Real Time Systems
Professor. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture No. 23
Handling Aperiodic and Sporadic Tasks in Rate Monotonic Scheduling (Contd.)

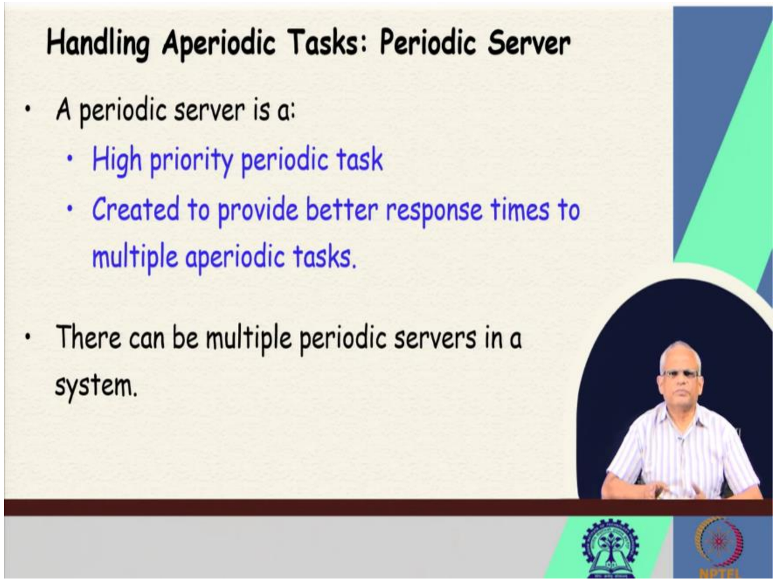
Welcome to this lecture, over the last few lectures, we have been trying to focus on the real time scheduling using rate monotonic scheduler. The rate monotonic scheduler is important scheduler, in real time systems used in an overwhelming number of applications and there are many issues associated with it, which we have been trying to investigate the different variations of the algorithm and then the different issues that can crop up.

At the end of the last lecture, we are trying to discuss how aperiodic and sporadic tasks can be handled in a rate monotonic scheduler. In the clock driven schedulers, it was very difficult to handle these tasks that are aperiodic tasks and sporadic tasks. But in the rate monotonic scheduler, we can handle such tasks and the simplest way to handle a sporadic task those which have stringent deadlines and critical tasks.

So, for that, in the last lecture, we said that we can convert a sporadic tasks into a periodic tasks and we fix a minimum separation between two sporadic tasks and that we call as the period of the tasks and we have admission control mechanism where if multiple such sporadic tasks arrive it spaces them out releases one task and queues the other one and releases other after that period of time which are fixed and for handling aperiodic tasks and other sporadic tasks.

The periodic server technique, the polling servers are very important technique we are discussing about it in the last lecture, let us continue from that point.

(Refer Slide Time: 2:31)



Handling Aperiodic Tasks: Periodic Server

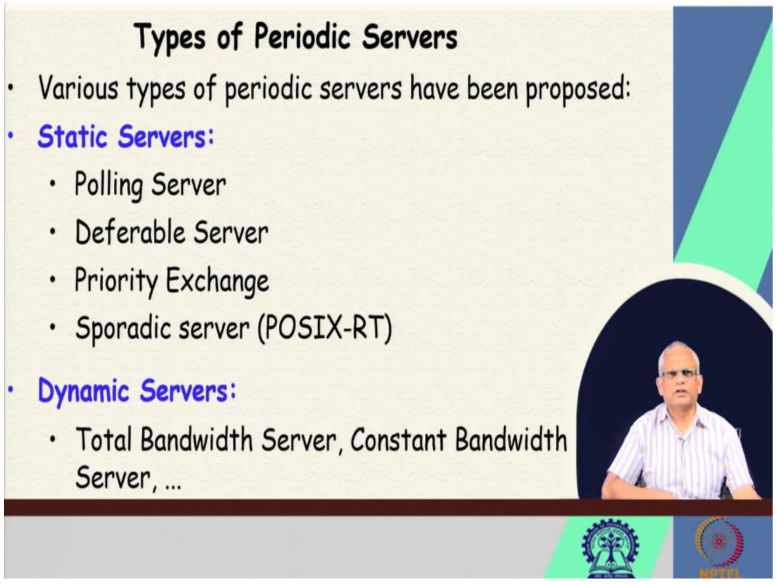
- A periodic server is a:
  - High priority periodic task
  - Created to provide better response times to multiple aperiodic tasks.
- There can be multiple periodic servers in a system.

The periodic servers are a mechanism by which you can handle aperiodic tasks and some sporadic tasks. Periodic server as the name implies, it is a high priority high priority periodic task and it is a server in the sense that this task serves many aperiodic tasks and without periodic server, the aperiodic task response time is very poor. But by using a periodic server technique, we can give reasonable or the required response times to aperiodic tasks and sporadic tasks.

And just to clarify, it is possible to have multiple periodic servers in a system. The rate of arrival of the periodic tasks that is the period of these periodic servers is determined by the priority of the aperiodic tasks. There are some aperiodic tasks which are very high priority. For example, an operator wants to check the configuration parameter and set the configuration parameter of a chemical plant.

So, that is a much more high priority task then let us says a task just logs the event or reports the routine statistics of the plant. So, to handle these different categories of aperiodic tasks, we can create multiple periodic servers some with very short period high priority tasks which cater to high priority aperiodic tasks and then we have lower priority cater to lower priority tasks.

(Refer Slide Time: 5:13)

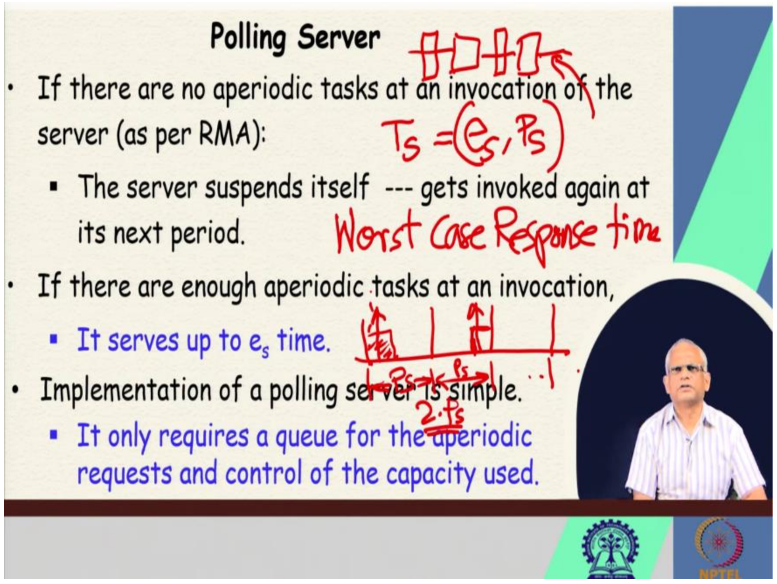


Types of Periodic Servers

- 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, ...

There are various types of periodic servers which have been proposed. Some are static servers, like the polling server, deferable server priority exchange and the sporadic server. We will look at this briefly and see the different situations in which this can be used. And there are also dynamic servers like total bandwidth server and constant bandwidth server etc.

(Refer Slide Time: 5:43)



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_s time.
- Implementation of a polling server is simple.
  - It only requires a queue for the aperiodic requests and control of the capacity used.

$T_S = (e_S, P_S)$
Worst Case Response time
P_S
$2 \cdot P_S$

Now, let us look at the one of the very basic periodic server called as the polling server. The polling server is a high priority task which is created to handle the aperiodic task it is a type of periodic server. And now, there are some aperiodic tasks which are assigned to this periodic server and there are periodic servers, sorry if there are such aperiodic tasks during a specific instance of the polling server job, which has been scheduled, if there are aperiodic tasks which are waiting, then the polling server will serve them as per its execution time period.

So, the polling server will have not only a period but the assigned execution time. Now, let us say in one instantiation in one instance of the polling server job, there are no aperiodic tasks, because aperiodic tasks have random arrival time. And it may so happen that in one invocation of the periodic server, there are no aperiodic tasks.

So in that case, the polling server will suspend itself. We know how a task can suspend itself can just wait for an event or something. And then, there is a context switch. Now next tasks execute, basically the polling server forgoes its allotted slot. And of course, the lower priority polling servers and other low priority tasks they can start executing. And the polling server that suspended itself in the next invocation after its period.

The next invocation again it takes it their aperiodic task to solve and therefore search them. But the question here is that what is the maximum response time or the worst case response time of an aperiodic task in a polling server. So, the polling server serves up e_s time. If there are a lot of aperiodic tasks which have come up, it cannot basically complete all of them. And if the polling server let us denote the polling server as T_s.

This is the polling server and it has an execution time of e_s and a period of P_s. Now the P_s will be our high priority for high priority polling servers that are the high priority aperiodic tasks. Now, let us assume, so we are trying to answer this question that what is the worst case response time of aperiodic task? Now let us see the timeline of invocation.

The period of the sporadic server is P_s. And during this instance, during this interval, one instance of the periodic server, that is T_s will be scheduled by the rate monotonic schedulers, it can get scheduled at the beginning of the interval, or towards the end of the interval and so on. Now, let us say the polling server is getting scheduled at the beginning of the interval. Now, the aperiodic tasks arrived just after the polling server has been scheduled.

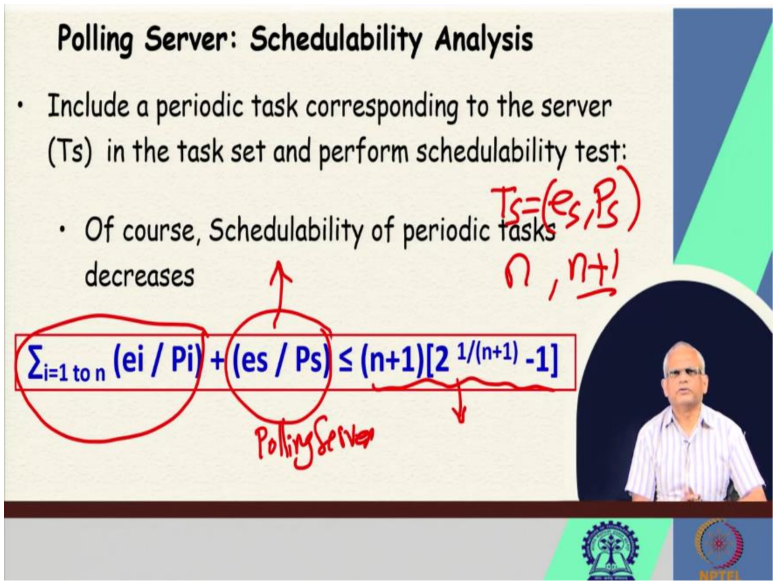
So, the polling server can, at best handle this aperiodic task in the next invocation, because here it has already started to execute it cannot handle and the next invocation of the periodic server, maybe towards the beginning of the interval, P_s or maybe towards the end of the interval, in the worst case, it can be towards the end of the interval.

And the task, the aperiodic task may get scheduled somewhere here. So, it just missed here just arrived just after it had started. And it got sort towards the end of the interval. So, 2 P_s is the worst case response time, of an aperiodic task using a polling server. And if we have aperiodic tasks, which have, let us say, we need to determine that the response time of this is let us say 10 seconds, then the period of the polling server need to be 5 seconds. To be able to give a worst case response of 10 second.

The polling server period needs to be 5 seconds. And this is one of the simplest periodic server. The implementation is rather straightforward. Here we need a queue where the periodic, aperiodic tasks are queued up. And the polling server will select from this queue. Depending on some criteria may be the priority of the periodic task. Or maybe just FIFO the first in first out.

And another thing that we need not only the queue, but there is a controller the capacity used, because these tasks have random arrival, it may so happen, that some invocation there are many aperiodic tasks which are queued up. And then, the polling server needs to stop after it has completed its quota of e_s time. And we need one mechanism for that. So that is it, the polling server is rather straightforward, just a queue and control mechanism for the capacity used.

(Refer Slide Time: 13:30)



Polling Server: Schedulability Analysis

- Include a periodic task corresponding to the server (Ts) in the task set and perform schedulability test:
  - Of course, Schedulability of periodic tasks decreases

$$\sum_{i=1 \text{ to } n} (ei / Pi) + (es / Ps) \leq (n+1)[2^{1/(n+1)} - 1]$$

Ts=(es,Ps)
n, n+1
Polling Server

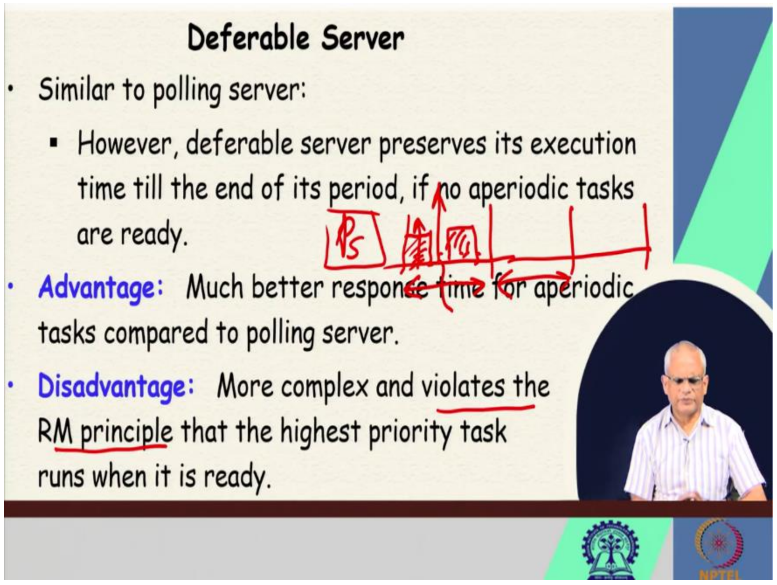
Now let us look at the schedulability analysis. If we handle aperiodic tasks using a polling server, will the periodic tasks get affected? Will their response times decrease? Or is it that the polling server does not affect the response times of the periodic tasks and none of the periodic tasks are going to miss their deadline if we introduce a polling server. Let us just investigate that. If we have a polling server, we have a periodic task corresponding to the server which is T_s, and $T_s = e_s$, P_s.

So, if we had n tasks earlier n periodic tasks with the polling server, we will have if we have one polling server, we will have n plus 1 tasks. And as I said, that we can have multiple polling servers then we might have n plus 2 or n plus 3 periodic tasks. So, the number of periodic tasks increases. That is the first thing with a polling server. And if we have a short enough period for assigned to the polling server, then the scalability of the periodic tasks decreases.

And if we use the Liu Leyland result the utilization due to all tasks if it is $\Sigma_{i = 1 \text{ to } n}$ there are n tasks ei / Pi summation 1 to n, this is the utilization due to the existing periodic tasks and now, we have created one more periodic task which is the polling server. And now, the bound under Liu Leyland becomes (n + 1) $[2^{1/n+1} - 1]$. So, on one hand the utilization increases due to the polling service and on the other hand the bound decreases because we have $2^{1/n+1}$ and therefore, the bound overall bound here decreases and here the utilization increases.

So, if a periodic task set was just schedulable, it may so happen that we cannot really introduce a polling server, because as long as we introduce a polling server, the bound will be violated and the tasks will no more be schedulable. It is possible, but if the utilization was low enough, we can introduce a polling server without really violating or exceeding the bound.

(Refer Slide Time: 16:55)



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.
- **Disadvantage:** More complex and violates the RM principle that the highest priority task runs when it is ready.

The polling server, we had seen that the response time for aperiodic tasks is a problem. In the worst case, the response time can be 2 P_s if P_s is the period of the polling server, but can this be improved. So, that is the idea behind a deferable server. It is similar to a polling server. Here again, there is a queue and control mechanism for the execution time. So, the aperiodic tasks, but here the way it works is that if there is an invocation of the deferable server, let us say that deferable server gets invocated at P_s interval somewhere between these intervals it will be scheduled by the rate monotonic scheduler.

Now, let us say that it got scheduled at the start of the interval. Because there are no other higher priority tasks, the rate monotonic scheduler, scheduled it at the start of the interval. Now, let us say the aperiodic tasks were not there. And they arrived a little later. This is the place where aperiodic tasks arrived. And the deferable server was scheduled here, but there are no tasks. And as a result, it would suspend itself.

But in a polling server, the polling server suspends itself. And it is again invoked in the next interval. But here, it does suspend itself waiting for the aperiodic tasks to arrive. So, whenever aperiodic tasks arrive, the polling server becomes active. If there are no higher priority tasks, then it starts executing, of course, the lower priority tasks would have to wait. So, maybe somewhere here, the deferable server would start to execute.

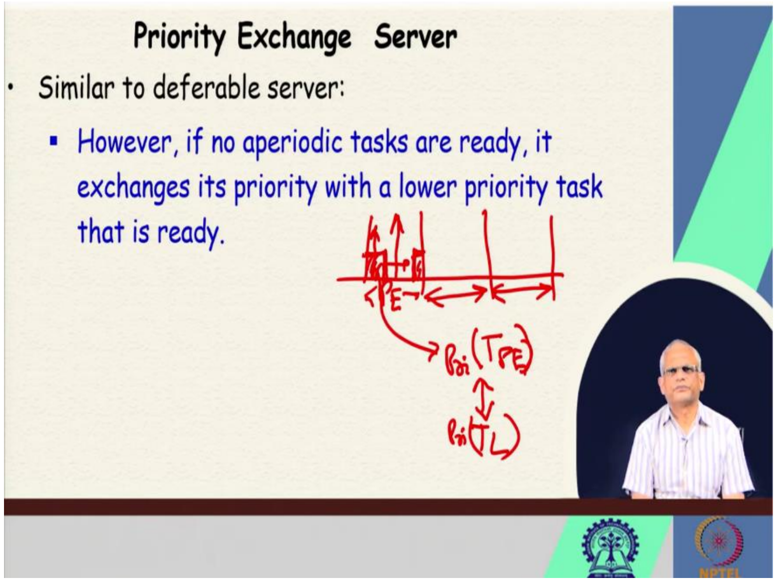
So, this is the difference between a polling server and a deferable server is that the deferable server suspends itself waiting for aperiodic tasks, whereas in a polling server, the polling server suspends itself only to be scheduled in the next interval. Now let us see the impact of the deferable server on the response time of the aperiodic tasks and also the impact of the deferable server on other periodic tasks.

So, definitely, it will provide a much better response time for aperiodic tasks. In the worst case, the response time of a, of an aperiodic tasks becomes P_s. Because even if they have not arrived, because these are of random arrival the aperiodic tasks. So, somewhere they arrive and they would be served before the line separates period P_s. So, and therefore, the worst case P_s is the time interval for which the aperiodic task would have to wait.

So, compared to a polling server, the response time there is a substantial improvement from 2 P_s we have response time P_s, but what about the disadvantage? The disadvantage is that it becomes much more complex and violates the rate monotonic principle. Because here the task needs to suspend itself waiting for an aperiodic task whenever there is an aperiodic task arrive, then the polling server sorry that deferable server needs to be enabled. And also, it violates the rate monotonic principle because it suspends itself when it was scheduled and later it reclaims its execution time, which is not in agreement to the rate monotonic principle.

And therefore, the schedulability of tasks which are of lower priority than the deferrable server can get affected they can, their response time may become large enough for them to miss their deadlines. So, this is a problem here. If the utilization due to the task set is low, we can use a deferable server. But again, the response times of the lower priority tasks can get increased by e_s. And that is a problem here.

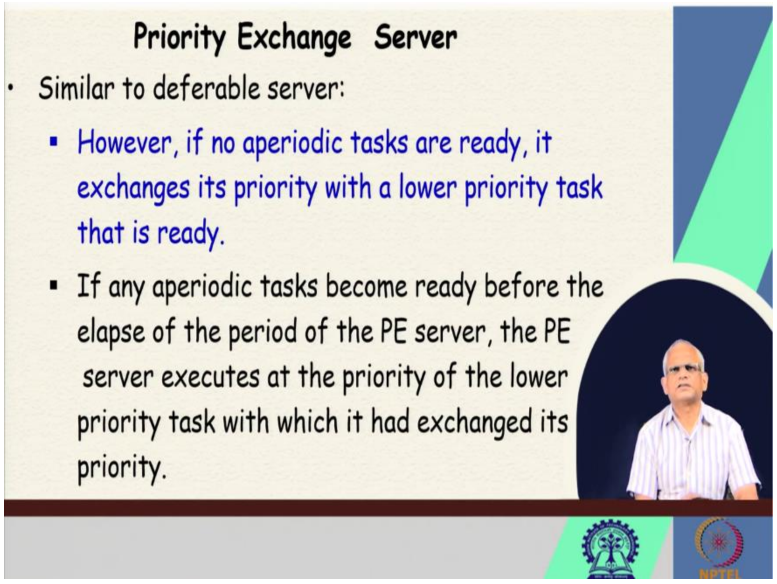
(Refer Slide Time: 22:17)



Priority Exchange Server

- Similar to deferable server:
    - However, if no aperiodic tasks are ready, it exchanges its priority with a lower priority task that is ready.

PE
Pri(TPE)
Pri(TL)



Priority Exchange Server

- Similar to deferable 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 had exchanged its priority.

Let us look at an improvement of the deferable server. Because the deferable server even though it had substantial advantages for aperiodic tasks, but for the periodic tasks, which are normally of higher priority, it is a problem. The deferable server can make them unschedulable or they may miss their deadlines. Now, let us look at an improvement of the deferable server called as a priority exchange server.

The priority exchange Server is similar to a deferable server. Again, when there are no aperiodic tasks ready during invocation of the priority Exchange server or the PE server it suspends itself. So, let me just explain with this diagram that if this is the priority exchange the period of the priority exchange.

So, during each of this period, it will get invoked once by the rate monotonic scheduler. Now, let us say it got scheduled at the beginning here and there are no aperiodic tasks here. So, the period the priority exchange server would suspend itself. And if later on some aperiodic task arise then the periodic the priority exchange server will be activated. But then it is priority. Ok let me just refine what I said that when it suspends itself, it gives its priority to one of the tasks which have been waiting.

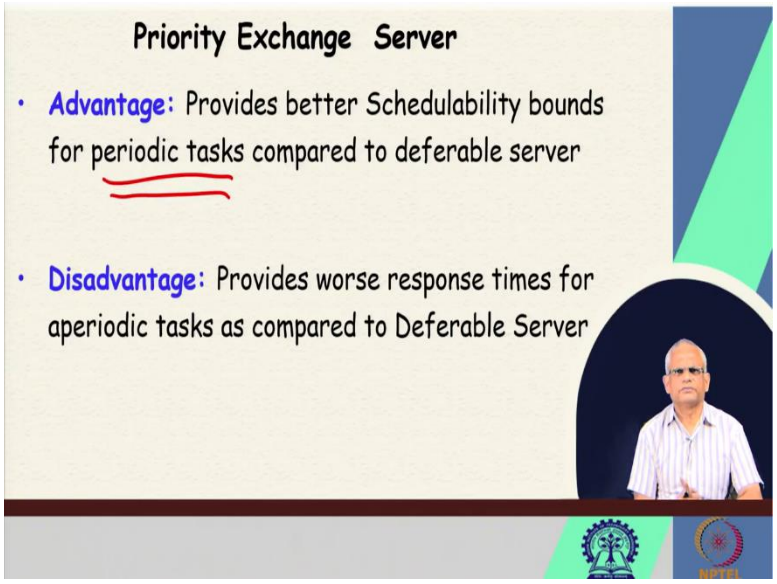
So, the priority exchange found that there are no aperiodic tasks and then it suspends itself. But while suspending the priority of the priority exchange server, let us say the priority is Pri(T_{PE}). So, this will be assigned to a lower priority task which was waiting so that it exchanges a priority.

The priority of the priority exchange server will be assigned to a low priority task the priority of a low priority task will be increased and the priority of the priority exchange server will decrease to the low priority. Now, later on when there is aperiodic task which arrives the priority exchange server will become active, but then it will not execute at its own priority it will execute the priority which it had exchanged with a low priority task.

And therefore, the rate monotonic principle is not violated very blatantly that it just suspended itself and start executing at again its own priority, it led to a lower priority task because it could not execute it assigned its priority to a lower priority task which started to execute at the priority of the priority exchange server.

And then later when aperiodic tasks arrived, the priority exchange starts to execute the priority of the low priority task and therefore, it may get scheduled a little later. So, the response times for the aperiodic tasks will be little worse compared to a deferable server. But then, it does not violate the rate monotonic principle as badly as the deferable server used to when aperiodic task becomes ready. The priority exchange server starts to execute at a priority of the lower priority task with which it had exchanged its priority.

(Refer Slide Time: 27:01)



Priority Exchange Server

- **Advantage:** Provides better Schedulability bounds for periodic tasks compared to deferable server
- **Disadvantage:** Provides worse response times for aperiodic tasks as compared to Deferable Server

The advantages of the priority exchange server is that it provides better bounds for the periodic tasks because it starts executing at a lower priority, it does not violate the rate monotonic principle and the periodic tasks will find that their response times is not much affected. But the disadvantages of course, worse response time for aperiodic task as compared to the deferable server. We are at the end of this lecture. We will stop here and continue in the next lecture. Thank you.