

Real Time Systems
Professor. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture No. 24
Coping Up with Insufficient Number of Priorities

Welcome to this lecture over the last few lectures, we have been trying to focus at various issues with the rate monotonic scheduler. The reason is that rate monotonic scheduler is possibly the most important scheduler in real time embedded system scheduling, if you are going to do programming and system development using an operating system, real time operating system, you are very likely to use the real rate monotonic scheduler.

And so, far we have been discussed several issues including the last issue that we discussed in the last lecture is about how to handle aperiodic tasks and sporadic tasks under a rate monotonic scheduler. And we looked at the periodic server's technique, there are various types of periodic servers, we looked at a few basic periodic servers, but then there are more sophisticated servers, which are of course, developed based on the basic servers.

Now, let us look at another issue that we encounter while developing a real time system using a rate monotonic scheduler.

(Refer Slide Time: 1:43)

Insufficient Number of Priorities

- All RTOS support a restricted number of priority levels. Why?
 - Higher scheduling overhead
- When the number of tasks is more than the number of available priority values:
 - The schedulability of the system decreases.
 - But, how to assign priorities to minimize the impact?

The slide features a video inset of Professor Rajib Mall in the bottom right corner. Handwritten red annotations include a vertical list of five boxes on the right side, with arrows pointing to the text 'more than the number' and 'assign priorities to minimize the impact?'. The words 'assign' and 'minimize' are underlined in red.

The issue is that of insufficient number of priorities later on in this course, we will look at various commercial real time operating systems, some are open source and some are available commercially. But then we will see that all these operating systems have certain

limited number of priorities. Some provide only 8 real time priority levels and some give 16 real time priority levels and so on.

But then, it is good to understand why they restrict the number of real time priority levels, we are saying that many of these commercial operating systems they have two bands of priority one is called as the real time priority, which is the static priority levels where we scheduled tasks using a rate monotonic scheduler, but then there is another band which is the dynamic band and there the other tasks are scheduled the periodic tasks and those with time bound and so on. They are scheduled using the real time priority levels.

Now, the number of real time priority levels are the static priority levels, these are fixed, but let us investigate why? Why do they fix why can they just make it 128 or 256 or something like that? If we investigate this, then the answer is that the efficiency of the operating system because if we have a number of priority levels, then what happens is the implementation we had seen is a multi-level feedback queue.

So, we have these various real time tasks, which are in every priority band in the, every priority level. There is a queue here and the tasks at that level they get queued at that level. Now, during an invocation and the scheduler at a scheduling point needs to check all these levels to find out if there is a task which can be scheduled.

Now, first is the providing so many queues like 128 if it is a priority level, we need to have 128 queues for the tasks for each of the priority level. And the second is that the scheduler needs to scan through all this and therefore, the response time of the scheduler deteriorates. And we know that many of these systems are very sensitive to time, their deadlines are in terms of milliseconds.

And if the operating system itself the scheduler itself takes a millisecond to run at every invocation at every scheduling point, if it takes several milliseconds to run, then the operating system becomes less usable for many applications, we cannot use because the operating system response time itself is bad. Later on in this course, we will see the benchmarking of these real time operating systems and there will see that this is a very important criterion to compare various real time operating systems is based on their response times.

So, we have to live with the situation that the real time operating system will provide us only with a limited number of priorities 8 or 16 or something like that. Otherwise, the priority the

scheduling overhead will be too much. Now, we might have the number of tasks in a non-trivial application may easily be 30, or 40, or something.

And for large applications, it may become hundreds, let us say a nuclear power plant, the number of tasks that are handling various types of sensors and so on conditions and so on, it may easily exceed 100. But the priority levels are some 8 or 16 or something. So, how do we handle the situation, because finally, what might happen is that many of the tasks even though they have different periods, we end up assigning them the same priority level.

And this is a violation of the rate monotonic principle, because the rate monotonic principle says that the priority of the task should be monotonically increasing in terms of its arrival time, or the rate the arrival rate or the inverse of the period. But we are forced to assign two tasks, which have different periods, the same priority level and this is a violation of the rate monotonic principle.

And therefore, even if the task set satisfies the Liu Leyland criterion, and so on, it may still miss its deadline. So, that is a very important problem. Now, how do we handle this situation of insufficient number of priorities, the situation is very real. In many applications, we will encounter this situation that the number of priority levels are less compared to the number of tasks that we have.

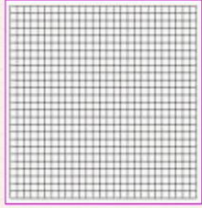
And unless we somehow tackle this problem, the system will become unschedulable. And we do not know how to do the schedulability analysis. And what is the let us say the scale the Liu Leyland and bound for a set of tasks is let us say 0.7. Now, we might conservatively have to do with the 0.3 utilization just because we know that we have not assigned our priorities based on the rate monotonic principle and therefore even if there is much less than 0.7 may still miss its deadline.

So, it is true that you will have to assign at least some of the tasks the same priority level, but which tasks How can we reduce the impact of this problem? So, that is the focus of our discussion now. How do we assign priorities to the tasks to minimize the impact of having to assign multiple tasks the same priority levels?



(Refer Slide Time: 9:57)

Priority Grid Approach

- Assign priorities using a priority grid:
 - Uniform scheme ✓
 - Arithmetic scheme
 - Geometric scheme
 - Logarithmic scheme ✓



60 tasks 8 prio 10



The usual technique that is used is a priority grid approach. We will use a grid like this. And then depending on the technique that we use, we will divide the grid into different slots. And we will assign the tasks to these different slots. We have different techniques for assigning priorities to tasks, namely, the uniform scheme, the arithmetic scheme, the geometric scheme, and the logarithmic scheme.

If the number of tasks exceeds the number of priority levels, very marginally, let us say we have 8 priority levels and let us say we have 10 tasks. So, only 2-3 tasks need to stay at the same priority level, then, the uniform scheme is simple, intuitive, and we can use it. But then, if the number of tasks are much more than the priority levels that are available, like we have 8 priority levels, and let us say we have 60 tasks, then the uniform scheme in that case, when the number of tasks is more will not provide a good schedulability for the tasks, the other techniques would be better and the best would be the logarithmic scale.

So, depending on our situation, whether we have the number of task is marginally exceeding the number of priority levels, or the number of tasks is much more than the number of priority levels that are available, we will use the different schemes. First, let us look at the uniform scheme is rather simple.

(Refer Slide Time: 12:23)

Uniform Scheme

- If there are N tasks and n priority levels, then:
 - $\lfloor N/n \rfloor$ number of task are assigned to each level. $N > n$ $\lfloor 10/8 \rfloor = 1$
 - Rest of the tasks are distributed among the lower priority levels. $\lfloor N/n \rfloor$

Diagram showing tasks T_1 to T_{10} grouped into priority levels.

In the uniform scheme, if there are n priority levels, N tasks and n number of priority levels, then we will assign N / n task to each level. N tasks, so number of tasks is more than a number of priority levels. So, N / n number of tasks are assigned to each level, let us say we have 10 tasks, but 8 priority levels. So, the $\lfloor 10/8 \rfloor = 1$.

So, first, we will assign let us say we have T_1 , to T_{10} . And we have arranged these in terms of their priority values. So, T_1 to T_8 , these will be assigned to different priority levels. Now there are two tasks that are left out T_9 and T_{10} . So, these will be distributed among the lower priority levels. But then we would really like the higher priority tasks to get a higher level. So, we cannot just distribute T_8 and then T_9 and T_{10} .

Because then T_9 will have higher priority than T_8 we do not like to do that. What we will do is we will have T_7 and T_8 share the priority and T_9 and T_{10} share the lowest priority. I hope that is clear here that we assign we take the ceiling of the number of tasks by number of priorities, because number of tasks are exceeding the number of priority levels if the number of tasks are less than the number of priority levels we have no problem.

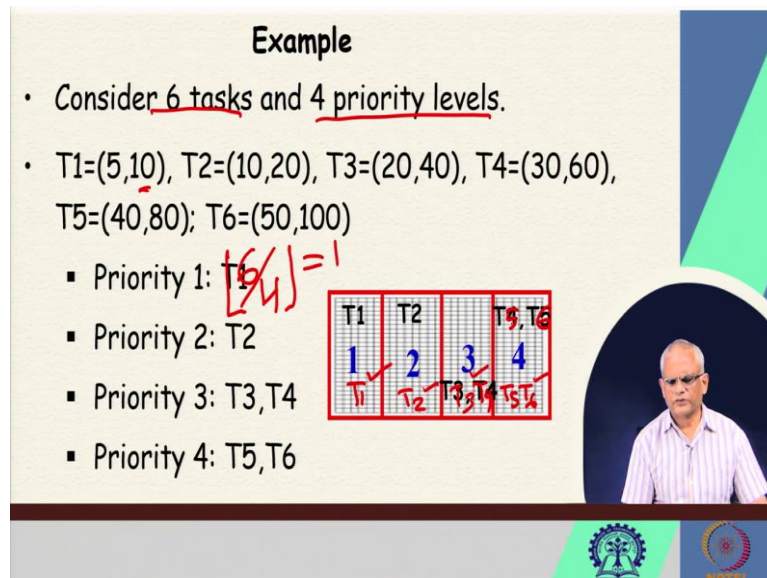
But here the number of tasks is exceeding the number of priority levels $N > n$ and $\lfloor N / n \rfloor$ and then that many we assigned to each level and the remaining we assign to the lower level. But to make sure that there is no priority violation, because if we fast distribute T_1 , T_2 to T_8 and then we assign T_9 to the 9th priority, then T_8 will be at a lower priority, we won't like to happen that and therefore, T_7 and T_8 will share a priority and T_9 and T_{10} will share another priority value. So, that is the essence of the scheme.

(Refer Slide Time: 15:30)

Example

- Consider 6 tasks and 4 priority levels.
- $T_1=(5,10)$, $T_2=(10,20)$, $T_3=(20,40)$, $T_4=(30,60)$,
 $T_5=(40,80)$; $T_6=(50,100)$
- Priority 1: T_1 $\left\lfloor \frac{6}{4} \right\rfloor = 1$
- Priority 2: T_2
- Priority 3: T_3, T_4
- Priority 4: T_5, T_6

T1	T2	T3, T4	T5, T6
1 ✓	2	3 ✓	4
T_1	T_2	T_3, T_4	T_5, T_6



Let us try to understand this with the help of an example. Let us say in some small embedded operating system, we have a rate monotonic scheduler that is implementable. But then we have only 4 priority levels supported by the operating system. But then, our application had 6 periodic tasks, the characteristics of the periodic tasks are 5 execution time, and 10 10, 20, etc, etc, 50, 100, etc.

Now, since we have 4 priority levels, some of the tasks we need to share the same priority levels, how do we assign the tasks to the priority levels by using the uniform scheme? Ok, First, we need to have $\lfloor 6/4 \rfloor = 1$. So, we can assign T_1 , T_2 , T_3 , and then there are two extra here. And they will be distributed to the lower priority levels.

So, we can just assign T_4 to the 4th level, and T_5 and T_6 will distribute because then T_5 will have a higher priority than T_4 . So, what we will do is we will let T_3 and T_4 share a priority level, and T_5 and T_6 they share another priority level. So, that is the main idea here. Priority one, we have T_1 , priority 2 is T_2 , priority 3 is T_3 and T_4 and priority 4 is T_5 and T_6 . So, very simple scheme.

But then the main idea behind the assignment of the priority levels to the tasks is that the higher priority tasks should not share their priority level to the other tasks or we have to minimize the number of tasks with which the higher priority tasks they share their priority levels otherwise they will be impacted.

Their scheduability will be impacted. And there may be severe errors in the system. We would like the lower priority tasks to share their priority levels. And the uniform scheme let

us do that distribute the excess number of tasks among the lower priority levels with the condition that lower priority task does not get a higher priority during our assignment, very simple scheme.

But it is better than a scheme where we randomly assign the tasks different priority values. So, T_1 gets the 1st priority T_2 get the 2nd priority T_3 and T_4 get the 3rd priority, and T_5 and T_6 get the 4th priority.

(Refer Slide Time: 19:09)

Arithmetic Scheme

- Consider N tasks:
 - $N = r + 2r + 3r + 4r + \dots + nr = r \cdot \frac{n(n+1)}{2}$

Handwritten notes: 32 tasks, 4th, 3-1, 8x2/4x5, 8x2/(n)(n+1)
 - r tasks having the shortest periods are assigned to the highest priority level,
 - $2r$ tasks are assigned the next highest priority level, and so on.

Handwritten notes: 3, 6, ..., n

Now another scheme is the arithmetic scheme. The uniform scheme works well when the number of tasks exceeds the number of priority level very marginally. But if the number of tasks is much more than the number of priority levels, maybe 10 times more or 8 times more and so on. Then one proposal is the arithmetic scheme.

Here, we assign the tasks, r number of tasks to the 1st priority $2r$ to the 2nd priority third, $3r$ to the 3rd priority and nr to the last priority. So, the last priority n times the number of tasks of the first priority, they share the priority level, if r is 2, if two tasks share the first priority, and n is 8, then we will have 16 tasks, sharing the last priority 4 tasks sharing the second priority and so on.

So, this maintains the intuitive idea that at the higher level of priority, too many tasks should not share their priority because this those are the critical tasks. Now, in this case, the uniform scheme does not do very well. Because if we have 8 priority levels, and we have 40 tasks, then each level gets 5 tasks, and then the higher level there is too much of sharing, it is not a good thing.

So, the arithmetic scheme would work better in that case, r tasks having the shortest period are assigned to the highest priority level and $2r$ to the next higher priority level and so on. So, we can simplify this as $r [n(n+1) / 2]$, or $r = (N \times 2) / (n(n+1))$. Let us take a hypothetical example. That we have 4 priority levels. And we have 32 tasks, 4 priority levels, and 32 tasks.

Now, the value of r in this case, will be $(32 \times 2) / (4 \times 5)$. So, that will be $8 \times 2 = 16 / 5 = 3.1$, r is 3.1. So, the first one will have 3 tasks, the second one, 6 tasks, and so on, but this is a fraction. And when it we can approximate it more than 0.5, let us say 3.6, we might assign 4 tasks and so on. So, the first one will get 3 tasks. And of course, we have to have this intuitive guideline that the higher priority tasks should have lower number of tasks sharing that priority, second one, 6, and so on. So, that is the way the arithmetic scheme works.

(Refer Slide Time: 23:14)

Geometric Scheme

- Consider N tasks:
- $N = r + r^2 + r^3 + r^4 + \dots + r^n$
- r tasks having the shortest periods are assigned to the highest priority level,
- r^2 tasks are assigned the next highest priority level, and so on.

Handwritten notes on the slide:

- 128
- $r(1 + r + r^2 + \dots + r^7)$
- $128 = r \cdot \frac{r^8 - 1}{r - 1}$

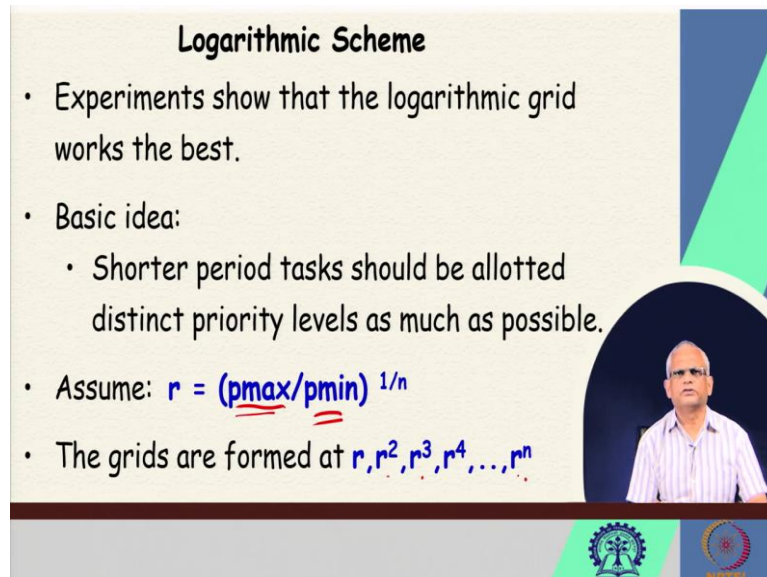
The slide also features a video inset of a man speaking and logos for IIT Bombay and IIT Madras at the bottom.

Now let us look at the geometric scheme. In the geometric scheme the number of tasks sharing the lowest priority is much more compared to the arithmetic scheme. So, if r tasks share the first scheme, let us say 2 share here and if there are 8 task 8 priority levels, then 2^8 tasks will share the last level. So, the geometric scheme might work better than the arithmetic and the uniform scheme if the number of tasks is too many.

If we sum this, if the number of tasks is let us say 128 and the number of priority levels is 4, then this becomes $r(1+r^2+r^3)$. So, it is basically $r[(r^3 - 1) / (r-1)]$ and so on which is equal to 128. And we know ok now we can solve this and determine the value of r .

And then we will assign the tasks according to r, r^2, r^3, r^4 and when the number of tasks are too many, maybe the geometric scheme worked better than the arithmetic scheme and definitely better than the uniform scheme. But then we might do a small simulation to check which will produce better results.

(Refer Slide Time: 25:18)



Logarithmic Scheme

- Experiments show that the logarithmic grid works the best.
- Basic idea:
 - Shorter period tasks should be allotted distinct priority levels as much as possible.
- Assume: $r = (\underline{p_{\max}} / \underline{p_{\min}})^{1/n}$
- The grids are formed at $r, r^2, r^3, r^4, \dots, r^n$

The slide features a video inset of a man in a striped shirt on the right side. The background has a blue and green geometric design. At the bottom, there are two logos: one of a tree and another of a gear.


The last scheme that we discuss here is the logarithmic scale and the experiment so that when the number of tasks are large they are not marginally more but large like number of priorities 8 and the number of task are 40, or something like that. And also if the task periods vary widely, then the logarithmic grid scheme works best. Let us look at the logarithmic grid scheme.

Here again, the guiding principle is that the shorter period tasks or the higher priority tasks are allotted distinct priority levels as much as possible. And here the value of r is computed as p_{\max} / p_{\min} . So, all the tasks we have their periods and they find the maximum period in the minimum period and then n is the number of priority levels. So, p_{\max} / p_{\min} . And that gives us the value of r . Now, the grids that are formed are r, r^2, r^3, r^4, r^n .

(Refer Slide Time: 26:42)

Logarithmic Scheme: Example

- $r = (P_{max}/P_{min})^{1/n_{prio}}$
- $P_{min}=1\text{msec}$, $P_{max}=100,000$ and $n_{prio}=32$
- $r=1.43$ $(10^5)^{1/32}$
- The gridlines become 1.00, 1.43, 2.04, ...



Let us just take an example. We have let us say an application where the minimum period is one millisecond and the maximum period is 100,000 milliseconds and the number of priorities are 32. So, here $p_{max} / p_{min} = 10^5$. And the number of priority levels is 32. So, $1/32$. And if we compute this, then r comes out to be 1.43.

Now, we can form our grid lines that are between 1 that is the minimum period of the tasks 1 and 1.43. We have the priority 1, priority 2 between 1.43 and 2.04, and so on. So, if the periods are varying widely, the logarithmic scheme would be a good technique to form the grid lines for the priority assignment.

We looked at the different techniques for priority assignment. When the number of tasks exceeds the number of priority levels and many applications, it can be the case that the numbers of tasks are higher than the number of priority levels available. We are almost at the end of this lecture. We will stop here. And we will continue in the next lecture where we have one or two small issues with the rate monotonic scheduler, and we will start discussing those issues first. Thank you.