so on and therefore, a single periodic server can handle multiple sporadic tasks because at anytime it is expected that only one or.

(Refer Slide Time: 16:36)

Periodic Server

- A periodic server is a:
 - High priority periodic task
 - Created to handle multiple sporadic tasks that have deadlines associated with them.
 - The period and execution time is decided based on the characteristics of the sporadic tasks.
- There can be multiple periodic servers in a system.




So, will occur but then we can also have a design where there are multiple periodic servers where we have from design considerations we have clubbed. Different sporadic tasks with similar characteristics into an assign them to one periodic server.

And another periodic sever handles tasks with different similar characteristics.

(Refer Slide Time: 17:09)

Types of Periodic Servers

- Various types of periodic servers have been proposed:
 - **Static Servers:**
 - Polling Server
 - Deferable Server
 - Priority Exchange
 - Sporadic server (POSIX-RT)
 - **Dynamic Servers**
- Slack stealer



Considering the practical importance of the periodic servers lot of work has lot of developments have taken place in the periodic server technique. There are static servers and dynamic servers. The static servers as the name says they have static priorities dynamic


servers are have dynamic priorities and then we have the slack stealer which we had already seen that whenever there is a slack time slack time the scheduler like a FIFO scheduler gets active and executes any pending background jobs.

There are several types of static servers the polling server deferrable server priority exchange and sporadic server. The sporadic server is the one which is supported the po6 real time standard the polling server is possibly the simplest among them and we will discuss the polling server in the next few minutes. The other ones the deferrable server priority exchange and the sporadic servers are simple improvements over the polling server. So, we look at only the basic concept here and if necessary you can read the text book that is we have we had refereeing for the other types of static servers.

(Refer Slide Time: 18:58)

Polling Server

- If there are no sporadic tasks at an invocation of the server (as per RMA):
 - The server suspends itself --- gets invoked again at its next period.
- If there are enough sporadic tasks at an invocation,
 - It serves up to e_s time.




The polling server is a periodic tasks is a periodic task and it is assigned a priority according to the rate monotonic algorithm and the period of this polling server is decided by based on the characteristics of the periodic tasks. The sporadic tasks it handles, but then compared to a simple server periodic server which is just keeps on repeating and even if there is CPU there is no sporadic task it just idles the CPU time a polling server suspends itself. So, that the other tasks can execute, but then if there are sporadic tasks during a invocation of the polling server because the polling server is invoked. A time t_s which is the period of repetition for the polling server during any invocation there is a sporadic task then it serves up to e_s time. It

should be clear that the polling server has a period P_s and execution time e_s . So, during the rate monotonic analysis we consider the utilization due to the polling server to be e_s / P_s .

(Refer Slide Time: 20:50)

Polling Server: Schedulability Analysis

- Include T_s in the task set and perform schedulability test
 - Schedulability of periodic tasks decreases
- Poor response time for aperiodic tasks



Now the schedule ability analysis become easy when we handle sporadic tasks through a polling server we just consider the polling server to be a task with execution time e_s and period P_s , but then since un on every occurrence there may not be enough sporadic tasks to run. The schedule ability of the other tasks which are lower priority than the polling server become less.

(Refer Slide Time: 21:40)

Polling Server: Schedulability Analysis

- Schedulability analysis involves
 - Schedulability of periodic tasks
 - Schedulability of sporadic tasks
- Schedulability of periodic tasks:
 - Introduce a periodic task corresponding to the server.

$$\sum_{i=1}^{n} (C_i / P_i) + (C_s / P_s) \leq (n+1)[2^{1/(n+1)} - 1]$$

166

Now, let us look at the schedule ability analysis how to perform we assume that the execution time each c_s or e_s or it is the notation here used is c_s .

Required every P_s time. So, the utilization due to the polling server is c_s by P_s for all other tasks it is $\sum_{i=1}^n c_i$ by p_i and therefore, the total utilization for all tasks including the polling server is given by the left-hand side expression and if there are n other tasks and 1 polling server then we have total $n + 1$ tasks and in the (Refer Time: 22:41) sorry the lui lay lands formula. We just use $n + 1$ into 2 to the power n by $n + 1$ minus 1. So, as long as this is satisfied, then along with the polling server the other tasks will become schedulable, but then how do we fix the deadline or given the deadline of the sporadic tasks?

(Refer Slide Time: 22:57)

Polling Server: Schedulability Analysis

- **Providing Sporadic task Deadline guarantee:**
 - Sporadic task A_i , arrived at t_a , with computation time c_a and deadline d_a .
 - In the worst case, may have to wait for one period before receiving service,
 - if $c_a \leq c_s$ the request would certainly be completed within two server periods.
 - $2P_s \leq d_a$

How do we fix the period of the polling server? We use a very common intuition here that the worst case for a sporadic event occurs just after if it occurs just after the time period of a pulling server is over. So, the polling server was getting this slot for execution, but towards the end of it just before it completes. The sporadic task occurred, now the sporadic task cannot be served here in this time slot because it has occurred towards the end of this.

It can only be taken up in the next slot that the polling server gets which is after P_s time, but then the polling server may not get the time at the time instance 0 from this period it may not immediately get served. So, it will get served by the rate monotonic scheduler before it is period P_s . So, it can be anywhere here and therefore, if the sporadic task a requires a

execution time which is c_a and c_a is less than c_s then it can be served in one instance by the polling server.

Therefore, in this worst case the deadline must be greater than $2 P_s$ because one P_s is just missed here. So, only it can be taken over in the next period where it gets a time slot for execution, but then there is no guarantee that it will immediately be taken up it may be taken up towards the end also end of the period. So, the deadline must be greater than $2 P_s$ or we can fix P_s as less than half the deadline let me repeat that.

To assign the period to the polling server we will look at all the sporadic tasks it needs to serve. We find the task that has the lowest deadline and then we take half of that and the polling server should have a period which is less than half the deadline of the sporadic task having the shortest deadline. Since, we were running out of time now we will stop here and we will look at few more issues before we look at the resource sharing among real time tasks.

Thank you.