Real -Time Systems

Prof. Dr. Rajib Mall

Department of Computer Science and Engineering

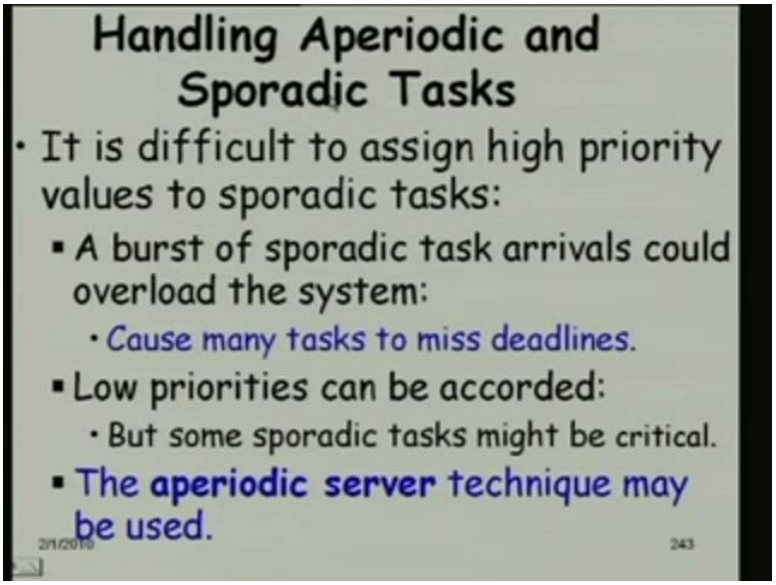
Indian Institute of Technology, Kharagpur

Lecture No. # 12

Few Issues in Use of RMA

So, let us get started, and we will now examine few other issues that we will have to handle while using RMA. So, far we had looked at the schedulability criterion, and we had looked at how to handle self-suspension, effective due to contact switching and so on.

(Refer Slide Time: 01:01)

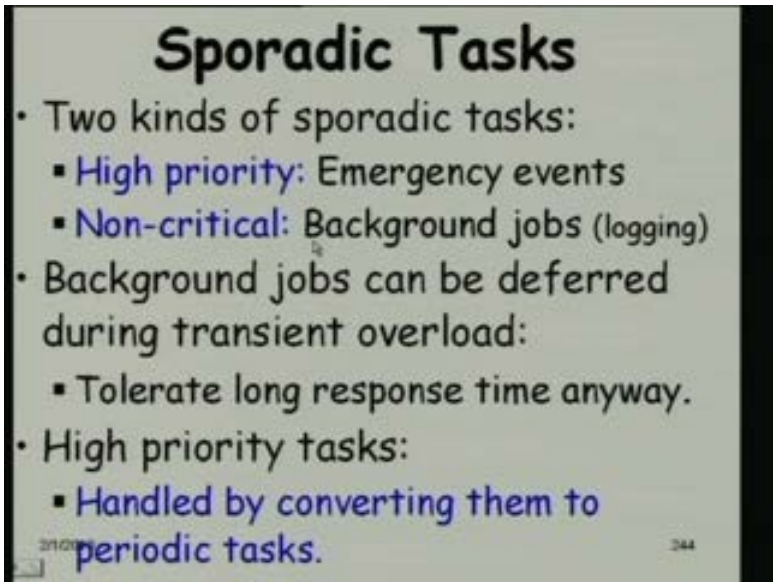


One thing that in a practical situation, we have to handle is that there may be few aperiodic or sporadic tasks; in every non trivial situation, we will have them. Several examples of such task, we have to seen earlier; for example, handling some exceptional events like fire condition zone or a robot finding an obstacle, these are examples of such tasks. Now, one thing is that we cannot really assign a very high priority to these tasks, and also another thing is that, how do we convert this aperiodic and sporadic tasks into periodic tasks, because the rate monotonic analysis, it assigns the priorities based on deadlines.

And if we just assign high priorities to sporadic tasks, our schedulability results become inapplicable; not only that, by assigning high priority to sporadic tasks, when a burst of sporadic tasks arrives, many tasks would miss their deadlines. And many situations, they have low priorities can also not be accorded, we cannot just say that all sporadic tasks are low priorities, because then the sporadic tasks might miss their deadlines; and the sporadic tasks can also be critical for example, fire condition handling.

So, this is a tricky situation actually, we can neither make the sporadic tasks, very high priority; just arbitrarily assign a priority one or two to sporadic tasks, because that will not only make our schedulability results inapplicable, but also whenever there is a burst of sporadic tasks, some tasks will miss their deadlines. We can neither assign them low priorities arbitrarily, because sporadic tasks will miss their deadlines, and they might be critical. So, the one, which is supported or which is being used in most of these real applications are the aperiodic servers.

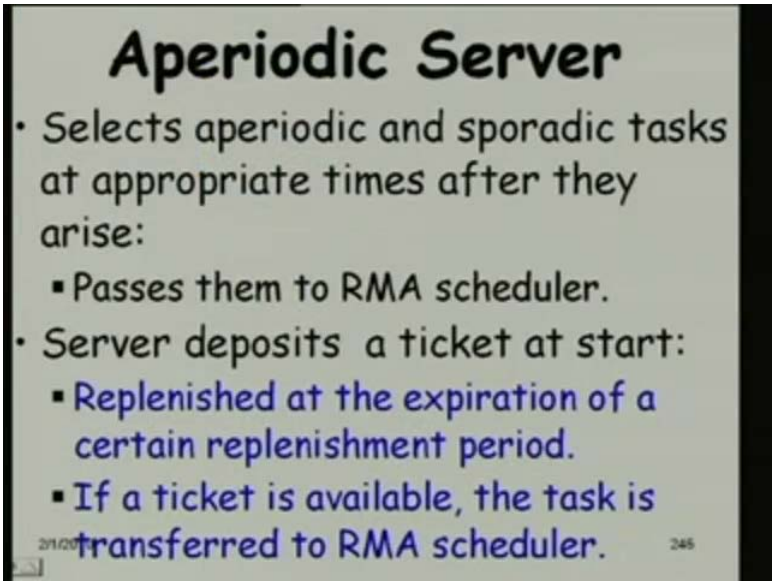
(Refer Slide Time: 03:22)



Now, let us investigate little bit about the sporadic tasks themselves. Actually, there are two types of sporadic tasks that a system can have; one is a very high priority task called as which results from handling certain emergency events like fire condition. There can be non-critical sporadic tasks for example, logging activities; the logging activity can arise as soon as some task completes right, so it can arise arbitrarily. So, both of these are sporadic tasks.

Now, the activity such as logging, this can be deferred, this can be delayed when there is a transient overload, but the high priority tasks, which arise due to emergency events, they have to be completed. The background jobs like logging they have long response time if the logging text after a minute or 2 minutes does not matter, but the high priority, the emergency events must be handled within few milliseconds or microseconds as the situation may be. So, the weight is handled is that the high priority tasks are converted into periodic tasks by the aperiodic sever technique; whereas, the background jobs, the non-critical tasks, they are assigned low priorities.

(Refer Slide Time: 05:00)



Now, let us look at this aperiodic server technique. So, the role of the aperiodic server, this is a component basically, harden component to a scheduler. The role of this is that it is reported about all aperiodic and sporadic tasks that arise, and it sends them to the RMA scheduler at appropriate types. So, just observe that the aperiodic server is a harden component to a RMA scheduler. Every aperiodic and sporadic tasks are send to the aperiodic server, who might keep some of them waiting, and as and when it feels necessary, it will pass them into RMA scheduler.

And we will see that that makes them into the high priority tasks, we will become periodic tasks. So, the way it will work is that the aperiodic server to start with we will deposit one ticket, and this ticket if it is consumed, we will be replenished after certain replenishment time. So, these are designer decision the replenishment time when a ticket

is replenished. So, this periodic replenishment of the ticket actually, make it makes it a periodic task right, because at best, if this period only one task will be sent, if a ticket is available, the task is transferred to the RMA scheduler; if no ticket is there, then the aperiodic task will keep on waiting, until the ticket becomes available.

So, sometimes the ticket may not be used right, because no aperiodic task occurred at that time. So, if a ticket is already there, it will not be replenished; otherwise now there will be too many tasks coming into the RMA scheduler. The replenishment will occur only the ticket has been consumed; if are some duration, the ticket was not consumed, will not be replenished. So, still at best, there will be one ticket anytime.

Ticket will take a great task.

No ticket is a pass; if a pass is there, the task will be allowed to go in.

So, RMA scheduler will see it as a task or or ticket.

No, RMA scheduler will not see the ticket; RMA scheduler will only see the task.

So, that is scheduler, we have not to make the schedule of RMA scheduler beforehand. So, in that schedule, we…

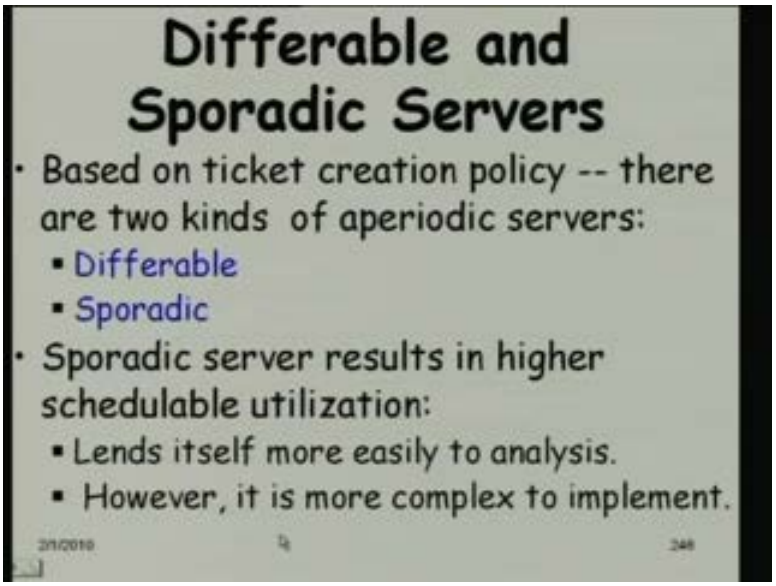
No, RMA, the schedule is not constructed beforehand, the priority is assigned beforehand. So here, the priority of the task will be the same as the replenishment time; that will be the rate. At what rate replenish…

So, (( )) assign a prior to to the ticket beforehand.

Not ticket, to the task. The tasks which are being handled by the aperiodic server, they will be assigned priorities based on what is the replenishment time of the ticket. And if you have different criticalities or different characteristic of tasks, we can use multiple aperiodic servers even for the same system; so simple case we are considering, just one single aperiodic server, where a set of tasks are coming in, and this can come at arbitrary times, but we are in effect converting them into periodic tasks, by using the concept of a ticket and replenishment of ticket after certain period. If the ticket is available, then only the task will be transferred to the RMA server. No ticket, the ticket has just been

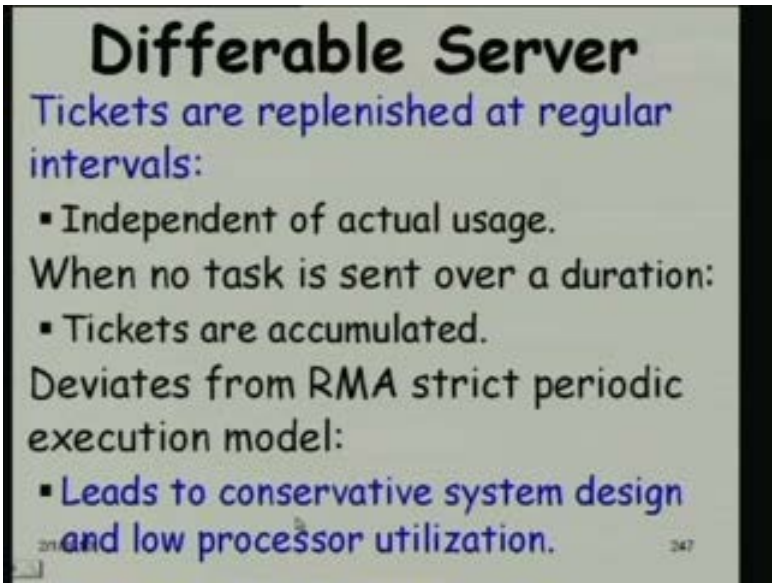
consumed, and task has is waiting, it will keep on waiting until the ticket becomes available.

(Refer Slide Time: 09:08)



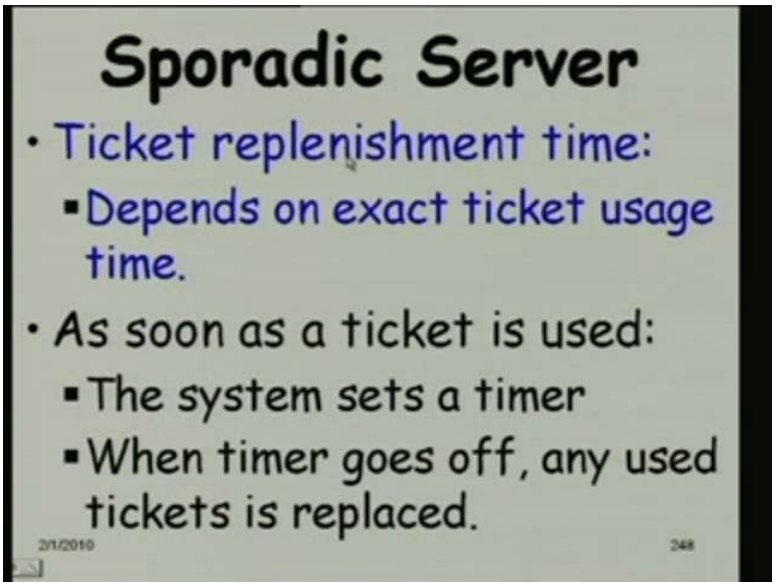
Now based on the ticket creation policy, there are two kinds of aperiodic server; one is called as the deferrable aperiodic server, and other is called as the sporadic aperiodic server. The sporadic server actually, results in higher utilization of the processor, and lends itself more easily to analysis, but it is more complex to implement. So, the one that is used widely is the differable server; even though the sporadic server has many good characteristics, like it results in higher schedulable utilization; more easily can be analyzed the schedulability analysis can be performed, but the problem is that it is more complex to implement.

(Refer Slide Time: 10:00)



So, first let us look at the differable server, which is used widely. So, here the tickets are replenished at regular intervals, independent of the actual usage, so just keep on depositing tickets; even if there is a ticket, do not worry; if it is not been consumed, just give another ticket. And if no task arises over a duration, the tickets are naturally accumulated. Now, the thing is that it deviates from the periodic execution model, because it might so happen, the two tasks might arise in quick succession. So, the analysis becomes difficult, but the implementation is easy, you just keep on assigning tickets; and if you are using a differable server, and you will have to make the system design very conservative, to choose a very low utilization of the processor.

(Refer Slide Time: 11:12)



In a sporadic server, the ticket replenishment time actually depends on the exact ticket usage time. As soon as a ticket is used, the system sets a timer; when the timer goes off, the ticket is replaced, but you might say that why not? Periodically, check if the ticket is there or not, and then replenished based on that. What what do we think? Will that help? See this we are saying that we have this slightly complicated to implement a sporadic server, because we have to check, when the ticket just got used; and as soon as it get used, we have to set a timer, and when the timer goes off, the ticket will be replaced.

But what if, we just kept on periodically generating tickets, and then if we find that a ticket exists, do not do nothing; and if the ticket does not exist, then we just deposit a ticket; will that help?

Why are you keeping a timer as well (( ))…

That is a point to think actually. So, he is asking the question that why we keep a timer. Cannot you just periodically exact? Set one periodic timer and keep on as soon as the period periodic timer expires; we just check the ticket status, and keep depositing ticket. Why do we have to set a asynchronous timer here? Each time a ticket is used and just check it, and…

But at a time more than one task come, they (( )) it will be avoided (( ))

Suppose one task does not come for the long time

Yes.

Then (( )) checking it over and over again.

That's not a problem; that is not a problem, because a periodic timer keeps on checking, and if it finds that the ticket is not used, it does not deposit a ticket; if it is used, deposits a ticket. But we are saying that, that will make it that will have some issues with it; we should not use that I mean the sporadic server is much more suitable. So, what is the advantages sporadic server to a situation, where the ticket is checked periodically, and it is replenished, if it is used.

May be between the period, if a task arises, it may have to wait until the ticket is…

Not a problem, because anyway it have to wait you know, see let us see, see here the main problem is that let us assume the situation, where a ticket was unused for let us say, most of the time, and now the ticket has just got used; and by that time, just immediately another ticket has got deposited, and another task has come and use that. So, there can be a burst of two tasks; is it not?

So, that will make the RMA results, periodic model inapplicable. So, whatever schedulability analysis etcetera will not be applicable here, because here the tasks can arrive in quick succession; do you see the point? The ticket was unused, and then as soon as the task came and it was passed on to the scheduler, just after that we have checked periodically now by that time, and then deposited a ticket; and by that time, another task has come, and we just had a pass with the ticket, and quick tasks successively can get in; do you see the point? So, we cannot really do that.

So, a sporadic server, it needs to set a timer each time the ticket is used, these enforces that two successive tasks cannot arise immediately, at least this much the timer duration a task has to wait. So, this has converted it to into periodic task; if some periods tasks will not arise, and that will cause no problem, but what we are preventing is that two tasks will not arise successively. So, here if we use a sporadic server, the utilization of the processors can be made high, and for very critical tasks and so on. We can use the

sporadic server technique. But the differable server has the bottleneck has the problem that multiple tasks might be transmitted to the RMA scheduler at the same time.

What is the… If there are multiple task, we can we can then execute them one by one, what is the problem?

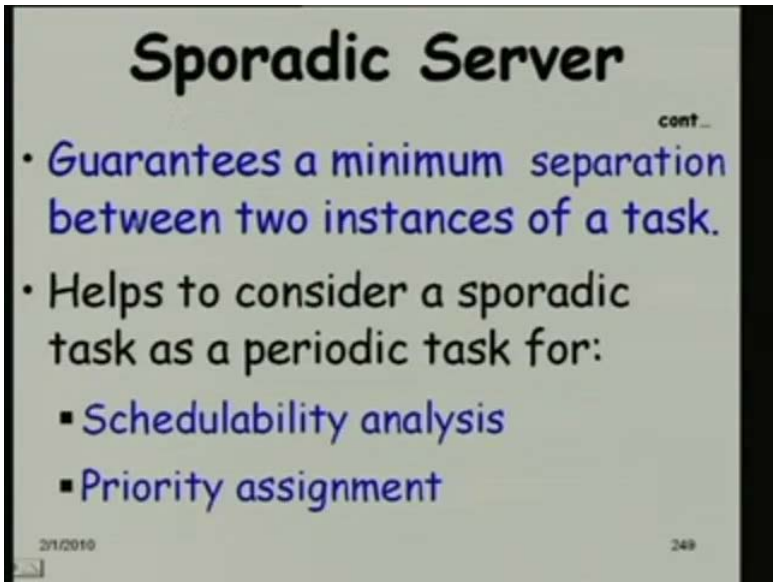
The problem is that see, his question is that what if there are multiple tasks which are transmitted at the same time to the server. See the problem is that we have done this schedulability analysis, assuming a periodic task model right. So, as soon as we have at certain period, multiple occurrences occurring within fast very quick succession, the higher priority task will not be affected, but the lower priority tasks will miss their deadline. So, that is the problem actually.

Even if we get one a periodic task, the same problem will be there; if we get two, also this problem will be there. No, if there is one, the problem will not be there, because we have made our analysis, the schedulability analysis having a periodic model in time. So, as long as there is one task occurring at most in one period, they will run, they will run.

Aperiodic task has (( ))

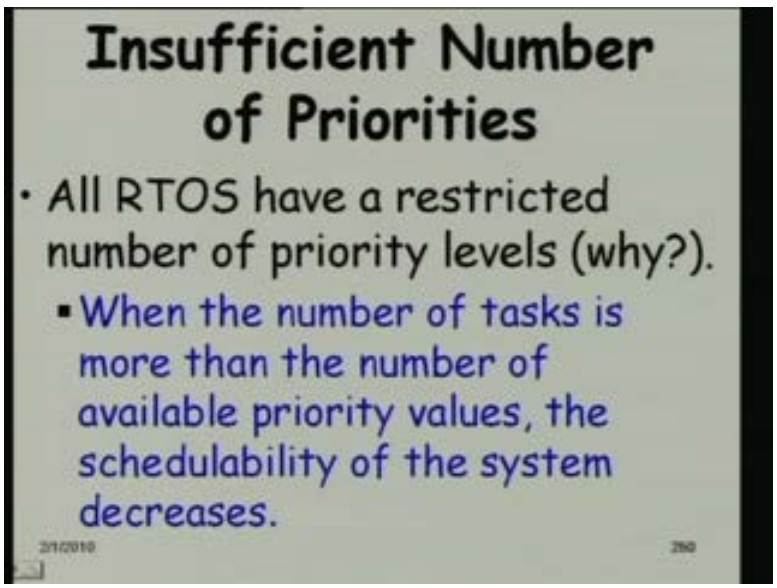
Aperiodic has been converted to its periodic model. So, in some periods, the aperiodic task is considered; some periods, aperiodic task do not exist. So, that slot is unutilized or not slot actually. So, there we do not have to consider the aperiodic task; is that ok? So, let us proceed.

(Refer Slide Time: 16:08)



So, the sporadic server has many good properties; guarantees a minimum separation between tasks. And helps consider a sporadic task is a periodic task or schedulability analysis and priority assignment. But the only problem with a sporadic server is the complexity in the implementation of setting a periodic timer each time. So, unless its required the sporadic server is not used; the differable server is more efficient.

(Refer Slide Time: 18:15)



Now, let us see another situation; specially, relevant to small embedded systems. This concerns the insufficient number of priorities; see all the real time operating systems as

we will discuss, they have very limited number of priorities, priority levels 8, 16 may be 4. So, why do they restrict the number of priority levels?

So, if the number of priority level increases, the context which is (( )) even a single deferential of priority with (( )).

That is not a problem; that is the problem of the designer. So, why the operating systems do not support many priority levels?

(( )) the implementation would be more difficult in (( )) more priority level.

What is the difficulty in the implementation?

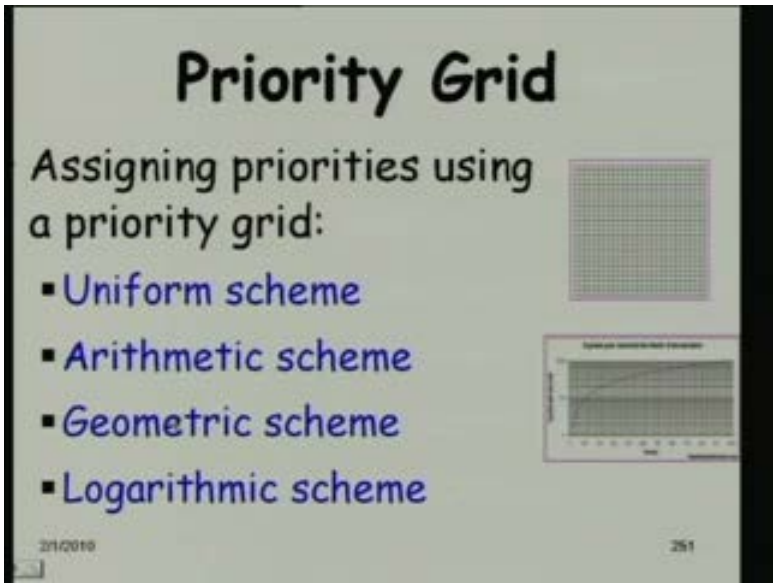
If we have a multiple feedback, you will have multiple…

Yes .

Exactly, exactly. So, that is the problem. The problem is that when the number of tasks… So, as he says, so I think this is not the reason, why I offer this? the implementation of the system becomes difficult, because as we said that we have multi level feedback used, that is why the systems restrict, the operating systems restricts the priority level right. The implementation will be complex, but the consequence, here we have the consequence here; the consequence is that it might so happen that if our operating system supports and let us say 4 or 8 priority values, and we have many more tasks, then multiple tasks have to be assigned the same priority level; is it not?

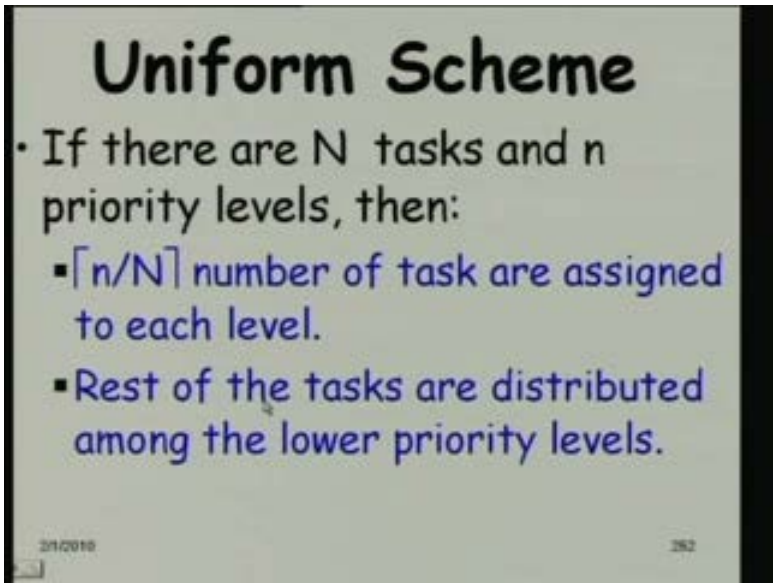
We have 4 priority values that are available, and we have let say 10 tasks, then some tasks have to share the same priority value; and that reduces the schedulability, because we had so far assumed that every task is given a unique priority based on its rate. So, even if the rate is different, we are assigning then the same priority, and that will reduce the schedulability and the utilization of the processor; we have to do a conservative design of the system. But even while assuming that multiple tasks have to be given in the same priority level; can we just arbitrarily give them different priority values? You know, club at priority one some tasks, club in priority three some tasks and so on; will that be meaningful or is there any systematic way, we can do that. So, lot of investigations have occurred in that let us look at the results.

(Refer Slide Time: 21:30)



One possibility is that we can use a uniform grid; and if you have four priority levels, so we can club those which are in a specific grid into those values right. The other is a logarithmic grid; we have multiple of them actually; the uniform scheme, arithmetic scheme, the geometric scheme and the logarithmic scheme. Now let see these four schemes.

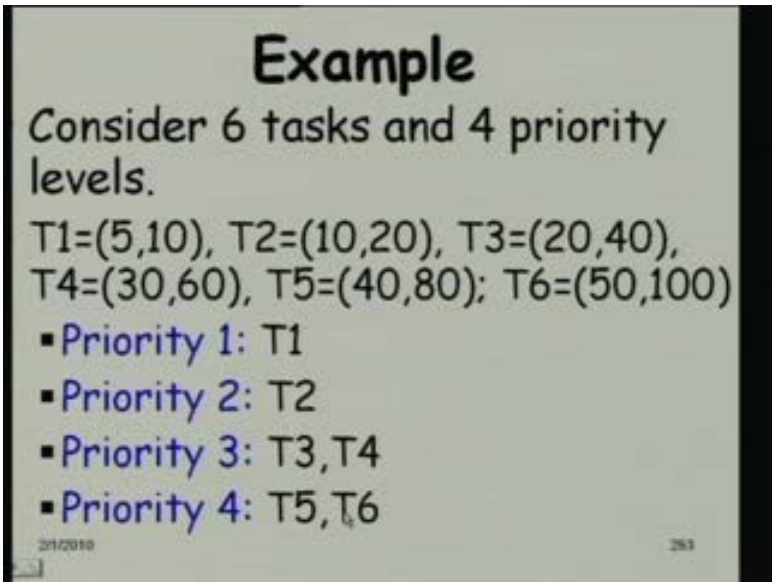
(Refer Slide Time: 22:02)



In the uniform scheme, if there are n tasks, and n is the number of priority levels; n is definitely larger than n that is the situation we are considering. Then we should have

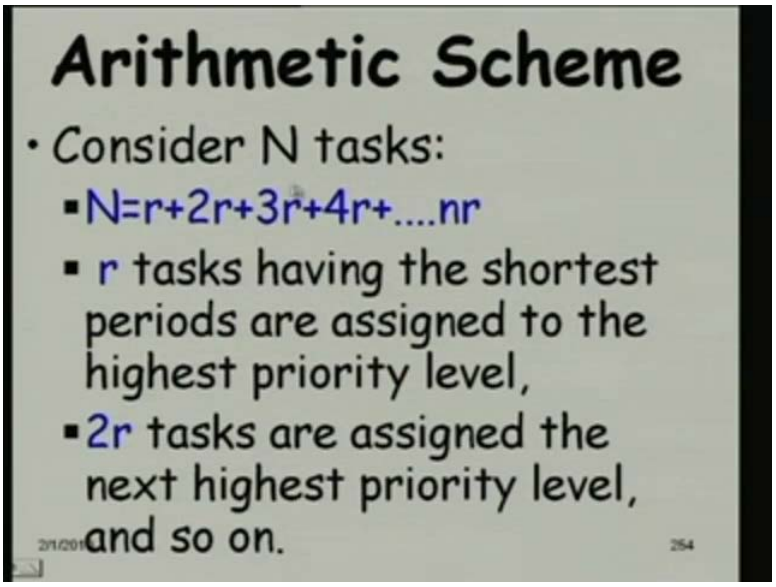
capital N by n, so there is mistake here. So, capital N by n number of tasks are to be assigned to each level right uniformly, we are assigning. So, we are trying to make the number of tasks similar at each priority level. So, we have 8 tasks and 4 priority levels, we are assigning 2 tasks per priority level. So, let us consider 6 tasks and 4 priority levels. So, here the first… So, for the, see the number I think this is all right actually; is it? Not really actually. So, the first few priority values, we should assign the highest priority values, we should assign 1; and the rest we can assign 2.

(Refer Slide Time: 23:30)



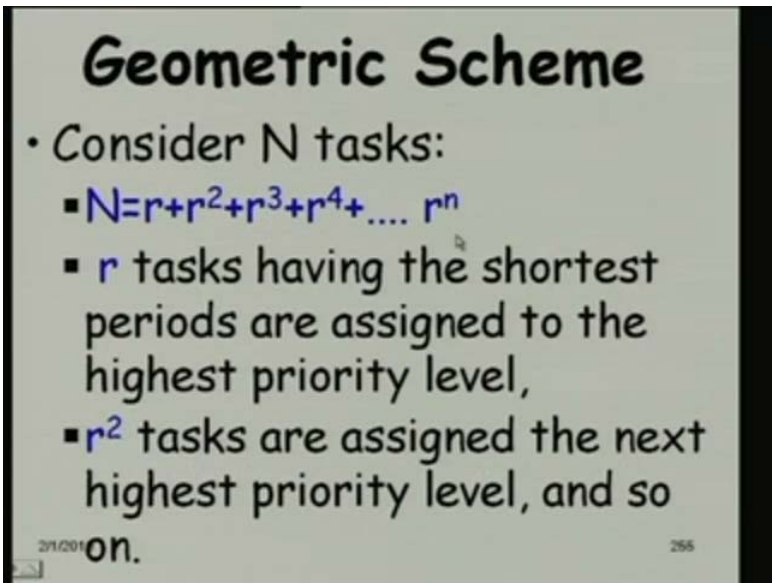
So, this is an example here. That we have 6 priority, 6 tasks and 4 priority levels. So, 2 tasks need to be distributed; is it not? So, two tasks will have one sorry two priorities will have one task each, and another two priorities, we will have two tasks each. So, the higher priority, we should assign one task; and the lower priority is two tasks. The idea is that the higher priority task should not miss their deadline, we should not assign higher priorities two task each, and then the lower priorities one task each, that will make it meaningless, because many task can miss their deadline. Here we are ensuring that these tasks will not miss their deadline.

(Refer Slide Time: 24:35)



Now, let us see the arithmetic scheme; in the arithmetic scheme, if there are n tasks and we express n is r plus 2 r plus 3 r plus 4 r up to n r, then the first priority, the highest priority, there will be r tasks; in the second highest there will be 2 r tasks; third highest 3 r tasks and the lowest priority will have n r tasks.

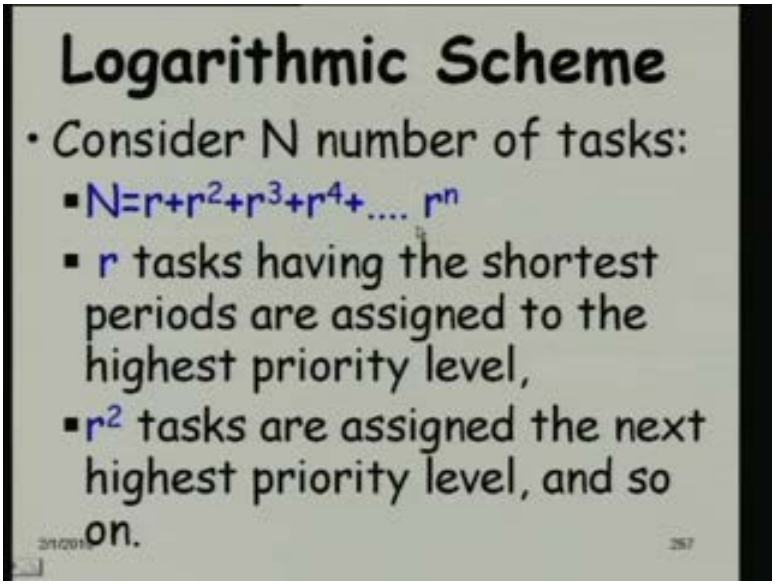
(Refer Slide Time: 25:08)



And it has been found that the arithmetic is more schedulable than the uniform scheme; it works better, the arithmetic works better than the uniforms scheme. And the geometric scheme, the first priority as r, second as r square, third as r cube, r 4. So, just see here

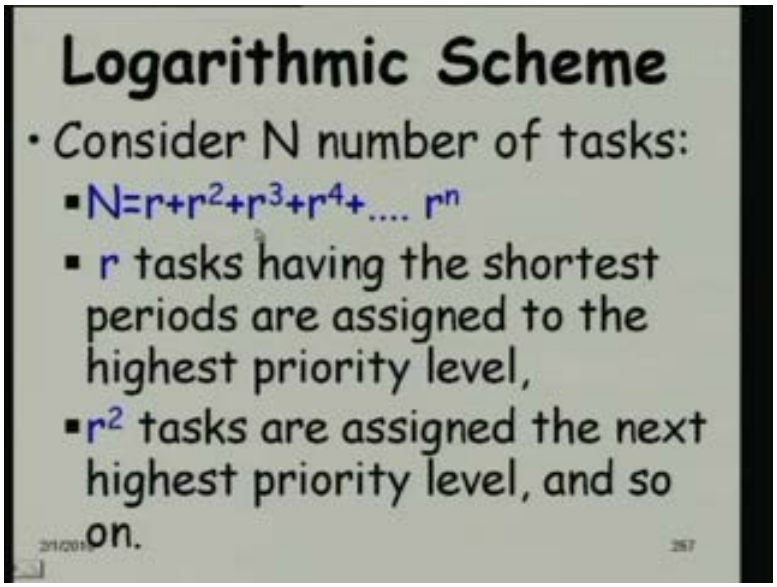
that the lower priority tasks are given more task, the lower priority values are more tasks, highest priorities have less tasks both the… See there in the uniform scheme see here, both 3 and 4 have similar number of tasks, but in a arithmetic scheme the priority 4 will have 3 tasks, and this will have possibly one task right. So, the idea is that the lower priority tasks, lower priority value should have more number of tasks.

(Refer Slide Time: 26:07)



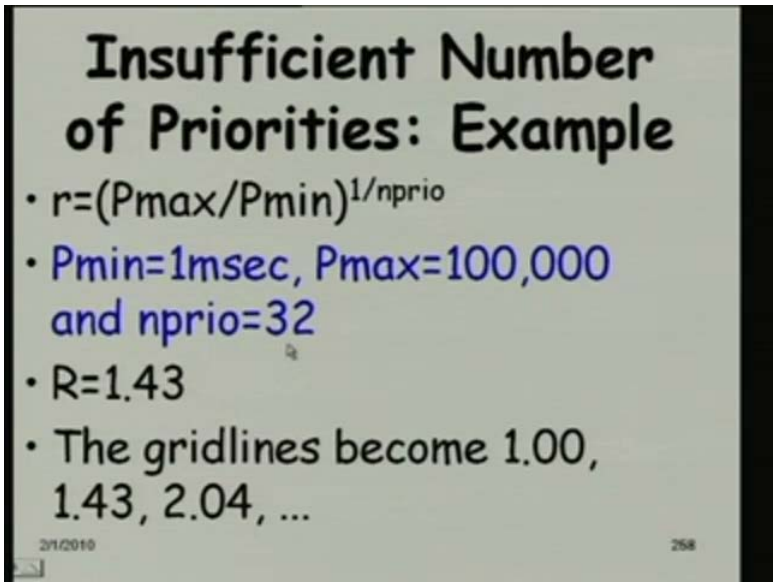
So, both the arithmetic and geometric scheme and we can also have a logarithmic scheme and the experimental result, so that this is the one, which is works best. The idea is that the shorter period tasks should be allotted distinct priority levels as much as possible; and the longer period tasks, they can say priority values. So, let us assume that the maximum period is p max, and the lowest period is p min, and there are n number of tasks. So, we just find p max by p min to the power 1 by n is let us say r.

(Refer Slide Time: 26:46)



Then, once the r has been found, we assign them in r r square, r cube up to r n. So, this is an example.

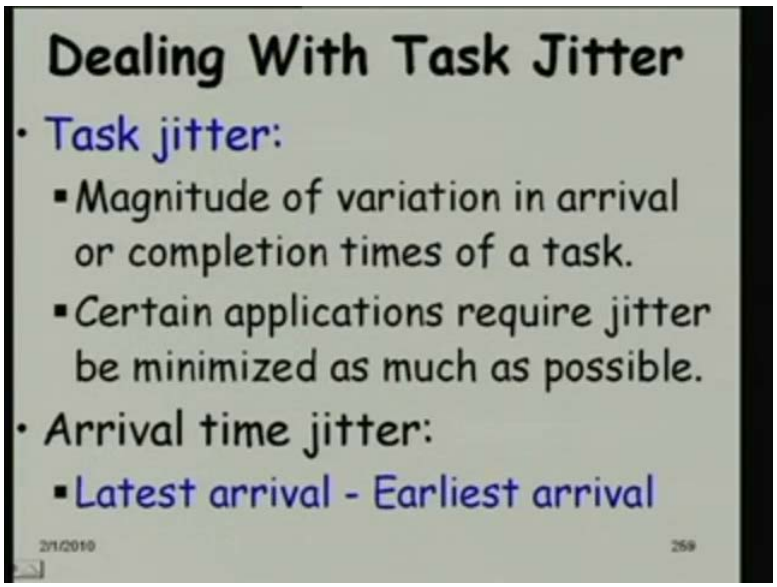
(Refer Slide Time: 27:03)



So, let us say p min is 1 millisecond, and p max is 100,000 and the number of priority is 32; then R works out to be 1.43 and the gridlines become 1, 1.3, 2.04; and then based on the tasks, which fall in the gridline, they are assigned with that priority level. But again, there are issues here, what if some gridlines are not occupied. So, some priority values are not used, is it? See here, we have assigned the priority grid says 1, 1.43, 2.04, 3.5 and

so on. So, now, we are distributing the tasks based on their periods, and then clubbing them into the corresponding priority value. Now, what if some priority range does not have a task of course, we can consider some tasks from the higher priority values.

(Refer Slide Time: 28:09)



Another issue here is dealing with task jitter, what is exactly a task jitter? Anybody would like to answer; what is a task jitter?

Delay in task arrival time.

He says delay in task arrival time. Anybody would like to answer any different way, what is a task jitter?

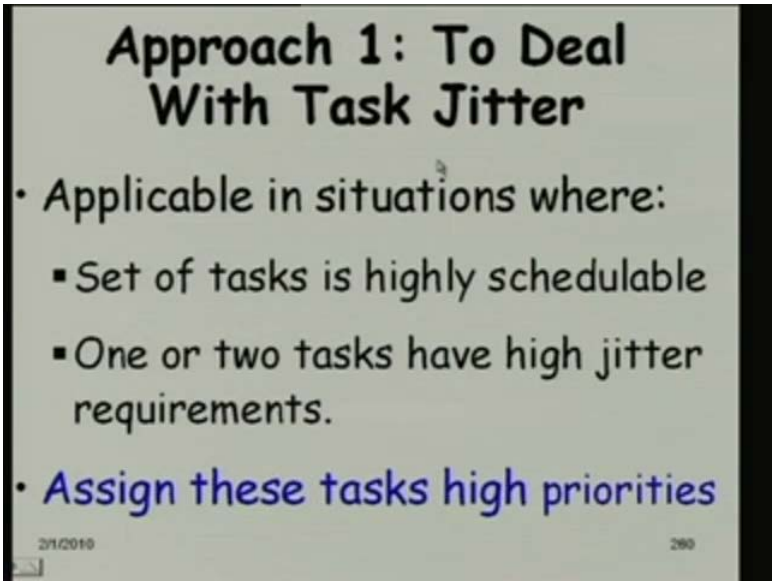
The difference in the delay that will go successive arrival (( ))

That is what he says; that is what he says that the it deviates from the periodic model, is it not? That is what he says; deviates from the periodic model, in rather than coming every 10 millisecond, some the task one instance, somehow it has come in at 9 millisecond, another at 11 millisecond, another at 10, let us say 20 millisecond, next one at 32 millisecond.

10 on the other end also (( )).

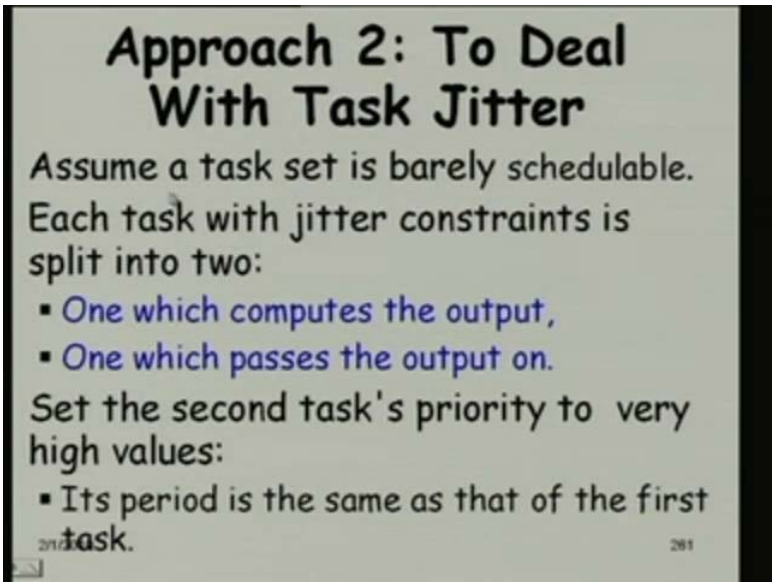
So, the task jitter, it is the magnitude of variation in the arrival or completion times of a task. So, one is that the variation in the arrival time caused the completion time jitter or may be the task took more time to execute and there was a jitter. But some applications, they require the jitter to be minimized as far as possible specially, the multimedia kind of applications, where one of the quality of service parameter is the task jitter, because if let us say frames are being transmitted on a IPTV or let us say IP phone, and the delay there is a wide variation in the arrival rate of the frames. Then you will find the quality has gone down. In the video, you will find glitches; in the audio also you will find it very funny to hear. So, in many applications, the jitter needs to be minimized, and the arrival time jitter is the latest arrival that is possible minus the earliest arrival. So, if the latest is 2 millisecond, and the earliest is 2 millisecond beforehand, so 2 millisecond after it can arrive, and even two millisecond before it can arrive, then the jitter is 4 millisecond.

(Refer Slide Time: 30:46)



So, how do you do deal with jitter, because the scheduling algorithm, so try to minimize the jitter. One is one solution is when we have a small number of tasks or we have a set of tasks that are the utilization of the processor is low, highly schedulable set of task. So, the tasks, which are very high jitter requirement, we can assign them high priorities; so that even they are commonly till late, arrival time jitter still they will complete, before the required time.

(Refer Slide Time: 31:38)



The other case is that when we have tasks set, that is barely schedulable; we have try to maximize the utilization of the processor. Now, we have a situation where some tasks there need to have minimum jitter, so what do we do? One is we can split the task into two, the one that has jitter; and the the two parts are one, which takes the which computes the output, and produces the result; and the other, which gives the result to be used right. One is it test the input and computes; the other is it actually produces the result. So, the second task's priority can be made into higher value, and the first task can be lower value right. So, this will this can also take care of the jitter.

But what if the arrival time, because of (( )), because of arrival time jitter also then deadline (( )).

Exactly see the let us just answer his question. So, what he is saying is that the completion time jitter is caused due to arrival time jitter right. Now, when there is a arrival, when there is a… When the completion time must be within the deadline, so the arrival time, it has come in late, how do we make it complete, and then produce the result? So, what we are saying here in this, when the schedulability is low, what we are saying is that see the task can be completed, the main processing can be completed and then when required, but finally, the result that is actually passed in, passed out should be completed the earliest.

So, it is not that this task actually runs for this entire duration, and then the result is passed; if they transfer the entire duration, and the result is passed, this will make no difference. But in an typical situation, what happens is there are many tasks right, so sometime are there, the result is produced; but when result is to be given out, that has a strict deadline, it must give out by that deadline.

Sir, this can be applicable, if the completion time jitter is because of some processing (( )).

No, it can be due to arrival time jitter also.

(( )) then it will (( )), but if it is due to arrival time (( )).

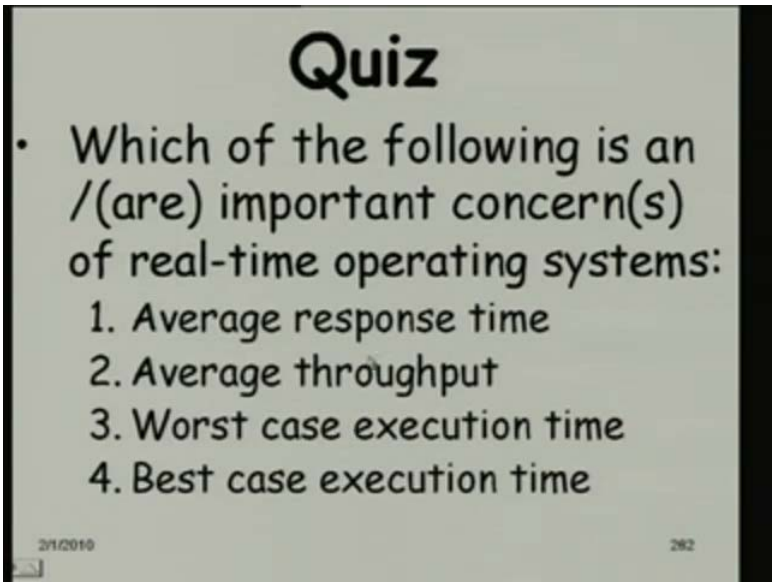
Yes, see what we are saying is that even if there is arrival time jitter, if you did not do this right, you just let the entire task run at as a low priority, then it will definitely miss its deadline. But here what we are saying is let it run and try to complete, and the last part of the processing, and passing the output will be done at a high priority. So, many of the jitter will be minimized here, compare to the previous case; it will have much less jitter.

Still it may.

Yes, still it can obvious.

Now, let us just we have discussed about the basics of the very simple uniprocessor scheduling algorithms, EDF and RMA and RMA variations. So, let us just have few quiz questions, let us see if you are able to answer them.

(Refer Slide Time: 35:25)



So, for a real time operating system, which of the following are important concerns. Is it average response time, average throughput, worst case execution time, best case execution time.

(( ))

So, which which ones, you think are important for a real time operating system.

(( )) worst case (( )).

Worst case execution time is definitely a requirement, because…

(( ))

Even under the worst situation, the deadline should not be missed; but what about the other things?

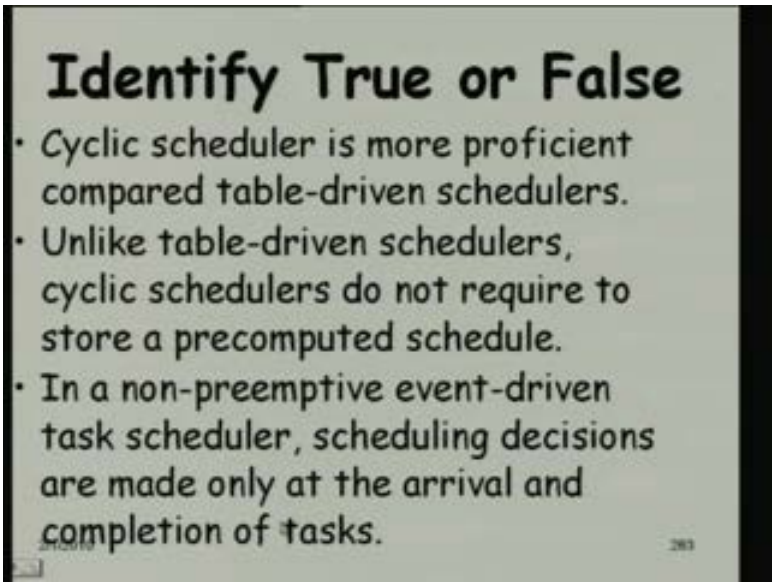
(( ))

Throughput and response time, is it?

(( ))

Not really. Actually, here in a real time operating system, the tasks have deadline, and as long as they complete within the deadline, it is fine; we do not want to, there is no benefit by completing tasks much early. So, the only correct answer here is that the real time operating systems, the concern of most real time of all real time operating systems is to see that the tasks in their worst case execution time also execute within their deadline.

(Refer Slide Time: 37:08)



Now, let us see some true false kind of questions. So, what about this question? So, suppose I give you a statement saying that the cyclic scheduler is more proficient compared to table driven scheduler; you know about a table driven scheduler; right? But the schedules are stored varies, and as soon as the task completes, the scheduler wakes up and tries to execute the next task. Whereas in a cyclic scheduler, it is based on the minor frames, the scheduler wakes up. So, what do you think?

(( ))

So, please think over; true, some of you are telling true. Anybody, telling false? Noone is telling false, is it?

Proficient (( ))

Proficient see we had defined earlier proficient, this term a proficient a scheduler is more proficient than another scheduler, if it can schedule some set of task, the other scheduler cannot it is more proficient, but whatever the other scheduler can schedule successfully, it will also be able to schedule, let that is a more proficient scheduler. One scheduler is more proficient than another, if it can schedule all set of task that have scheduled by the other, but there can be some tasks which more proficient scheduler schedules, but the other less proficient scheduler cannot do that. So, what do what do we think about this assertion? Cyclic scheduler is more proficient compared to table driven schedulers. All are saying true, is it?

Sir, as proficient (( )).

As proficient means, if one task, scheduler can run one set of tasks, and the other will also be able to run.

Sir, these are as proficient not more (( )).

So, anybody would like to answer? He says that both are equally proficient. Anybody, saying false? That table driven is more proficient than cyclic scheduler, no one think so as it. Actually the table driven scheduler is more proficient than the cyclic scheduler, this is a false statement. The reason is that in a cyclic scheduler, a part of the minor cycle or the frame is wasted right it it schedules only at the frame boundaries. So, parts of it, the CPU remains unutilized; whereas in a table driven scheduler, the utilization is hundred percent. So, if the table driven scheduler can construct some schedules, the cyclic scheduler may not be able to have a schedule for that. But the reason why cyclic schedulers are used overwhelmingly is that…

(( ))

It is much more efficient; the implementation is very efficient; you just need a periodic timer just keeps and giving those frame boundaries; whereas, in a acyclic, you have to set up a periodic timer, each time for a table driven scheduler. And setting a timer text time, so if you have just 2 or 3 instruction task, and out of that, it is a two instruction to set the timer, then becomes inefficient; anyway let us proceed.

Now, whatever the second statement? Unlike the table driven schedulers cyclic schedulers do not require to store any preempted precomputed schedule. So, it says that only the table driven schedulers need to store a precomputed schedule whereas, the cyclic schedulers do not require any precomputed.

(( ))

So…

(( ))

No, do we have to compute a precomputed schedule? Yes or no.

(( ))

Yes. So, for the major cycle, we need to compute the schedule. So, which frame, which task we will run, also needs to be stored for a cyclic scheduler. So, these are offline scheduler, both table driven and the cyclic scheduler are offline schedulers; the schedule need to be computed by the programmer needs to be stored.

Let us look at the third question. In a non-preemptive event driven task scheduler, scheduling decisions are made only at the arrival and completion time task. See that scheduler characteristics are it is a event driven tasks scheduler, but it is non-preemptive.

Only at the completion not at the arrival.

Exactly. See, it is a non-preemptive scheduler. So, on arrival it cannot run it cannot preempt and run right. So, let us look at the next question; for scheduling a set of soft real time tasks on uniprocessor, RMA is a better scheduling algorithm compared to a time sliced round-robin algorithm.

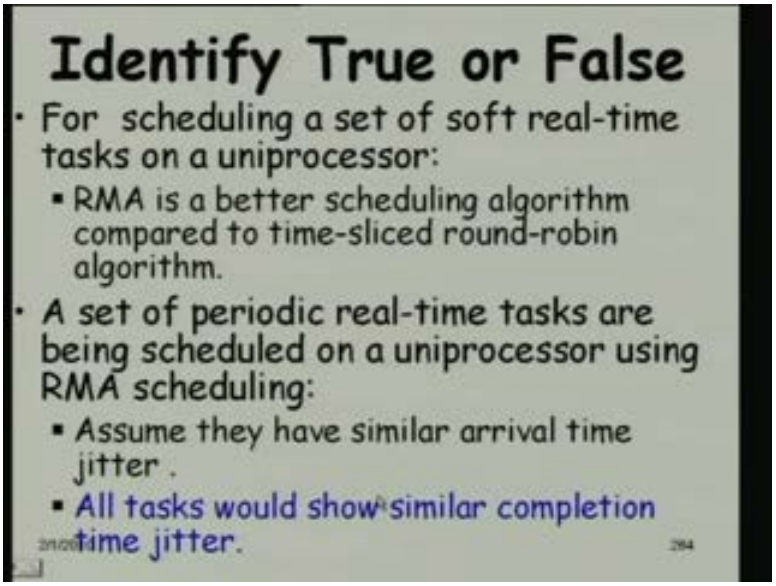
Soft (( ))

Just a clarification on the previous question, I think see here, this question here, in a non-preemptive event-driven task scheduler, the scheduling decisions are made only at arrival and completion of tasks.

Only (( ))

Only at completion. So, even if a task arrives, you will keep it waiting, until nothing completes, is it? So, if nothing is completing let us say no tasks are running, so that means, you will not even invoke the scheduler is it? So, here also both the arrival and completion times will be considered, because as soon as the task arrives, it will not preempt, but if the CPU is ideally, it should be taken off our scheduling. So, the scheduling points are both the arrival and completion of tasks for this also. Is that appear?

(Refer Slide Time: 43:56)



Now, let us look at this statement. For scheduling a set of soft real time tasks on a uniprocessor, RMA is a better scheduling algorithm compared to time sliced round-robin algorithm. How do you think of this statement? Is it true or false?

False (( ))

Why is that? So, all of you are saying false.

Soft real time (( )).

Soft real time means what…

The responsibility (( ))

(( ))

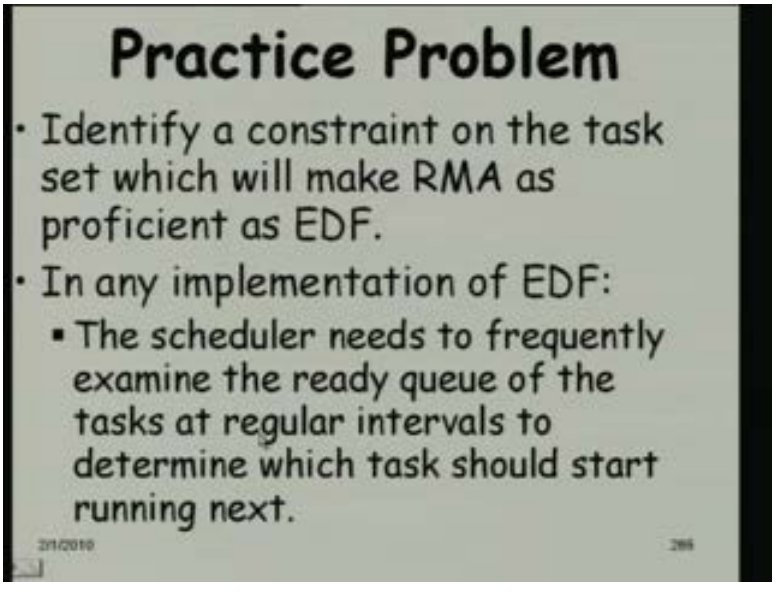
Exactly, exactly; the number of tasks that complete per unit time is important here. So, throughput etcetera, these become important parameters; and the round-robin, time sliced round-robin is a better algorithm for soft real time tasks.

Now, assume a situation, where a set of periodic real time tasks are being scheduled on a uniprocessor using RMA scheduling. So, we are using RMA scheduling, and we have a set of periodic real time tasks on a uniprocessor. And assume that, they have similar arrival time jitter. So, the tasks have similar arrival time jitter, and the statement is that all tasks would so similar completion time jitter.

(( )) need not be.

Need not be, because their priorities are different right. So, the lower priority task might so larger jitter, see you are saying that see they have similar or the same arrival time jitter for different tasks, but the completion time jitter might be different, because the priorities will be different.

(Refer Slide Time: 45:45)



Now, let us do one problem; not a problem actually, just a third type of question we have to give an example; not example, you have to identify a constraint on the task set, which will make RMA as proficient as EDF, we have to give a constraint on the task set, which

will make RMA as proficient as EDF or in other words, we have to give some types of tasks, for which RMA and EDF produce identical schedules, and they have similar schedulability results; the schedulability analysis become similar for them, can you think of some tasks it?

(( ))

No, but for that also EDF will be more proficient, even if the period and deadline are same.

(( ))

(( ))

Any other answer?

(( ))

No, that does not matter, because he says no that period deadline are same and zero phasing, but still EDF can produce a more it can be more schedulable its optimal; something, which is not scheduled by RMA can also be scheduled by EDF. But one thing is that when the task are harmonically related, RMA is as good as EDF, well produce similar schedules, the schedulability check is the same in the constraint on the task set is that they are period sorry integral multiples of each other.

Now, let us see this question; please tell whether this statement is true or false? In any implementation of EDF, the scheduler needs to frequently examine the ready queue of the tasks at regular intervals to determine which task to start running next.

What do you think? Any any implementation of EDF…

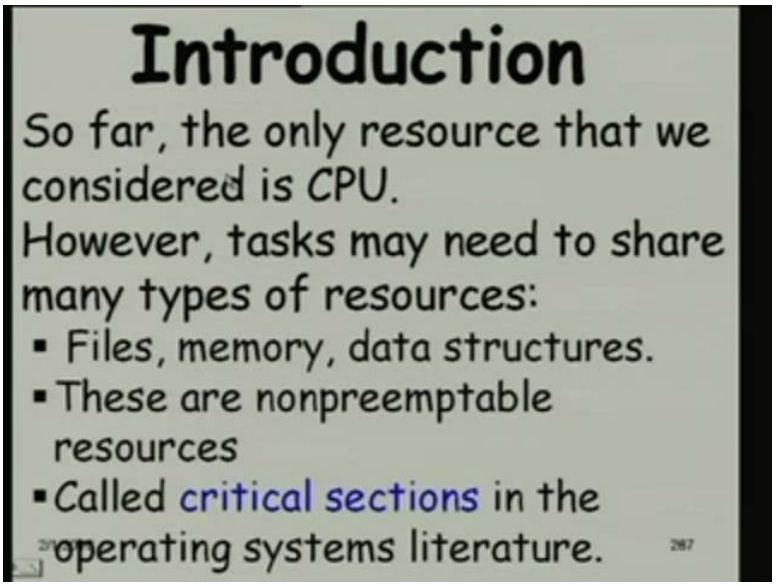
(( ))

(( ))

So, EDF is a event driven scheduler; see here you have said here that the scheduler needs to examine the ready queue of the tasks at regular intervals; regular intervals is basically,

a clock driven scheduler. So, EDF is not a clock driven scheduler, it is a event driven scheduler, and events are arrival of if tasks and completion of tasks.

(Refer Slide Time: 48:48)



So, we have so far considered a very simple situation, where we considered tasks that run independent of each other, but tasks actually need to share results with each other; one task you know, compute some result passes on to another task, take some previously computed result and so on. So, the only resource we considered so far is CPU, but tasks need to share many other types of resources for example, they can use files, memories, data structures and these are non-preemptable resources. And in the operating system literature, they are called as critical sections, but these are resources, why are they called as critical sections, because you have done, all of you done operating system course.

What is a critical section?

(( )) one (( )) enters nobody can access it while it is being used by another. So…

(( ))

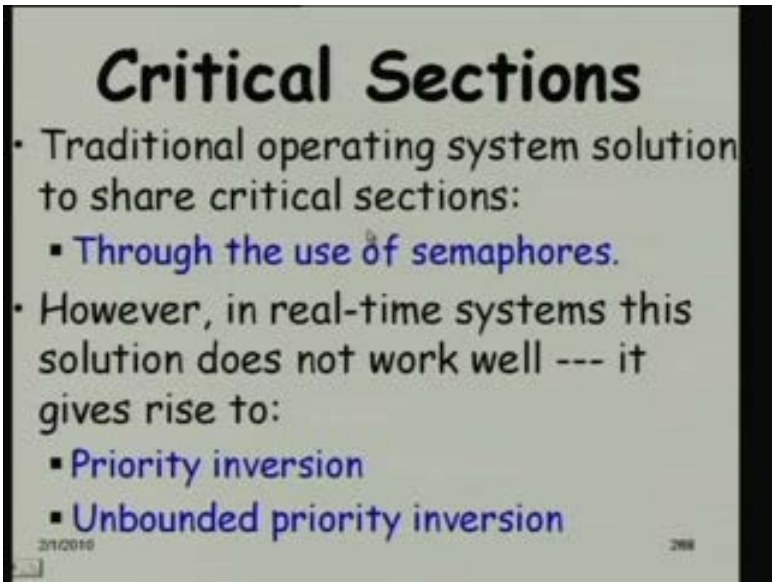
No critical…

(( )) critical, because that multiple people want to use it in a different way (( )) reading and writing are done. (( ))

Task is depending on someone else, it is not executing by themselves; it is depending on some other…

No that is ok, but we are saying that these are non-preemptable resources, what it means is that as long as the task starts using the resource, it must complete the use, and then another task might start using it. So, until it completes its use, another task will not cannot use it. So, these are called as critical sections.

(Refer Slide Time: 51:03)



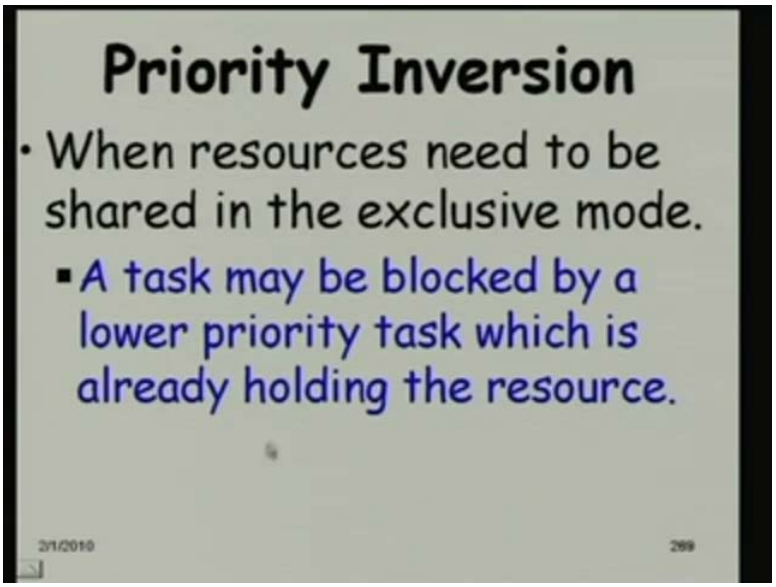
So, why we have these critical sections? So, is it section of a code or is it… what do you think?

(( ))

Just think about it, why is it called as critical section? It is a resource basically, it is a resource; it is a resource, which is non-pre-emptable. If you are task starts using it, it must complete using it; and otherwise if it half way it leaves like another tasks starts using it, it will become inconsistence like as you are saying. So, normal operating system questions like give an example of a critical section and so how it will become inconsistence so on; we will not go into that.

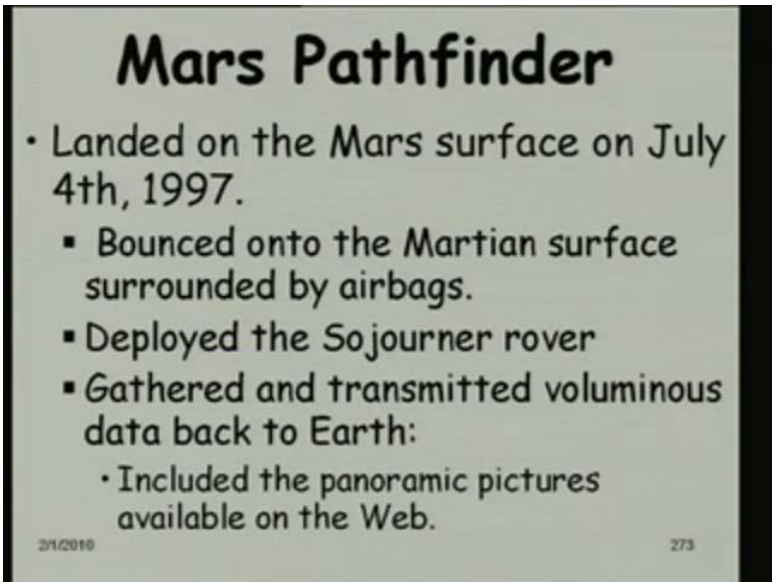
And normal operating system solution is to use semaphores; to enforce that tasks, use the non-preemptable resources; the critical sections without really preempting each other right. But these solution of semaphores, what well in the traditional operating systems, but it does not work well in the real time situations. There are too main problems; one is called as priority inversion, priority inversion is a problem; no doubt, but it is something which is not really can make a task, miss it is deadline; the one that is the severe one is actually unbounded priority inversion.

(Refer Slide Time: 52:47)



So, when a resource needs to be shared in exclusive mode, a task is blocked by a lower priority task, which is already holding a resource. So, that is the situation called a priority inversion. This is a term, which will use very frequently. So, a lower priority task is able to run, whereas a higher priority task will have to wait just, because the lower priority task is holding the resource right. So, the lower priority task is causing a priority inversion, so we will just discuss in the next lecture about the consequences of priority inversion and the unbounded priority inversion.

(Refer Slide Time: 53:52)



And the most celebrated example of the priority inversion is the mars path mars pathfinder, where they had this same thing the priority inversion almost caused the mission to fill, but they could… we will discuss about this next time.

(Refer Slide Time: 54:01)

That the mars pathfinder, they could once it is started malfunctioning, they debugged it and found that actually a priority inversion plug was not set. So, the operating system was not considering the priority inversion, how to handle this using the priority ceiling protocol. And once the plug was set, then it is started operating the the mission was

rescued it. So, we will discuss this in the next class, and see the problems that unbounded priority inversion causes, and the solutions to it, and the schedulability analysis that we need to do for the solutions; so that we will discuss in next lecture, we will stop now. Thank you.