

**Real Time Systems**  
**Professor. Rajib Mall**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Lecture No. 22**

**Handling Aperiodic and Sporadic Tasks in Rate Monotonic Scheduling**

Welcome to this lecture in the last lecture, we are discussing about some generalization of the rate monotonic scheduler, how to make it applicable with different constraints, the constraints that we considered is that what if the execution time is so small that it becomes comparable to the context switch times, what if the task self-suspense and what if the most critical task has a long period and it is the lowest priority task and so, on.

We saw a few simple techniques and also how to use the analysis the rate monotonic analysis, how can it be suitably modified to take care of those principles like self-suspension, context switches and the period transformation where we raise the priority of a critical task with long period to become a high priority task. Now, there are few other issues with the rate monotonic scheduler. Now, let us discuss those first.

(Refer Slide Time: 1:52)

**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.

*Handwritten notes:* ① Admission Control,  $P_i$

The slide includes a video inset of Professor Rajib Mall and logos of IIT Kharagpur and the Department of Computer Science and Engineering.

We had said that the aperiodic and sporadic tasks these are they are in any non-trivial real time system, for example, the user queries of the system parameters, user intervention and sporadic tasks like alarms or the system logging and so on. In cyclic scheduler, these are not possible to handle, it is very difficult because otherwise the cyclic scheduler will become extremely complicated and the main advantage of the cyclic scheduler is that it is simple.

And if we make it complicated with letting it support context switches and so on will defeat the very purpose of having a cyclic scheduler. We could actually use the rate monotonic scheduler. So, let us see how the periodic and sporadic tasks are handled in a rate monotonic scheduler.

One thing is that we must realize that we can just assign a high priority value to a sporadic task because the sporadic tasks are random in nature, and suddenly a burst of sporadic tasks could arrive and then our periodic high priority tasks would miss the deadline. How do we handle the situation that sporadic tasks have deadline and this may be critical like alarm and so on.

And we cannot just assign them a very high priority value. And also, just assigning a high priority to the sporadic tasks violates the rate monotonic analysis premises. Of course, we can assign very low priorities without discuss disturbing the other high priority periodic tasks, but that may not solve serve the purpose because some sporadic tasks might be critical.

One way we can handle this is that we have a admission control mechanism, the admission control mechanism at the sporadic tasks is that since these occur randomly there is a chance that many of these sporadic tasks may occur in a very soft interval, there is a burst of sporadic tasks.

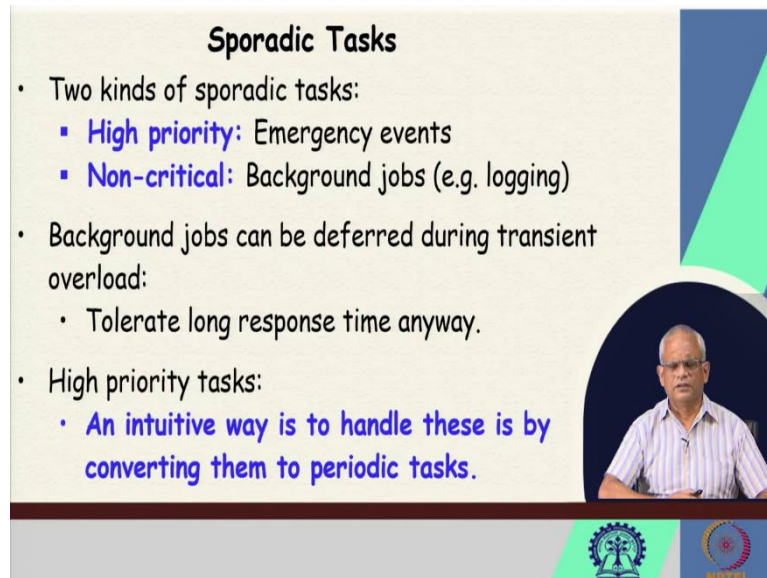
In that time, we will, the admission control will allow only one of the tasks and it will space them out. So, the purpose of the admission control is that we will assign a small period to the sporadic task, let us say  $p_i$ . Now,  $p_i$  is small enough to give a very reasonable priority value to the sporadic task. And let us say in  $p_i$  interval, only one periodic task arises, so that is fine. But let us say 2-3 periodic tasks or 5 periodic tasks occurred, then the admission control will space them out and release them one  $p_i$  interval. So, that is the best we can do.

And normally, many sporadic tasks occurring within a set interval is not very high. But this is the best we can do. But the other possibility, so the first one is we can use the admission control and we set a small period value even though these are random arrival, but we assign a small period value to the task so that we can do the rate monotonic analysis and we have this admission control mechanism which if there are multiple occurrence of the sporadic task, it will just admit only one of them and space out the other one at interval a  $p_i$ .

The second option is to use the periodic server technique. The periodic server is a very popular technique used for sporadic and aperiodic tasks. Now, let us just investigate the

periodic server which lets it possible to handle aperiodic and sporadic tasks in a rate monotonic scheduling environment.

(Refer Slide Time: 7:42)



**Sporadic Tasks**

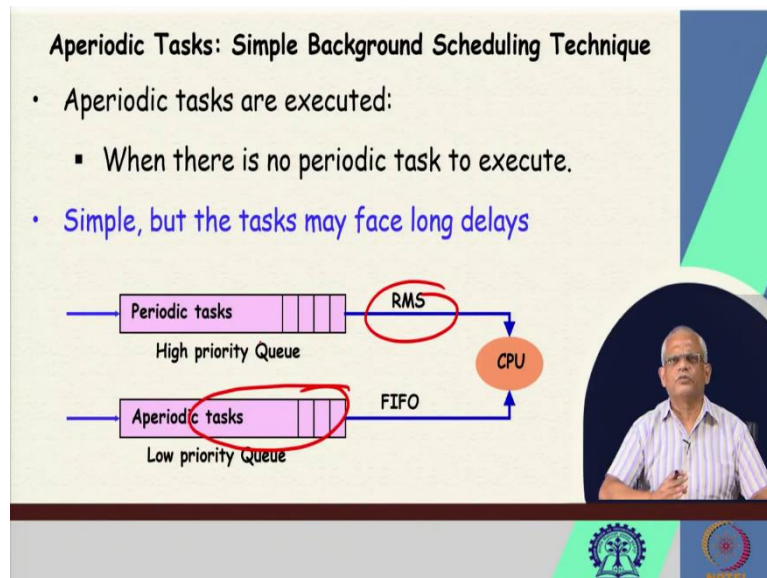
- Two kinds of sporadic tasks:
  - **High priority:** Emergency events
  - **Non-critical:** Background jobs (e.g. logging)
- Background jobs can be deferred during transient overload:
  - Tolerate long response time anyway.
- High priority tasks:
  - **An intuitive way is to handle these is by converting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

The slide features a video inset of a man in a striped shirt speaking. At the bottom, there are logos for IITM and another institution.

The sporadic tasks if we really analyze there are two types one are very high priority like emergency events, alarms and so on. Then we have non critical tasks like background jobs like logging etc. The background jobs can be deferred, these tolerate long responses anyway, but the high priority tasks need to be handled as quickly as possible in a very small interval of time needs to be handled and the time interval is typically specified that once these high priority tasks needs to be handled within this time.

As we were discussing, that we can assign a small period to this and have admission control mechanism we by chance, multiple occurrences a day or in a small period it will admit one and space out the other in they will make it wait and after the next period and so on to space them out.

(Refer Slide Time: 9:05)



Now, let us look at the simple background scheduling technique for aperiodic tasks. One way we can handle the aperiodic tasks is that when there are no periodic tasks to execute, the CPU is idling. We can let aperiodic tasks that are aperiodic tasks are typically user commands, user queries and so on.


But the problem with this approach is that the aperiodic tasks can face long delays because it may be a user command to sort of the system or it may be a user query and based on that the user would take some action and we cannot really make it face long delays, but if we are prepared for that, that the commands that the user give are not very critical, then we can have this technique, where you can have a simple background scheduling.

The high priority tasks are handled by the rate monotonic scheduler and the aperiodic tasks; they are queued in a low priority queue. And whenever there is idle instance, the low priority tasks are handled in a FIFO manner and they are assigned a CPU and as soon as a high priority task sorry periodic task gets ready the aperiodic task will be preempted.

So, this is a simple arrangement. But the problem with this arrangement is the long delay and unpredictable delay for the low priority aperiodic tasks, if there is suddenly a lot of periodic tasks, then the response time for the aperiodic tasks will be very low, but it can sometimes be very fast also, if the periodic tasks are not there at that moment.

(Refer Slide Time: 11:34)

**Periodic Server**

- A periodic server is a:
  - High priority periodic task 
  - Created to provide better response times to multiple of aperiodic tasks.
- There can be multiple periodic servers in a system.

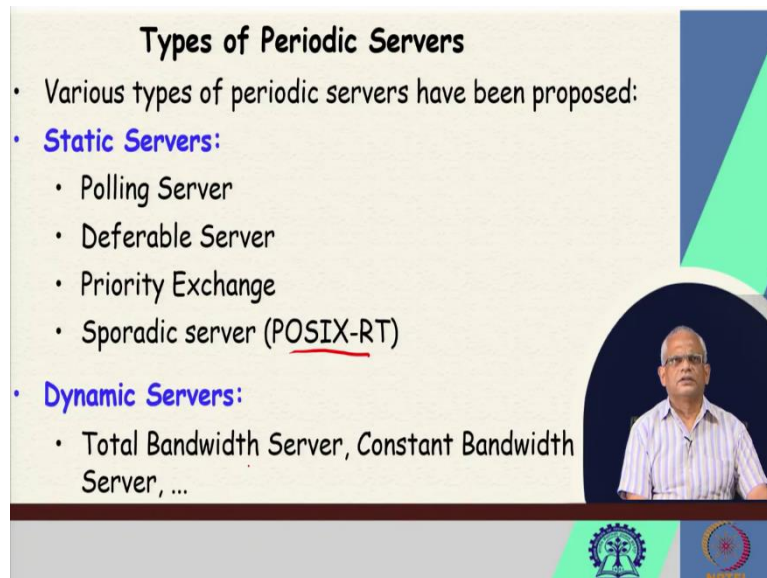
The slide features a hand-drawn diagram in red ink. It shows a horizontal line representing a timeline. A red circle is drawn above the line, with a red line extending from its center to the right, crossing the horizontal line. Below the horizontal line, there are several vertical bars of varying heights, representing a queue or a set of tasks. The slide also includes a video inset of a man in a striped shirt and two logos at the bottom right.

Can we do better? Can we improve the response time of the aperiodic tasks assuming that there are some commands or the user queries based on which some important decisions to be taken and we cannot just let it run and sometimes it gives fast response and sometimes very poor response, can we have a predictable response for the aperiodic tasks?

So, that is achieved with the use of a periodic server. Now, let us look at the periodic server technique. A periodic server is a high priority task. So, we create a high priority task and in this task, it is a periodic task. And here we look at the low priority aperiodic queue. And then depending on the time slot available, execute some of the aperiodic tasks.

And of course, there can be multiple periodic servers in a system; some periodic server has a very high priority. This serves the asynchronous the periodic tasks which have a very high priority, and then there is a low priority periodic server which caters to aperiodic tasks which are rather low priority.

(Refer Slide Time: 13:26)



### Types of Periodic Servers

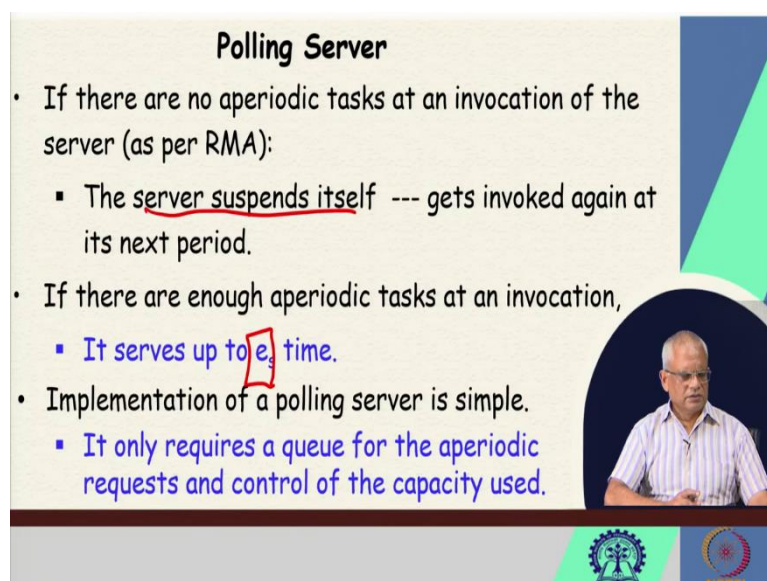
- Various types of periodic servers have been proposed:
- **Static Servers:**
  - Polling Server
  - Deferable Server
  - Priority Exchange
  - Sporadic server (POSIX-RT)
- **Dynamic Servers:**
  - Total Bandwidth Server, Constant Bandwidth Server, ...

The slide features a speaker's video inset on the right side and logos at the bottom.

The periodic servers are a popular technique to handle the aperiodic sporadic tasks and many types of periodic servers have been proposed. One category is the static servers. And here we have polling server, deferrable server, priority exchange and the sporadic server which is used in the POSIX real time later we will look at the POSIX standard, POSIX real-time standard and the operating systems that are commercially available.

Most of them are complainant to the POSIX real time standard. And then we can have the dynamic servers like total bandwidth server, constant bandwidth server and so now let us first look at the polling server.

(Refer Slide Time: 14:30)



### Polling Server

- If there are no aperiodic 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 aperiodic tasks at an invocation,
  - It serves up to e time.
- Implementation of a polling server is simple.
  - It only requires a queue for the aperiodic requests and control of the capacity used.

The slide features a speaker's video inset on the right side and logos at the bottom.

The polling server is a high priority task. Now, according to the rate monotonic scheduling, the polling server will be invoked, depending on its priority. Now the polling server once it is invoked, it checks what the tasks are pending in the queue that is assigned to it. If there are tasks that are waiting, it executes them.

But what if there are no aperiodic tasks in invocation of the polling server? The polling server was scheduled by the rate monotonic schedule started running. But then once it tried to examine the queue assigned to it found that there are no tasks. And if it just waste, its computation time, that is not good.

So, the server suspends itself. We know self-suspension, how the self-suspension is achieved by waiting for event and so on. And here, once it is suspended, just forgoes its time slot. Now other tasks, low priority tasks and the other polling servers start running and this polling server gets invoked again in the next period.

And if there are a lot of aperiodic tasks waiting to be served, then it is served them up to some constant time. That is the time assigned to it. Implementing a polling server is rather simple. A queue for it needs to be maintained where the aperiodic tasks, the aperiodic jobs are waiting. And then as its capacity is used, then it needs to stop. It suspends itself. So that is the control.

(Refer Slide Time: 17:05)

**Polling Server: Schedulability Analysis**

- Include  $T_s$  in the task set and perform schedulability test:
  - Of course, Schedulability of periodic tasks decreases
- Schedulability analysis:
  - Introduce a periodic task corresponding to the server.

*Uti Poll Server*

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

Let us do the schedulability analysis using the polling server. When we use the polling server, how does it affect the schedulability? The polling server becomes another periodic task and therefore the schedulability of the aperiodic tasks that decreases and depending on the period

we assign to the polling server, the lower priority periodic tasks, their response time will increase.

And if they were schedulable earlier, it may so happen that after we created this polling server, they may become unscheduled. So, we need to do a schedulability analysis. Now let us see how do we do the schedulability analysis? We introduce a periodic task a periodic task corresponding to the polling server. .

And now our Liu Leyland equation becomes the following. In the utilization expression, we have a new term here, which is due to the polling server. The polling server runs per es as time every Ps seconds milliseconds. This is the period of the polling server es / Ps is the utilization due to the polling server utilization due to polling server.

And if the number of tasks was n now, with the introduction of a polling server, it becomes n + 1 and the Liu Leyland and expression becomes  $(n + 1)[2^{1/(n+1)} - 1]$ . If n was already high, then this remains almost the same 69 %, but the utilization increases.

And if n is small, like 3, 4 etc., then this term decreases due to the polling server and this term increases due to the polling service. So, the task set that was schedulable may become unschedulable with the introduction of a polling server and as I was saying that we can have multiple polling servers to handle various categories of aperiodic tasks, some aperiodic tasks are very high priority aperiodic tasks.

And for that we assign a very small period to the corresponding polling server. For the low priority aperiodic tasks, we create a separate polling server. And there we assign the period of the polling server to be large enough so that it gets a lower priority.



(Refer Slide Time: 20:25)

### Deferable Server

- Similar to polling server:
  - However, deferable server preserves its execution time till the end of its period, if no aperiodic tasks are ready.
- **Advantage:** Much better response time for aperiodic tasks compared to polling server.  $P_i$
- **Disadvantage:** More complex and violates the RM principle that the highest priority task runs when it is ready.

A variation of a polling server is the deferable server. It is like polling server. But here the deferable server preserves its execution time till the end of its period. In case of a polling server, we had said that if there are no aperiodic tasks when the polling server gets invoked, when the polling server self suspends itself and in the next invocation of the polling server, if there are any tasks, which have arrived in between, they are taken up for execution.

But here the deferable server it preserves its execution time till the end of its period. So, that means, if the polling server was invoked here and that time there was no aperiodic task waiting to be executed the deferable server self suspends and it waits for aperiodic task to arrive.

If the periodic tasks arrived somewhere here then the polling server deferable server becomes active wakes up here and it gets scheduled. So, it retains its execution slot or the time quantum till the end of its period, it just defers that instead of it getting lapsed at the beginning of its invocation by the rate monotonic scheduler, it just defers it until the task arrives. And if none of the task arrives, a periodic task arrives, then it again gets invoked on the next cycle next period by the rate monotonic scheduler.

And of course, it would provide a much better response time for the aperiodic tasks because if that aperiodic task arrived just after the polling the deferable server sorry the polling server was invoked, then it will not be invoked in this period it may be invoked in the next period maybe towards the end.

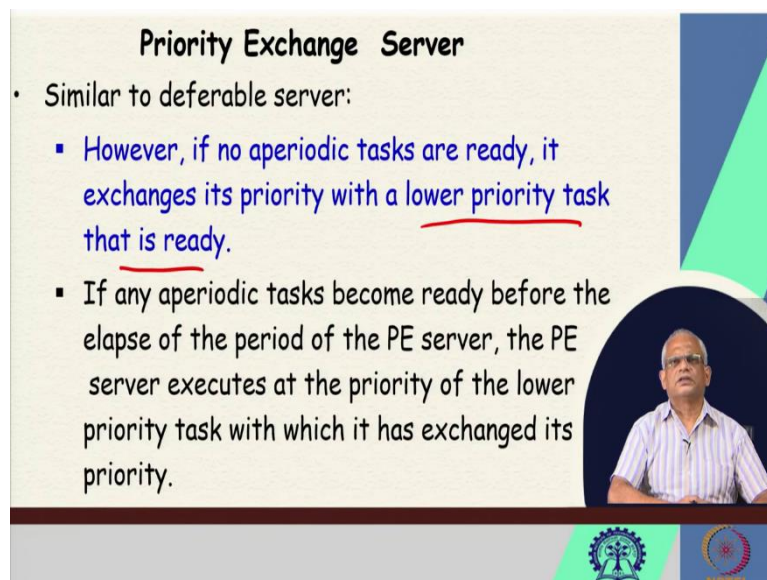
So, in the worst case, in the polling server, the delay that an aperiodic task can incur is  $2 \cdot p_i$ . On the other hand, in a deferrable server, at most it can get a delay of  $p_i$ , because it will be scheduled somewhere here less than  $p_i$  actually. So, it provides a much better response time compared to a polling server. The deferrable server provides a much better response time for the aperiodic tasks compared to a polling server.

But the main disadvantage here is that it becomes much more complex than the polling server finds that aperiodic tasks are not there self-suspends keeps on waiting for the aperiodic tasks to arrive. And then as they become arrived it becomes ready and it is scheduled. And it also violates the rate monotonic principle that a task is once it is invoked.

It can be invoked in its next period. And that is the basis by which the schedulability analysis is performed. If we relax this, then the schedulability analysis results become difficult to validate. And we will have to do a very conservative analysis, we need to put a lot of slack times, if the utilization is, let us say 70 %.

For the tasks to be schedulable, we might just keep it at 40 % or 30 %. So, that this change to the rate monotonic policy that we are doing will not make tasks miss their deadlines.

(Refer Slide Time: 25:45)



**Priority Exchange Server**

- Similar to deferrable server:
  - However, if no aperiodic tasks are ready, it exchanges its priority with a lower priority task that is ready.
  - If any aperiodic tasks become ready before the elapse of the period of the PE server, the PE server executes at the priority of the lower priority task with which it has exchanged its priority.

The slide features a video inset of a man in a striped shirt speaking. At the bottom, there are logos for IIT Bombay and IIT Madras.

Another variation of the polling server is the priority exchange server. It is similar to a deferrable server. But here, the task does not just self-suspend and wakes up again when the aperiodic tasks arrive. But here, if there are no aperiodic tasks, it just exchanges its priority with a lower priority task. So, a lower priority task becomes ready that is ready, it starts executing.

And it inherits the priority. It takes the priority of that lower priority tasks for that period. And as soon as aperiodic task arrives, it executes at that low priority, lower priority with which it had exchanged its priority. And here, the violation to the rate monotonic principle that used to happen with the deferable server does not happen here. Because it just changed, exchange the priority with the lower priority task and the lower priority task executed in its place.

We are at the end of this lecture. We will stop here. And we will continue from this point in the next lecture. Thank you.