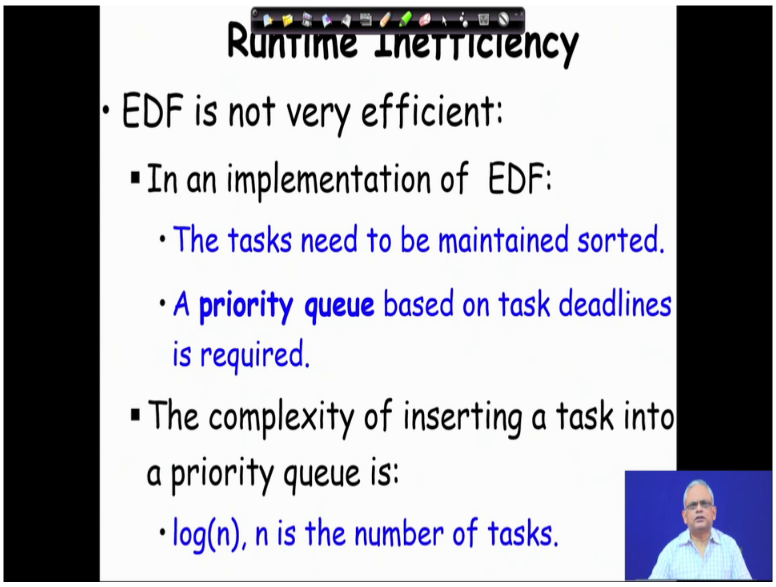
Real Time Operating System

Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 08
Rate Monotonic Algorithm

Welcome to this lecture. In this course, so far we have looked at scheduler's task schedulers.

(Refer Slide Time: 00:24)



Runtime Inefficiency

- EDF is not very efficient:
  - In an implementation of EDF:
    - The tasks need to be maintained sorted.
    - A **priority queue** based on task deadlines is required.
  - The complexity of inserting a task into a priority queue is:
    - log(n), n is the number of tasks.

And we said the task scheduling is possibly one of the most important activity of a real time operating system and we started looked at some simple real time tasks schedulers those are those goes by the name clock driven schedulers. So, those are used in very simple applications where the processing power memory etcetera are very small the operating system itself is only a small program takes very little memory and computing power.

We looked at the table driven scheduler and spent couple of lectures on cyclic schedulers because cyclic schedulers are an important category of schedulers used extensively and then we are looking at the event driven schedulers the event driven schedulers are an important category of schedulers used in larger applications where the number of tasks are more and the operating system in the significant processing overhead and also memory footprint.

We said if you remember in the previous lecture that there are large variety of event driven schedulers, but then all of them are variants of 2 basic category of schedulers one is the EDF earliest deadline first and the second is RMA rate monotonic algorithm the EDF is the optimal scheduler it is a dynamic priority scheduler and we started looking at the EDF and we were trying to find out its difficulties because EDF even though it is a very good real time tasks scheduler it can schedule it can have successful feasible schedule for tasks which no other schedulers can schedule its overhead is small, but then it is not used in common applications.

The reason is are its short comings and you are looking at the short coming three major short comings one is that if there a transient overload some task gets delayed may be because its waiting for some signal may be it just went into a larger path the code may be it went into a non-terminating loop whatever, but then if that happens even to a very non critical task very routine task still the most critical task can be missed its deadline.

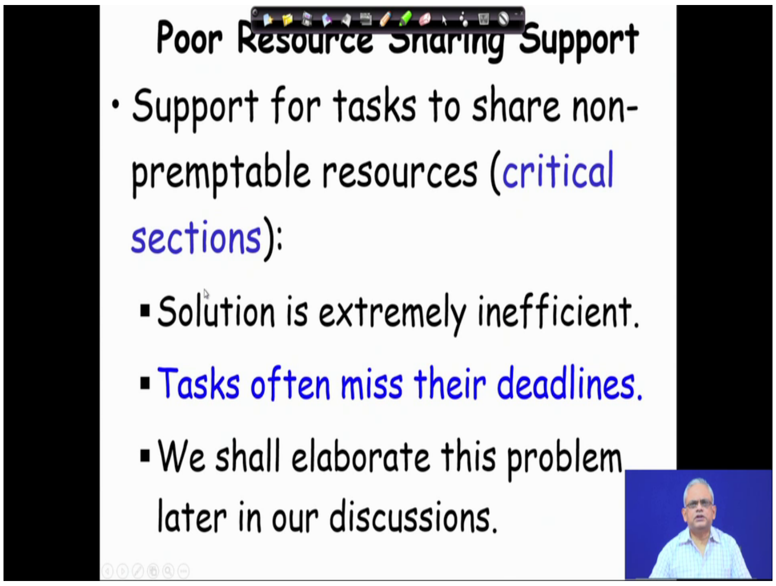
So, that is one major problem with this the second is runtime inefficiency the other schedulers are much more efficient for example, the RMA based schedulers are much more efficient let us see the source of inefficiency of EDF scheduler. So, if you remember at every scheduling point the scheduler becomes active and then selects the next task to run in the EDF algorithm it checks all the ready tasks find out, which has the earliest deadline to do this it must have the tasks in some data structure lets first examine if all the tasks are ready tasks are waiting in a queue. So, I need to traverse the queue and find out which is the task having the earliest deadline.

Traversing a queue is order of in if n tasks are there then o n is the complexity of finding the task of course, when a task gets generated inserting in the queue o one because it gets inserted at the end of the queue o n at every scheduling point is actually inefficient we should ideally have o one as the complexity because if the tasks are more and the scheduling points occur very frequently every few micro seconds or so, then the runtime overhead becomes significant.

Now let us see what are the other options for implementing the EDF scheduler one is a priority queue if you can recollect what is a priority queue it is the one; it is a type of queue where the highest priority task is available at the top of the queue. So, finding the task having the shortest priority is o 1, but then what about inserting a task if you look into the complexity of inserting into a priority queue find that it is log n. So, even though it is a better structure than a singly linked list a linear queue, but still log n is not a very desirable time complexity for a task or a scheduler.

Therefore we can say that EDF is not really very runtime efficient.

(Refer Slide Time: 07:16)



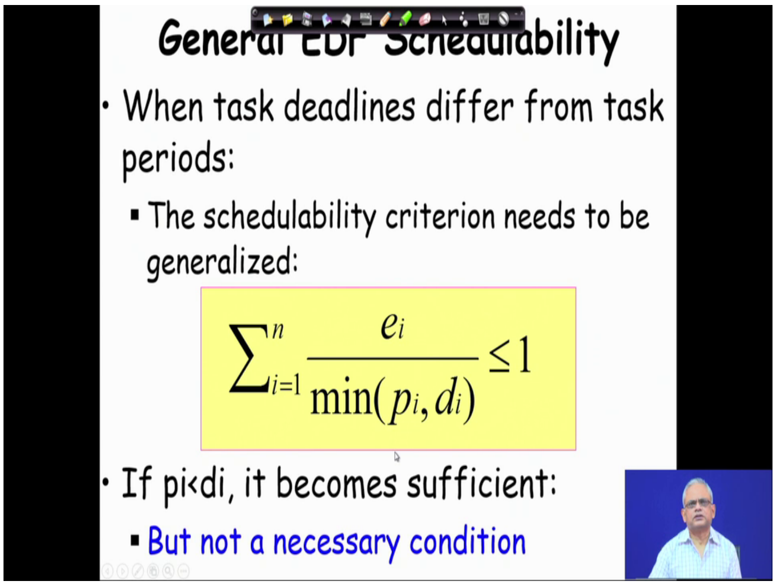
Poor Resource Sharing Support

- Support for tasks to share non-premptable resources (critical sections):
  - Solution is extremely inefficient.
  - Tasks often miss their deadlines.
  - We shall elaborate this problem later in our discussions.

The most important set coming that is faced in a application real time application development is poor resource sharing support in EDF based scheduler in a practical application the real time tasks need to share resources; resources can be some data items based in memory it can be open files etcetera.

So, these are called as a critical sections if we look at the solutions that exists for resource among tasks in EDF and critical resources we are talking of then the solution is extremely inefficient and this causes the tasks to miss their deadlines as we proceed in couple of lectures, we will examine the issue of resource sharing we will see that for rate monotonic based schedulers very good algorithms are available for resource sharing. But for EDF resource sharing is a big problem, we do not have a satisfactory solution for that and possibly that is one of the main reasons why EDF is not used in practical real applications.

(Refer Slide Time: 09:01)



General EDF Schedulability

- When task deadlines differ from task periods:
  - The schedulability criterion needs to be generalized:

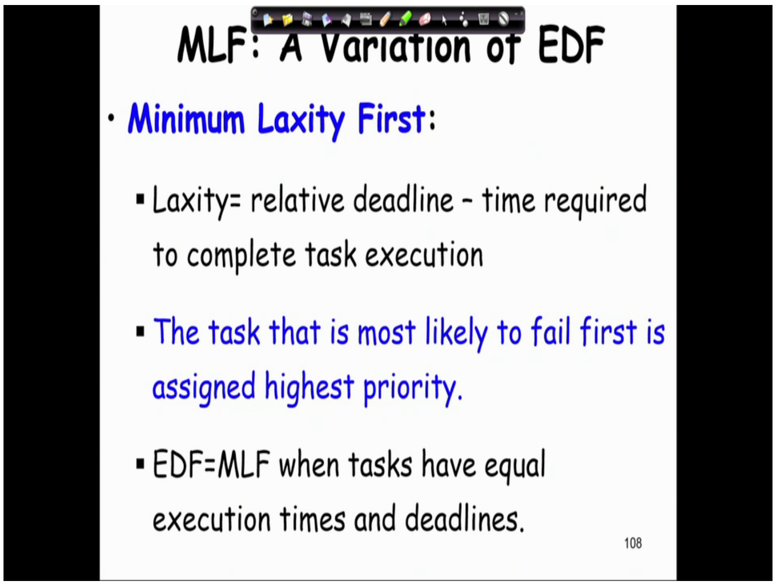
$$\sum_{i=1}^{n} \frac{e_i}{\min(p_i, d_i)} \leq 1$$

- If pi<di, it becomes sufficient:
  - But not a necessary condition

But then before concluding our discussion in EDF, let us look at one point that is we have. So, far assumed that the for every task the period and deadline coincide the period is equal to deadline, but in many situations not many, but then in some situations the deadline and the period can be different for example, deadline can be less than the period or rare cases deadline can be more than the period.

So, in these cases how do we check EDF schedulability one solution is to take the minimum of p i d i. So, e by e i by minimum of p i d i that gives the utilization and see that that is less than 1; this is a sufficient condition that is if this is satisfied then of course, the tasks set is schedulable in EDF, but then it is not a necessary condition because we can find that some sets is not meeting this criterion this is too stricter criterion minimum of p i d i it doesnt meet the criterion. But then still it may be possible to feasibly schedule this we will not go into details of that, but just to know that if p i is not equal to d I, then we need to use e i by mean p i d i which is a sufficient schedulability condition, but then there can be some cases where they do not need this criterion, but still becomes schedulable.

(Refer Slide Time: 11:13)



MLF: A Variation of EDF

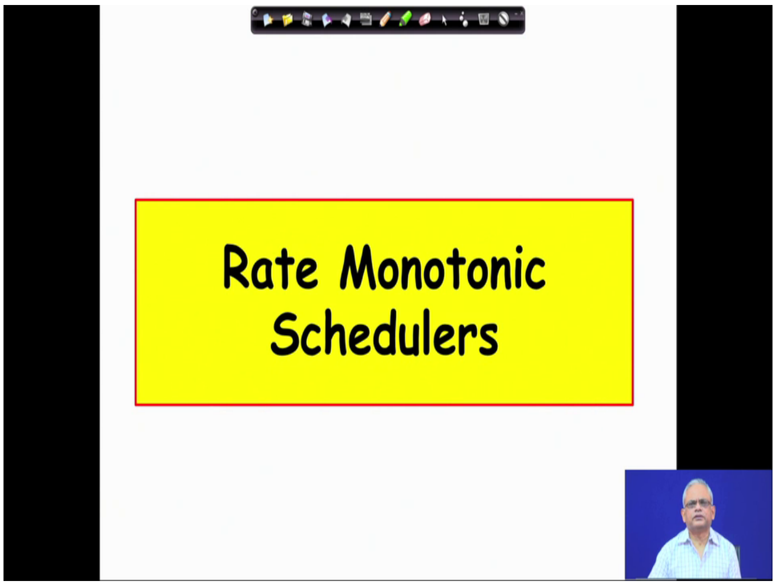
- Minimum Laxity First:
  - Laxity= relative deadline - time required to complete task execution
  - The task that is most likely to fail first is assigned highest priority.
  - EDF=MLF when tasks have equal execution times and deadlines.

108

There are some variations actually many variations of EDF one important variation is called as a minimum laxity first in EDF we look at only the deadline how much deadline is remaining. But then we do not consider how much execution time is required over the deadline if the deadline is 100 micro seconds and the computation time required is 5 micro second compared to another task whose deadline is 100 micro second. But the computation time required is 50 micro seconds then it may be better to give higher priority to the task having more computation left over the same deadline.

That is the main idea here in the MLF. So, here the laxity is defined as the deadline minus the time required to compute the task execution and this turns out to be a good algorithm because we are considering the task that is most likely to fail first is assigned the highest priority.

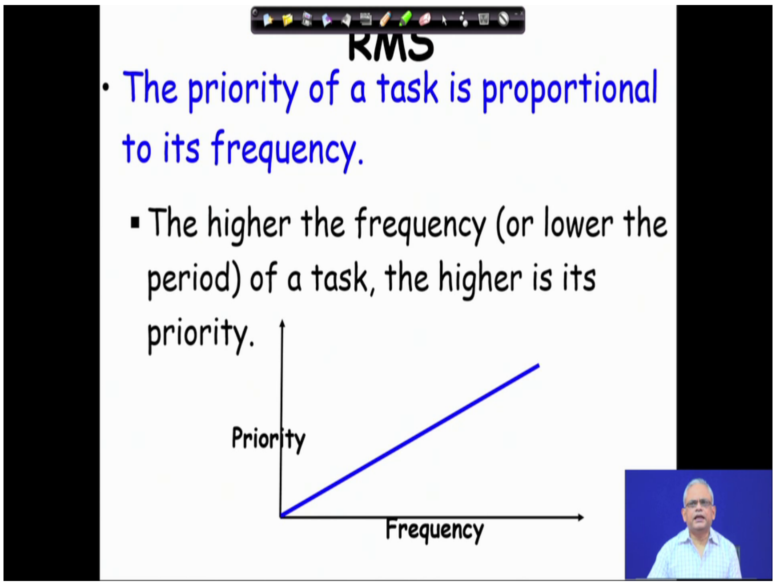
(Refer Slide Time: 12:41)



Rate Monotonic Schedulers

And it performs at least as good as the e d f, but then in some cases it can perform better than EDF now we have concluded our discussion on EDF the earliest deadline first algorithm that is the optimal scheduling algorithm now we will start ha discussing the rate monotonic schedulers, these are I think the most important class of real time schedulers even though these are not proficient than the EDF, but practical implementation of these is much simpler and also more efficient and the set comings of the EDF like resource sharing and so on. Become easy here in the rate monotonic scheduler and therefore, this these schedulers are used extensively in many real time applications.

(Refer Slide Time: 13:44)



RMS

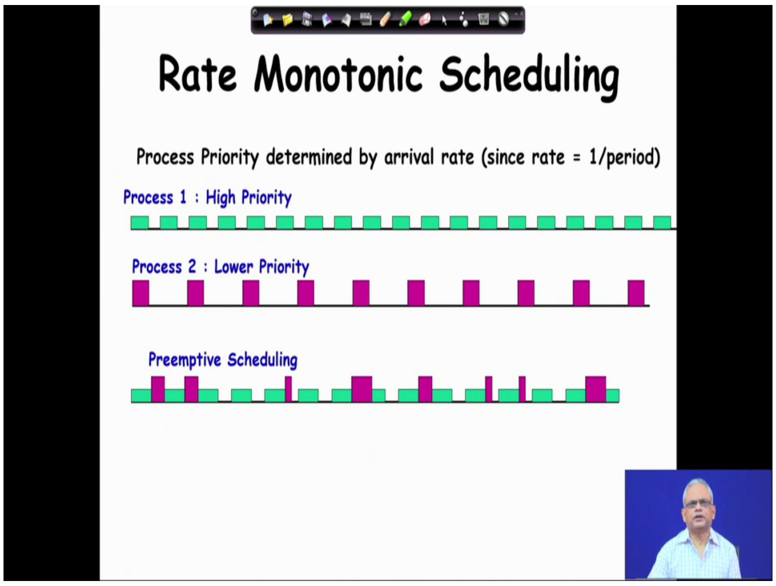
- The priority of a task is proportional to its frequency.
  - The higher the frequency (or lower the period) of a task, the higher is its priority.

Priority
Frequency

The main idea in the rate monotonic scheduler is that the priority of a task is proportional to its rate of arrival. Let me repeat that here the priority is statically allocated by the programmer or the designer and the priority that is assigned to a task should be proportional to the rate at which the task arrives or the proportional to its frequency if a task the period is 2 millisecond. It should definitely have higher priority compared to a task whose period is 10 millisecond.

So, the higher the frequency or lower the period of a task the higher should be its priority that is the basic premise or the basic algorithm for this scheduler. So, if the frequency or the rate of arrival or whatever we give name to this if we arrange the tasks in this order then the priority should be increasing in order of their frequency or decreasing in terms of their period. So, the priority is increasing function of the task arrival rate.

(Refer Slide Time: 15:30)



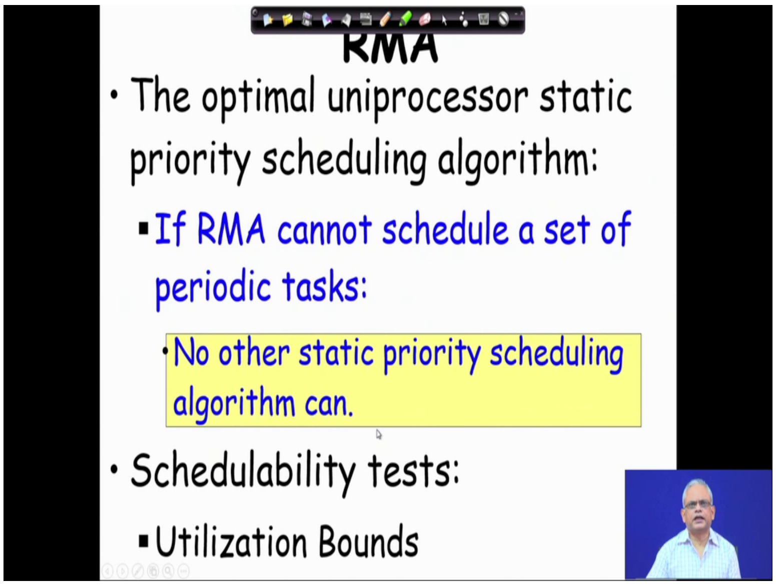
Rate Monotonic Scheduling

Process Priority determined by arrival rate (since rate = 1/period)

Just to give an example, look at the task one its repeating frequently rate of arrival is very high compared to the task 2 rate of arrival is lower and therefore, task one has higher priority and task 2 has lower priority and the essence of this scheduler is that the higher priority task will pre-empt any lower priority task and start running on its arrival.

So, let us look at this actual execution whether that is happening a time 0; both task 1 and task 2 instance are ready, but then task one has higher priority and therefore, this instance starts running the first 2 instance has to wait and then as soon as the task one instances completes execution then the task 2 instance starts running and then the next instance of task one arrives it pre-empts the task 2 starts running and then as it completes the task 2 again become active and so on. So, as long as there is high priority task the lower priority task cannot run and it is the higher priority task that runs that is the main concept in rate monotonic scheduling.

(Refer Slide Time: 17:20)



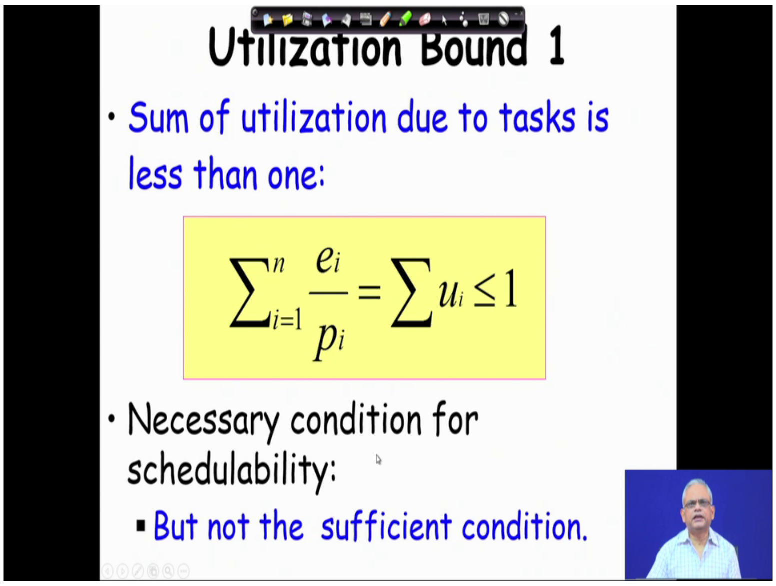
RMA

- The optimal uniprocessor static priority scheduling algorithm:
  - If RMA cannot schedule a set of periodic tasks:
    - No other static priority scheduling algorithm can.
- Schedulability tests:
  - Utilization Bounds

One important result is that among all static priority scheduling algorithms RMA the rate monotonic algorithm is the optimal algorithm. So, if RMA cannot schedule a set of periodic tasks no other static priority scheduling algorithms can, but on the other hand any task set that is scheduled by any static priority tasks scheduler can also be run under RMA is a very good static priority algorithm.

Now, let us spend some time on the results that are available on the schedulability analysis. So, that is given a task set can we say that it can be feasibly scheduled on a rate monotonic scheduler. So, those results are typically given as utilization bounds what is the utilization. So, the tasks set or the processor and based on that we can conclude whether it can run feasibly on a processor under the RMA scheduler.

(Refer Slide Time: 18:53)



Utilization Bound 1

- Sum of utilization due to tasks is less than one:

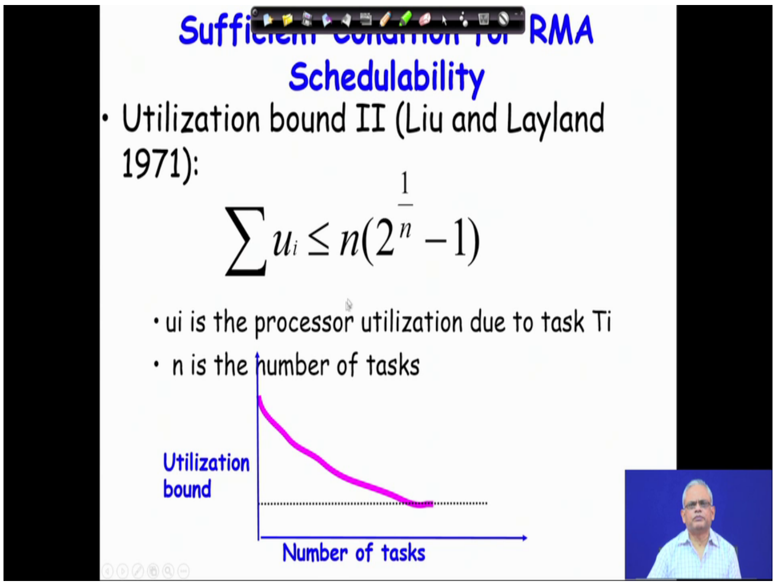
$$\sum_{i=1}^{n} \frac{e_i}{p_i} = \sum u_i \leq 1$$

- Necessary condition for schedulability:
  - But not the sufficient condition.

The first result available since long time since 1971, I think is called as the utilization bound one this s the basic result. So, this is available since long time that the utilization is less than 1 obvious because utilization of a processor cannot be one and this is natural that the sigma e i by p i should be less than 1 and this is a very basic condition called as the utilization bound one these are necessary condition, but then a tasks set meets this e i by p i sigma e i by p i is less than 1 where e is the execution time and p is the period we need to check further conditions just because tasks set has met this condition does not meet mean that the task set is schedulable.

But if you remember in the EDF schedulers this was both the necessary and the sufficient condition the same expression sigma e i by p i less than equal to one was both necessary and sufficient condition for the EDF schedulers, but here it is just the necessary condition you need to first check whether e i by p i sigma is less than 1, then we can look for other utilization conditions otherwise even this condition is not met; obviously, it will not be schedulable under the rate monotonic algorithm.

(Refer Slide Time: 20:47)



Suffi[illegible] RMA Schedulability

- Utilization bound II (Liu and Layland 1971):

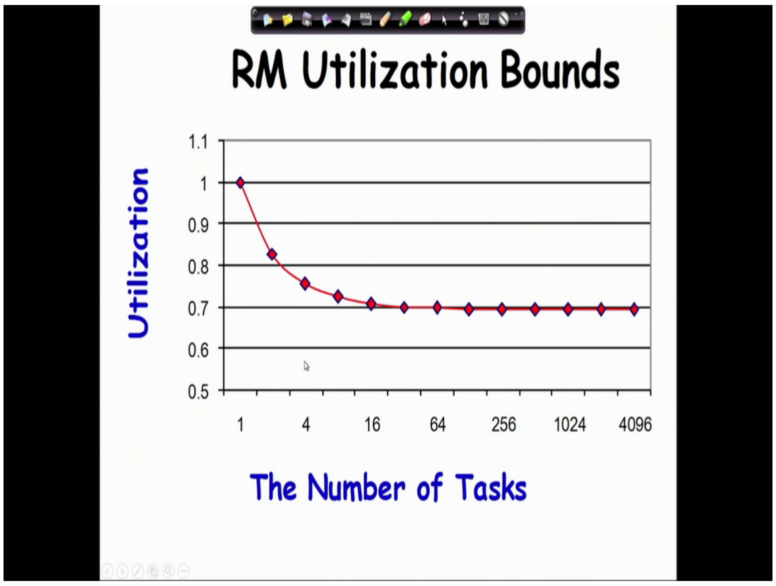
$$\sum u_i \leq n(2^{\frac{1}{n}} - 1)$$

- ui is the processor utilization due to task Ti
- n is the number of tasks

The second result is again a utilization bound available for long time nineteen 70 one Liu and Layland proposed in a landmark paper that the utilization bound should be less than n into 2 to the power 1 by n minus 1 for a tasks set to be schedulable. So, if there are 10 tasks. So, 10 into 2 to the power 1 by 10 minus 1 the utilization sum of utilization of due to the tasks should be less than that of course, you can check that it is always less than 1 because even if n is equal to infinite. So, infinite into 2 to the power 1 by infinity minus 1. So, that is a indeterminate form, but if we use the l hospitals rule we will find that it is 0.6. We will come to that.

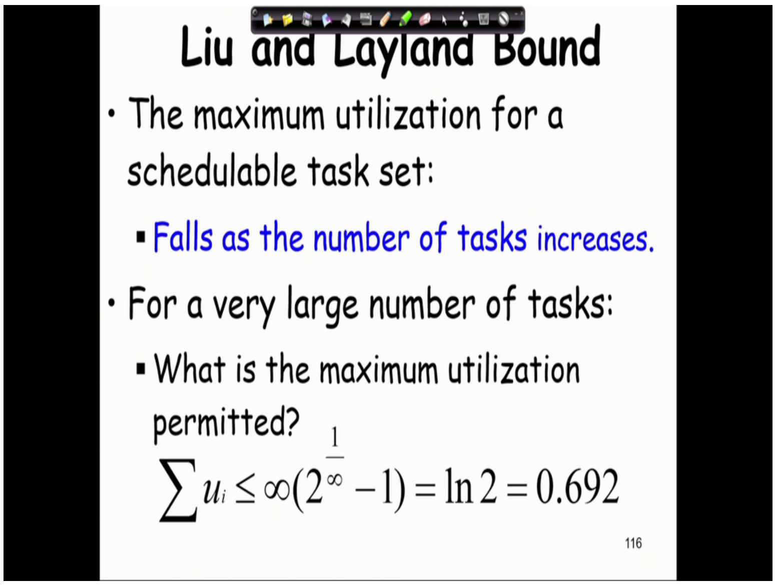
Let us see how the utilization bound varies with number of tasks if n is equal to 1, then it becomes 1 into 2 to the power 1 minus 1 which is 100 percent. So, if only one task is there, then even if it has 100 percent utilization of the processor, still it can be schedulable under the rate monotonic algorithm, but what if there are 2 tasks it becomes 2 into 2 to the power 1 by 2 that is square root f 2 minus 1 and square; square root of 2 is 1.41 minus 1 is 0.41 into 2 becomes 0.82. So, from point sorry from one, it becomes 0.82 with 2 tasks for three tasks it further reduces and so on, but finally, it saturates at some value; let us find out what is this value because according to this expression it has n increases the number of tasks increases the utilization at which the task set is schedulable decreases.

(Refer Slide Time: 23:38)



So, this is the behaviour if you really take number of tasks 1, 2, 3, 4, 5 etcetera we will see that by the time, there are 10 tasks it almost settles in this 0.7. So, 0.7 or 70 percent utilization if a tasks set has more than 0.7 or 70 percent utilization we can safely assume that it is schedulable. So, if sorry; I need to re state what is said that if the utilization is less than this if the utilization is less than 70 percent anywhere in this region all this the tasks that are utilization is in this region that is less than 70 percent this is upper bound of utilization actually. So, as long as we find that a task set the utilization is less than 70 percent, we can conclude that it can be feasibly scheduled by a rate monotonic scheduler.

(Refer Slide Time: 25:02)



Liu and Layland Bound

- The maximum utilization for a schedulable task set:
  - Falls as the number of tasks increases.
- For a very large number of tasks:
  - What is the maximum utilization permitted?

$$\sum u_i \leq \infty(2^{\frac{1}{\infty}} - 1) = \ln 2 = 0.692$$

116

As given by this Liu and Layland bound, the maximum utilization at which the tasks becomes schedulable falls as the number of tasks increases and for very large number of tasks it settles at around 0.7 to be precise it is 0.692 and how do we get this 0.692, it is based on the solving the indeterminate form infinity to the power infinity into 0 indeterminate expression applying the l hospitals rule to this indeterminate expression we get log 2 which is 69 percent 69.2 percent.

The main thing to note here is that as long as the task set the utilization is 69.2 percent we can safely say that the task set is schedulable under Liu Layland bound, but if it is not schedulable it its utilization is more than 69.2, then we have to check how many tasks if the task is let us say 2, 3, 4, etcetera, then we need to compute this expression n into 2 to the power 1 by n minus 1 as I was mentioning that for 2 tasks even up to 82 percent utilization the tasks set can be feasibly scheduled by the rate monotonic scheduler.

Because the rate monotonic scheduler is important class of scheduler used extensively in many real time applications and also supported by almost every commercial real time operating system, we are just trying to study this scheduling algorithm slightly more carefully and the next lecture we will continue from this point we will conclude now for this lecture.

Thank you very much.