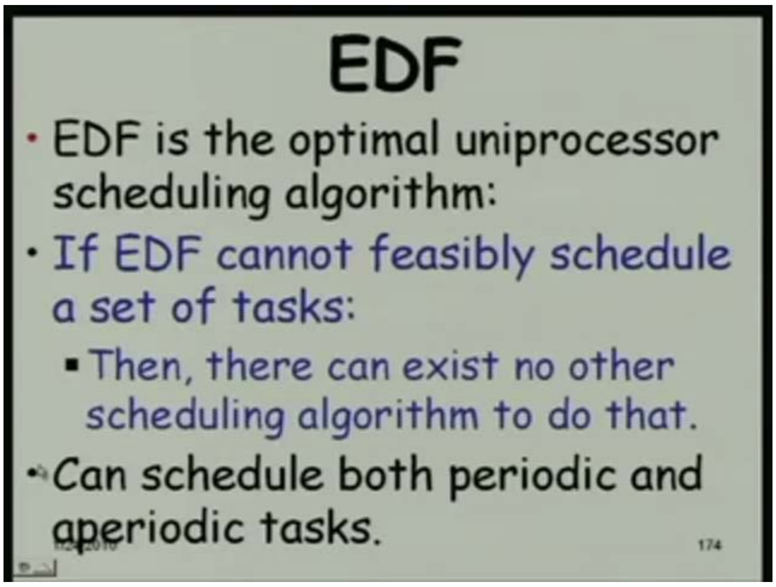
Real – Time Systems

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

Module No. # 0
Lecture No. # 09
Rate Monotonic Scheduler

Good morning. So, let us get started from where we left last time, so if you remember last time we were looking at the earliest deadline first scheduler the E D F scheduler. We had said that it is a good one, its very good scheduling algorithm rarely used and so on.



EDF

- EDF is the optimal uniprocessor scheduling algorithm:
- If EDF cannot feasibly schedule a set of tasks:
  - Then, there can exist no other scheduling algorithm to do that.
- Can schedule both periodic and aperiodic tasks.

174

So, let us just catch up on that and proceed from there on.

(Refer Slide Time: 00:42)

So, we had said that it is the optimal algorithm, E D F is the optimal algorithm and if set up task cannot be scheduled using E D F, then we can have no other algorithm that can successfully schedule the set up tasks.

It also can schedule both periodic and a periodic tasks (( )) positive points and we had said that, what are the scheduling points for E D F? What do you think, what are the scheduling points for E D F? Yes, anybody likes to answer.

(( ))

No, what is the scheduling point? A scheduling point can anybody say, what is a scheduling point for any real time task scheduler.

(( ))

No, the scheduling point is the point, on the time line at which the scheduler wakes up and decides which task to run next .In a time driven scheduler we had said that at periodic clock intervals the scheduler wakes up and decides what to do next right.

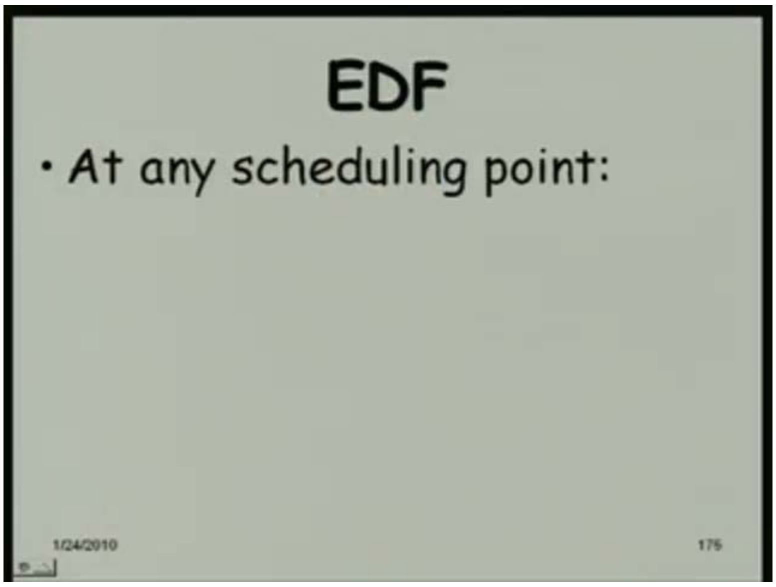
Possibly ,it consults a table tries to see which tasks need to run now, but what about E D F, what are the scheduling points?

Anybody like to answer, is it a time driven scheduler or an event driven scheduler, what about E D F? What do you think? Is it a clock driven scheduler or is it an event driven scheduler, no we had said that that clock driven schedulers are the simplest ones and we had discussed two clock driven schedulers.; one was the table driven scheduler and the other was the cyclic scheduler. And we had said that those are simple used in very simple applications, the main difficulty with that was when the number of events increases it becomes very difficult to construct a schedule.

And also for a periodic and sporadic tasks, the clock driven scheduler cannot handle them, because they occur any time not according to the clocks, then we had discussed about the event driven scheduling. Said that, any application that is slightly complicated many events are there and some of the events are a periodic or sporadic then we will have to go for a event driven scheduler and then we said that lot of results are available.

But, the two algorithms which actually summarize the all the results in this area, if we know those two algorithms we know all about the event driven scheduling on a uniprocessor and those we said the E D F or the earliest deadline first algorithm and the rate monotonic algorithm and how does a E D F work? Can somebody answer this question?

(Refer Slide Time: 03:58)



EDF

- At any scheduling point:

1/24/2010 175

How does the E D F scheduler work?

Whenever the task (( )).

Yes.

(( ))

Yes.

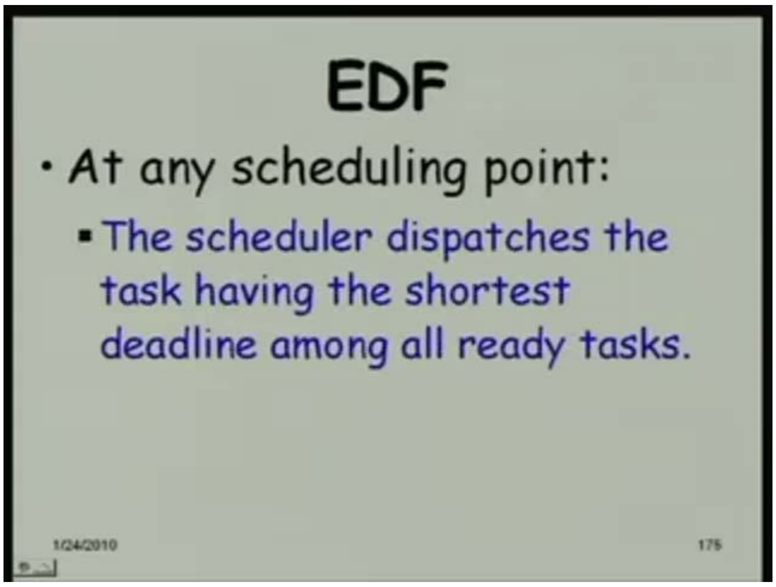
And then if the entire deadline (( )).

It finds the task which has the shortest deadline and selects that, and when does that do it at what points?

(( ))

But, what are the scheduling points correct its does that at scheduling points, but how are those scheduling points known? Or on timeline at, what points on the timeline will the scheduler wakeup? Whenever the event (( )) when a new task arise, Yes. That time, we have to see? Whether the deadline. Of course, yes. So, a task arrival and also task completion, these are two events that define the scheduling points, the task arrival and the task completion.

(Refer Slide Time: 04:53)



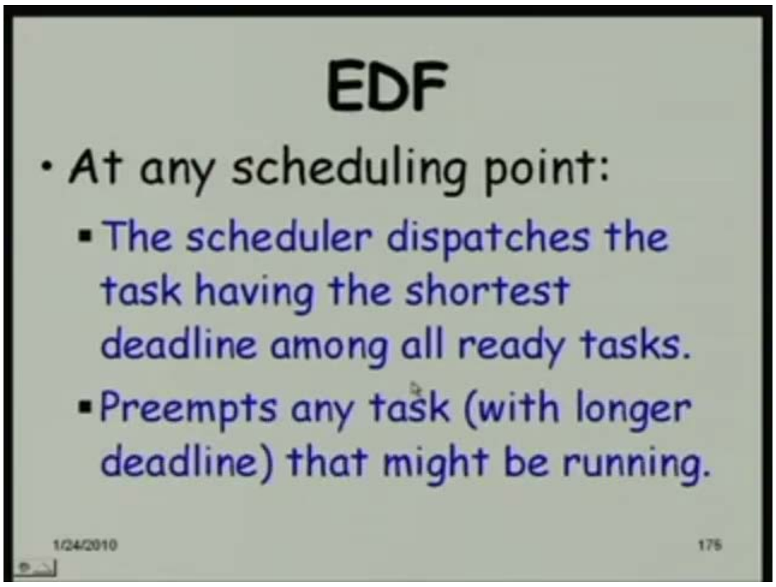
EDF

- At any scheduling point:
  - The scheduler dispatches the task having the shortest deadline among all ready tasks.

1/24/2010 175

So, at any scheduling point, the scheduler dispatches the task having the shortest deadline among all ready tasks very simple algorithm. Checks the tasks, the one that has the shortest deadline it dispatches it.

(Refer Slide Time: 05:14)



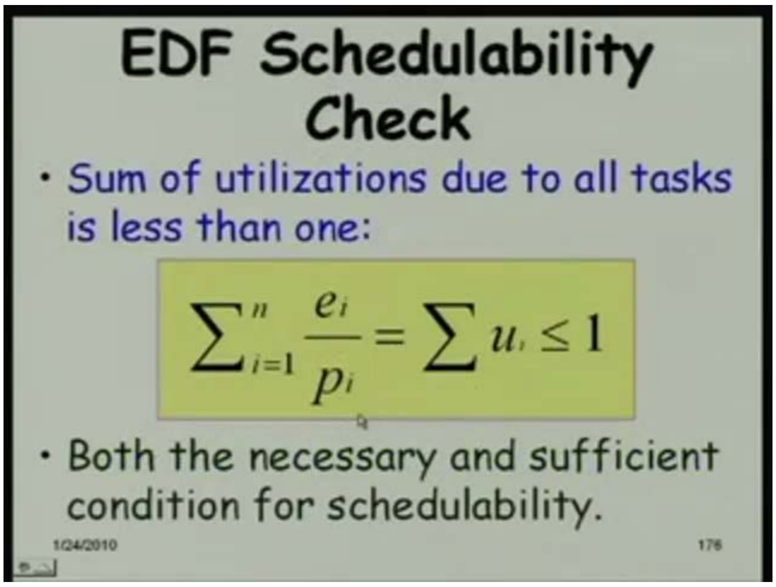
EDF

- At any scheduling point:
  - The scheduler dispatches the task having the shortest deadline among all ready tasks.
  - Preempts any task (with longer deadline) that might be running.

1/24/2010 175

And of course, if there is any task that is already running then this will preempt it and let the task which has the shortest deadline run very simple algorithm.

(Refer Slide Time: 05:28)



EDF Schedulability Check

- Sum of utilizations due to all tasks is less than one:

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

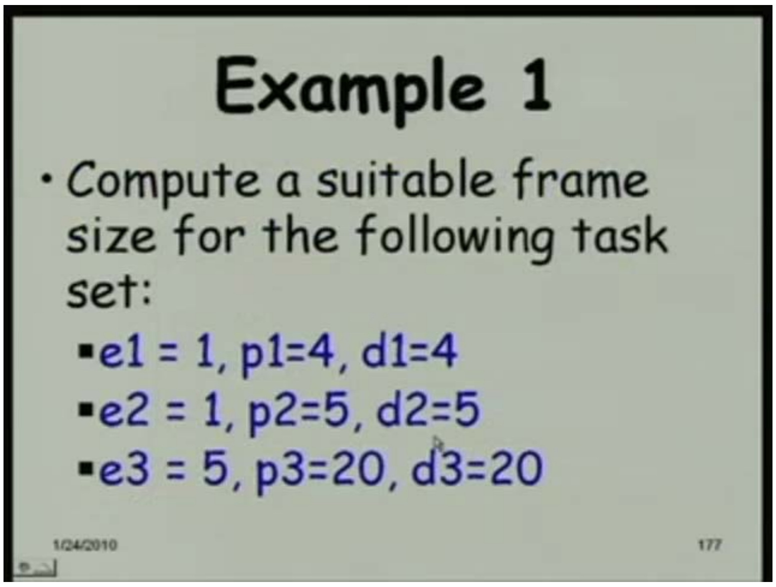
- Both the necessary and sufficient condition for schedulability.

1/24/2010 176

Now, we had tried to define a check on given a set of tasks, how do I know? Whether it will be schedulable or not said that the check in simple, check whether the utilization due to all the tasks is less than one. And this is true for any scheduler, why E D F? Even the traditional scheduler as long as the utilization due to a task is more than one it cannot meaningfully run listen it.

So, this restriction is actually obvious or any scheduler it holds that if the computation time requirement. For the time in which it arises the sum of all that is more than one it cannot be scheduled. So, it is very simple condition here and this is both the necessary condition and a sufficient condition for scheduling it.

(Refer Slide Time: 06:29)


Example 1

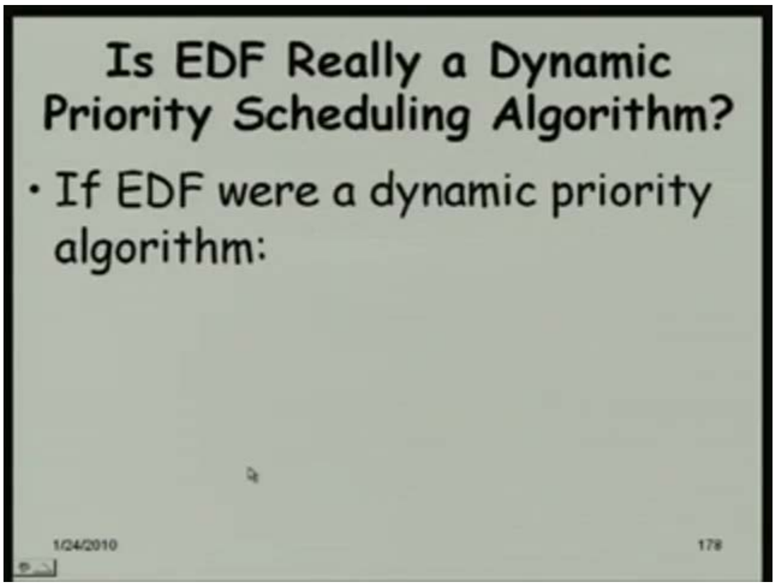
- Compute a suitable frame size for the following task set:
  - e1 = 1, p1=4, d1=4
  - e2 = 1, p2=5, d2=5
  - e3 = 5, p3=20, d3=20

1/24/2010 177

So, we had looked at some examples last time and we had said that see if given any three tasks or any set up tasks, how do you decide? Whether it will run using a E D F schedule successfully or not. The only thing that, you will have to do is? Find out the utilization that is e i by p i for each of this tasks and see, here the period and deadline is the same.

That is true for almost every real time task that you will encounter rare cases the deadline will be different from the period. So, find the utilization one by four plus one by five plus five by twenty sum it up checks whether it is less than one or not as long as it is less than one it will run successfully using E D F.

(Refer Slide Time: 07:23)



Is EDF Really a Dynamic Priority Scheduling Algorithm?

- If EDF were a dynamic priority algorithm:

1/24/2010 178

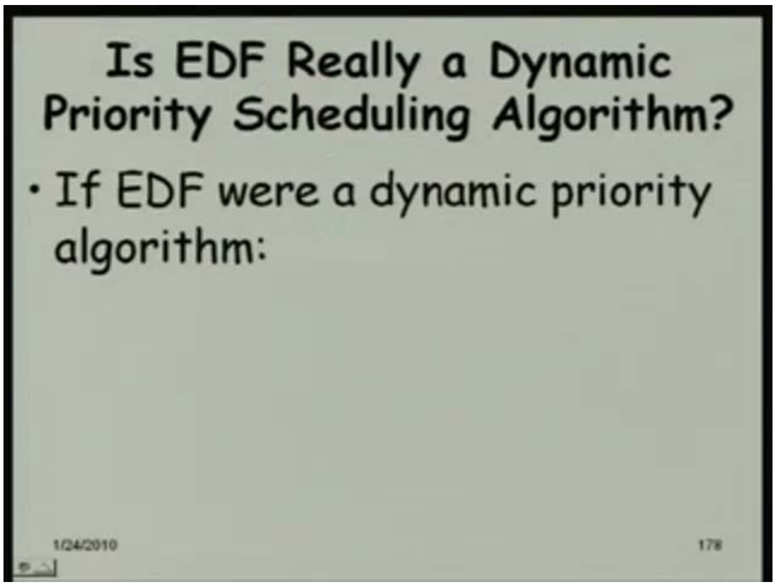
And then we were discussing that, why do we call it as a dynamic priority algorithm? Because, it does not have a concept of priority at all, it just finds the one that is shortest deadline and just runs it. But then we said that see if you look at the working of the algorithm it just a task that is waiting for sometime slowly the chances of it getting selected for run increases listen it.

So, we can think of a virtual priority that is associated with tasks and which keeps on increasing at the tasks (( )) right. So, there is no real priority value associated with tasks.

Priority (( ))

Yes, that is a virtual value that is for our own understanding, but actually it works without using a concept of a priority just looks at the deadlines and just schedules right. So, that we had discussed now let us proceed further. So, we had said that we cannot really determine the priority value of a task at any point you can give a number 20 or 30. And we cannot really say that from 20 it has become 19 or 21 etcetera no.

(Refer Slide Time: 08:24)

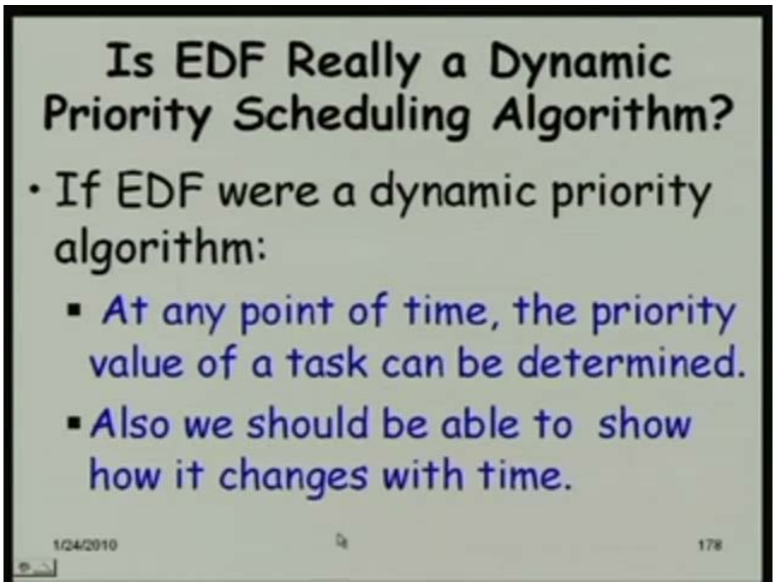


Is EDF Really a Dynamic Priority Scheduling Algorithm?

- If EDF were a dynamic priority algorithm:

1/24/2010 178

(Refer Slide Time: 08:37)

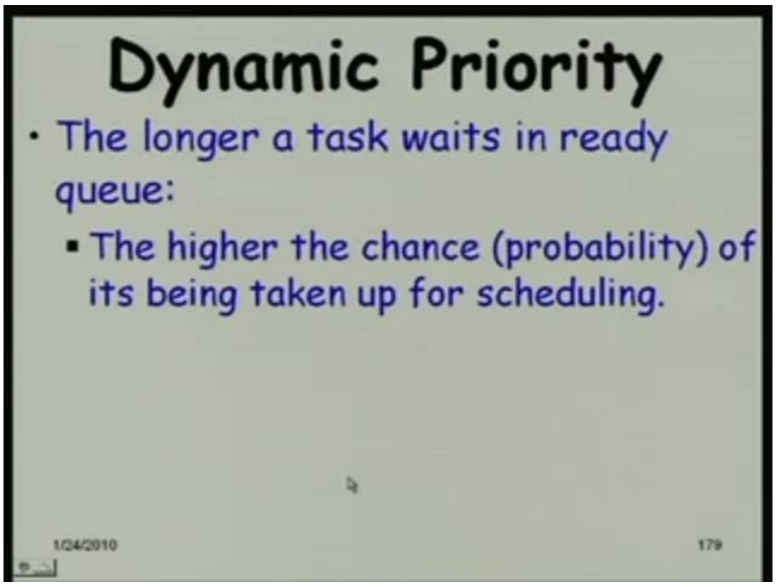


Is EDF Really a Dynamic Priority Scheduling Algorithm?

- If EDF were a dynamic priority algorithm:
  - At any point of time, the priority value of a task can be determined.
  - Also we should be able to show how it changes with time.

1/24/2010 178

(Refer Slide Time: 08:45)

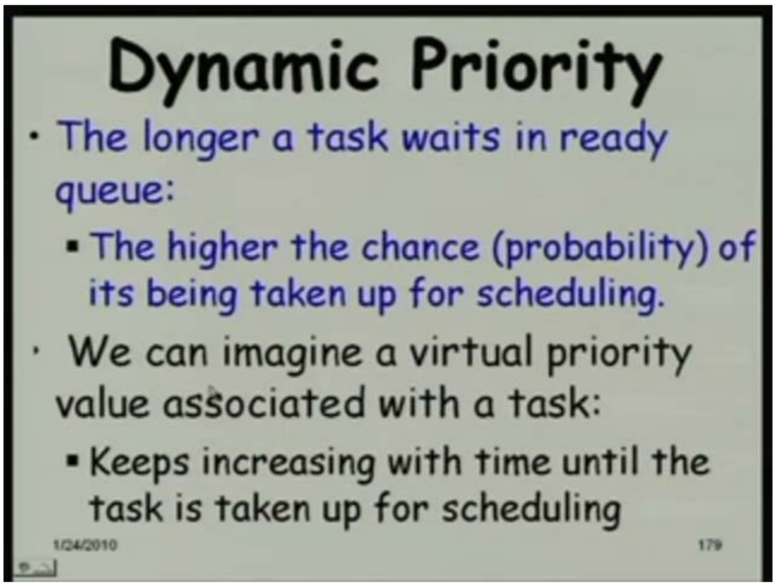


Dynamic Priority

- The longer a task waits in ready queue:
  - The higher the chance (probability) of its being taken up for scheduling.

1/24/2010 179

(Refer Slide Time: 08:47)



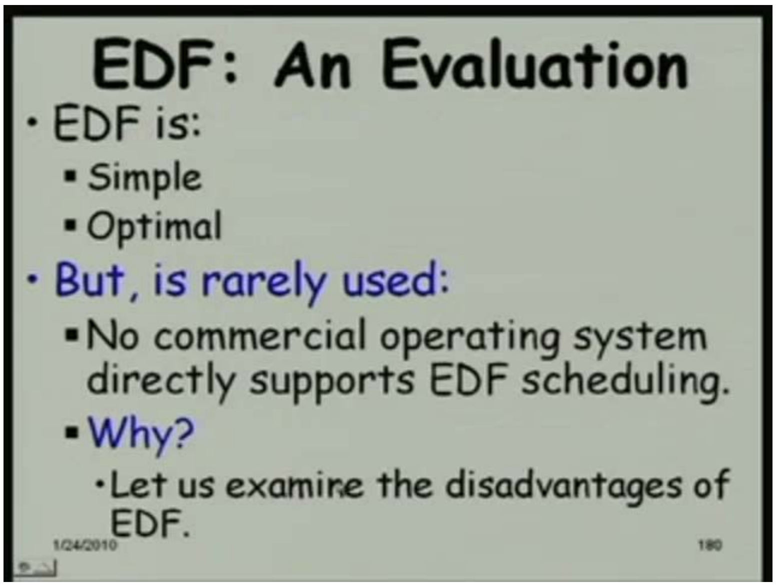
Dynamic Priority

- The longer a task waits in ready queue:
  - The higher the chance (probability) of its being taken up for scheduling.
- We can imagine a virtual priority value associated with a task:
  - Keeps increasing with time until the task is taken up for scheduling

1/24/2010 179

It is only a virtual priority value which we can imagine a value associated with a task which keeps on increasing with time until the task is taken for scheduling.

(Refer Slide Time: 08:57)



EDF: An Evaluation

- EDF is:
  - Simple
  - Optimal
- But, is rarely used:
  - No commercial operating system directly supports EDF scheduling.
  - Why?
    - Let us examine the disadvantages of EDF.

1/24/2010 180

If we look at the E D F critically, if we try to evaluate it we will see that it is a very simple algorithm just looks at the deadlines and selects the shortest one. But it is optimal if it cannot do something, it cannot successfully run a set of tasks no other algorithm can run. But is rarely used when we look at the commercial operating systems you will see, that most none of the m directly supports E D F scheduling there must be lot of disadvantages.

Sir, does it support indirectly.

Indirectly you can write a scheduler, yes. See you have you know you can write a scheduler a custom scheduler for your operating system most of the commercial operating system will see that they give you a facility to write your own scheduler and to run it and even select between two three scheduler as the system runs.

Practice whether it is (( )).

In very rare cases E D F is used very rare cases, because of its problems if we have an application for which this problems are nothing then E D F is a good algorithm. Let us see the problems.

Sir (( ))

Yes.

(( )) if task having 5 milliseconds

Yes.

This is running the another task is coming

Yes.

(( )) deadline only 3 milliseconds

Yes.

Its (( )) that (())

5 milliseconds.

Taking a 3 millisecond.

Right.

Another task is coming at two milliseconds.

Yes.

The period,

Yes.

So, it will again preempt this 3 millisecond and.

Exactly, see the question that he asked now, is that what? if one task is running that has lets ten millisecond as the deadline that task was running because, that was the shortest deadline that time. And as it was running we had another task which just arrived, which are 5 millisecond deadlines. So, that task will be taken off this task will be preempted and then as the 5 millisecond deadline task was running another task came very short task which had a 3 millisecond deadline. So, that task will be taken off and after that completes the 5 millisecond task will be taken off. Then it may make the deadline.

See, as long as the utilization is less than one it will ensure that all the deadlines are met you can just tryout in your note book take task combinations see if 3 is a situation, where the deadline is missed and the utilization is less than one. Just let us construct such an example of course; theoretical it has been proved very easy to prove actually that if the utilization is less than one then the deadline will be guaranteed to meet.

That has been that we can prove easily, I did not take up the proof, but you can also try proving that right, does that conserve your question? Yes.

Now, the 10 millisecond thing.

Yes.

6 milliseconds are over.

Yes.

Another 4 millisecond is left,

Yes.

That time if 5 millisecond is occurring,

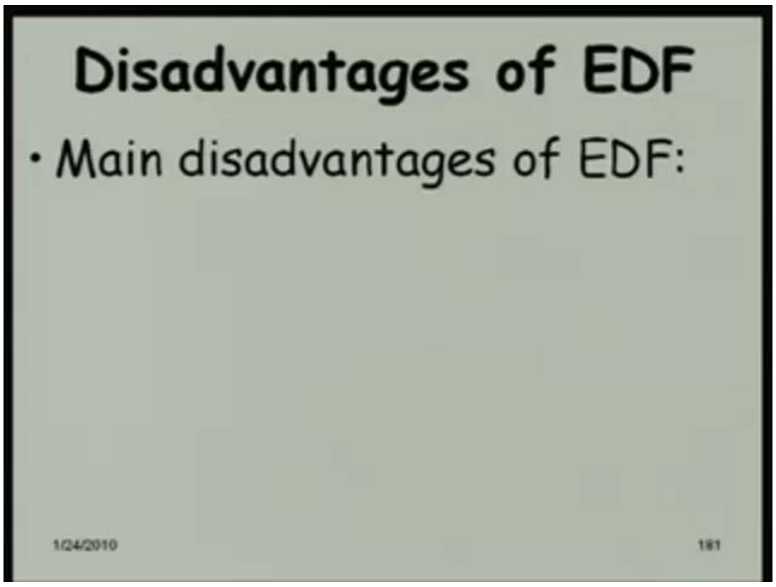
Yes.

The period causes the 5 millisecond It is occurred now, what it will do sir? Whether, it will take a 10 millisecond (( )) or 5 millisecond.

So the question is that, a task which had 10 millisecond as deadline has computed up to let us say it started sometime and at 6 millisecond it had not completed. So, it has only 4 millisecond remaining and there is another task which has come up which has 5 millisecond deadlines. So, the nearest deadline is the one which had you know 10 millisecond listen it.

Sir, on the timeline we have to see? Yes. We have to see, what is the deadline remaining for it? So, the one which had ten millisecond at some point of time. Now, at another point time it has only 4 millisecond left and a task which has come which has another 5 millisecond deadline; obviously, the one which has 4 millisecond will run listen it, yes. So let us proceed, we said that there must be severe disadvantages of this algorithm otherwise people would love to use this it is a good algorithm simple efficient let me not use the word efficient simple and optimal algorithm.

(Refer Slide Time: 13:36)



Disadvantages of EDF

- Main disadvantages of EDF:

1/24/2010

181

So let see, the main disadvantages of E D F? One of the severe disadvantage is that it has poor transient overload handling we will elaborate this as we discuss and then its inefficient runtime inefficiency. And the most important or the most severe short coming is that, it cannot support resource sharing among tasks actually even if you try to support tasks that share resources, see that the performance becomes really unusable and the tasks miss their deadline. But, it must be possible that because the tasks are there is only one task executing at a time. So, in that case we must be able to share their tasks.

So let us here others answer, what do you mean by resource? And, why resource sharing is necessary among tasks? What is a resource? Some tasks require that previous output. Yes. Previous task output, yes. That is one way like let us say they are working on let us say some data items and one task is modifying that data item and another task also needs to use that data item.

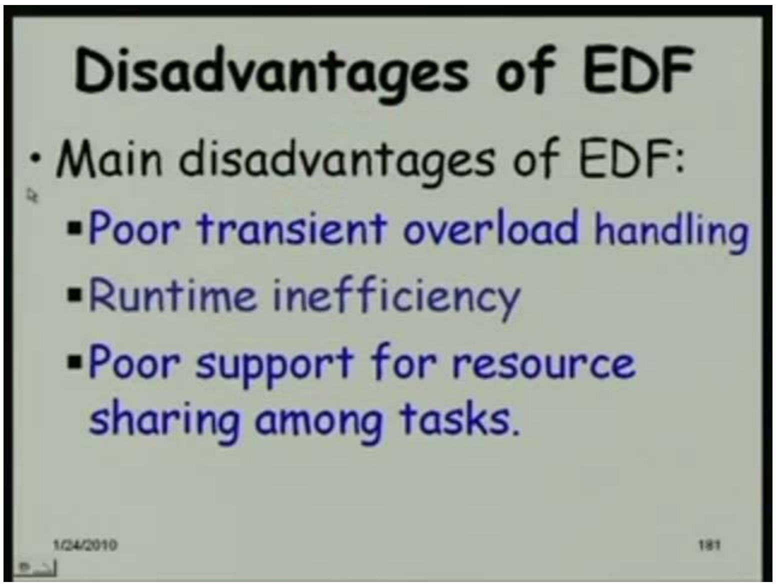
So, unless the first tasks complete the other one cannot use it listen it and if the other one uses the data will become inconsistent right. So, these are the critical resources actually.

So, one common example is certain data structures. Mainly the data. Data structures memory locations files devices etcetera can be critical resources right.

But (( )).

See, until one task completes its use of that device the other task cannot use it that becomes a critical resource of course, if you look at the traditional operating system printer even though it is a device it is not a critical resource, because of spilling that is a first level cause let us not time on that. Due to spilling printer is not a critical resource because a task just outputs whatever it wants to output gets stored in a file and meanwhile another task also can output the printer which just gets skewed. So, but there can be devices, which are critical resources? Until a device finishes reusing that resource another device cannot use it right. So let us elaborate these 3 sources.

(Refer Slide Time: 16:14)



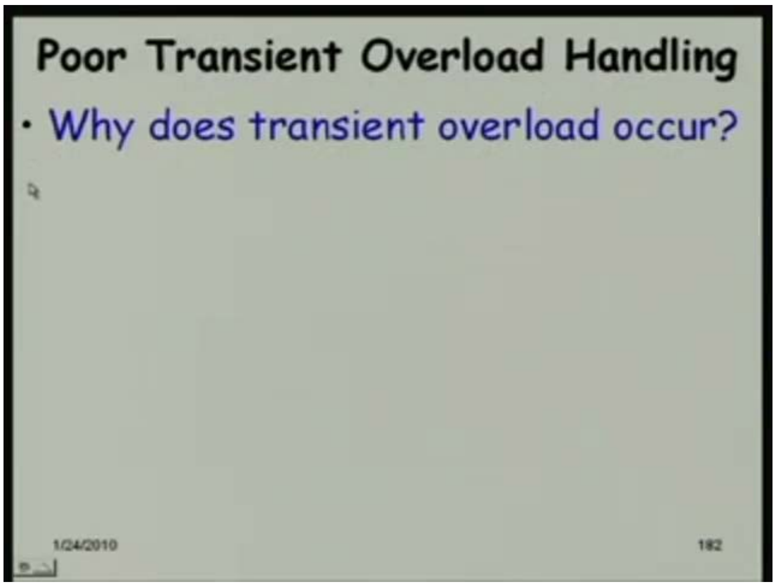
Disadvantages of EDF

- Main disadvantages of EDF:
  - Poor transient overload handling
  - Runtime inefficiency
  - Poor support for resource sharing among tasks.

1/24/2010

181

(Refer Slide Time: 16:16)


Poor Transient Overload Handling

- Why does transient overload occur?

1/24/2010 182

The first disadvantage we had said that poor transient overload handling, but what exactly is a transient overload and why does it occur? Anybody likes to answer this question. Because of infrequent or frequent arrival of task there could be switching between them. No, what is a transient overload? And can be overload on the system, overload on which system? On the scheduling algorithm. Not on the algorithm, the overload means the processor demand for task suddenly increases right. So, suddenly many tasks have risen all have their completion time requirement and the c p u is overloaded its, even if it is working hundred percent or it has to work really know hundred percent for them to meet their deadlines.

Earlier many times the c p u was idle now, the c p u is getting overloaded? we are talking of the c p u overload right.

Now, why should an overload occur? Can somebody answer this question, why should transient overload occur? transient means the overload occurs for a very short time let us say for one second it was working for one day nothing it was working fine, but for one second suddenly many tasks arrived and the system was overloaded. So, why should this transient overload occur under what situations?

Due to environmental conditions (( )),

Environmental conditions no not clear, I mean, what environment? can you just give some example.

Like suppose if you take a (( )),

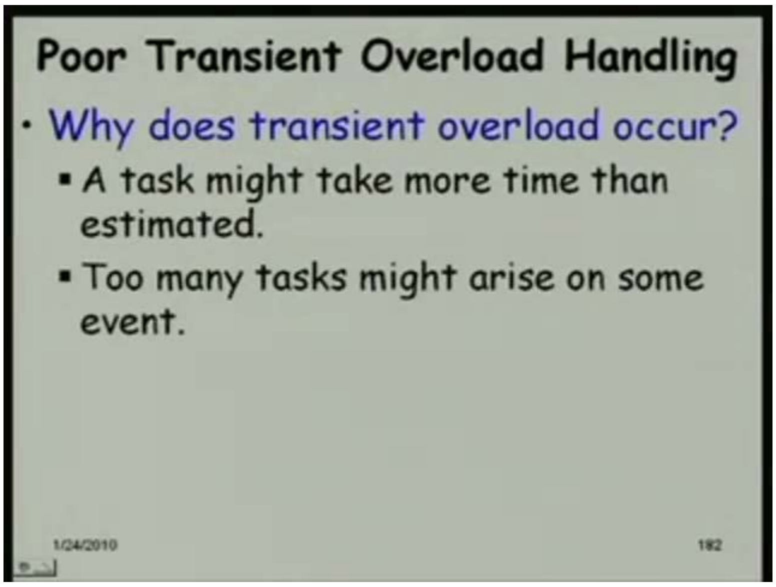
Yes.

Many systems are failing or giving unwanted output and accordingly there may be unwanted (( )),

Ok.

So there are one situation is that some events like he was saying that some events they cause many task to arise at the same time the simplest if example may be a fire condition was detected in the building and then you have to sound the alarms you have to start the water sprinkler we have to inform the security many tasks will arise at the same time. So, one reason for a transient overload is that some event has occurred which is triggering many tasks at the same time, but there may be other reason also.

(Refer Slide Time: 18:46)



Poor Transient Overload Handling

- Why does transient overload occur?
  - A task might take more time than estimated.
  - Too many tasks might arise on some event.

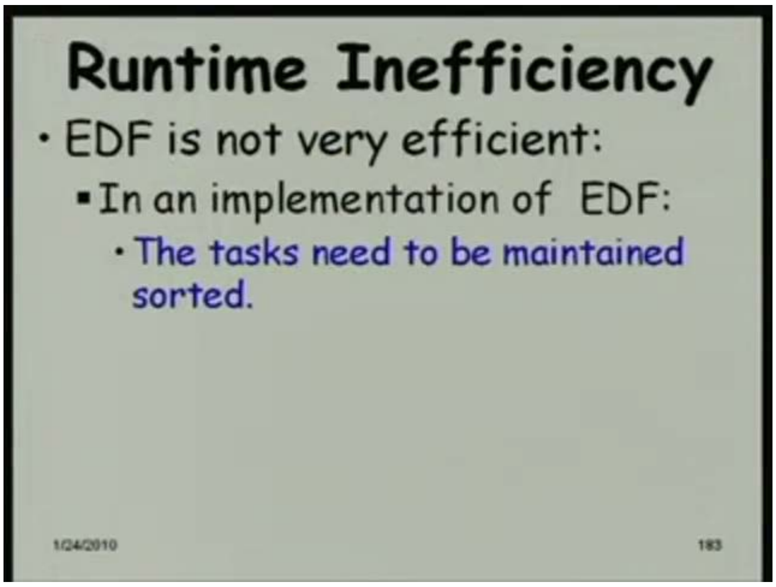
1/24/2010 182

So let us see that there are two main reasons, why transient overloads occur? one is too many tasks arise on some event like we are saying the other is that a task which was suppose to complete within sometime is taking more time. More than the expected time its taking maybe it has gone into a loop or it is gone into some inefficient algorithm is taking too much of time and the other tasks the c p u will get overloaded the other tasks are waiting to run. So, these are two situations where a transient overload might occur.

And when a transient overload occurs using the E D F algorithm, it becomes extremely difficult to predict which algorithm will oh sorry which task will miss its deadline. So, even the most critical task can miss its deadline whereas, something which is non critical like a logging task which logs the result might actually be running successfully.

There is another task which is supposed to be very crucial task might miss its deadline that will result in a system failure. So, using E D F a main problem one of the main problems is that when there is a overload situation we cannot really predict which task will miss its deadline because criticality etcetera is not taken into consideration see only based on the deadline. So that is one disadvantage

(Refer Slide Time: 20:22)



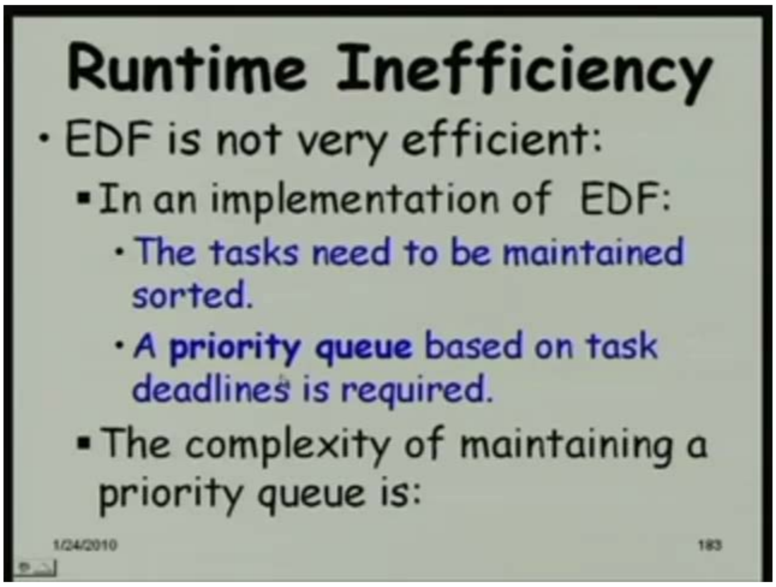
Runtime Inefficiency

- EDF is not very efficient:
  - In an implementation of EDF:
    - The tasks need to be maintained sorted.

1/24/2010 183

So, now let us see the other disadvantages, E D F is not very efficient in a typical implementation of e d f the tasks have to be maintained sorted according to the deadline. So, that we can tick the shortest deadline first, if you do not keep them sorted it will become still more inefficient listen it. So, here you maintain them sorted.

(Refer Slide Time: 20:55)



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 maintaining a priority queue is:

1/24/2010 183

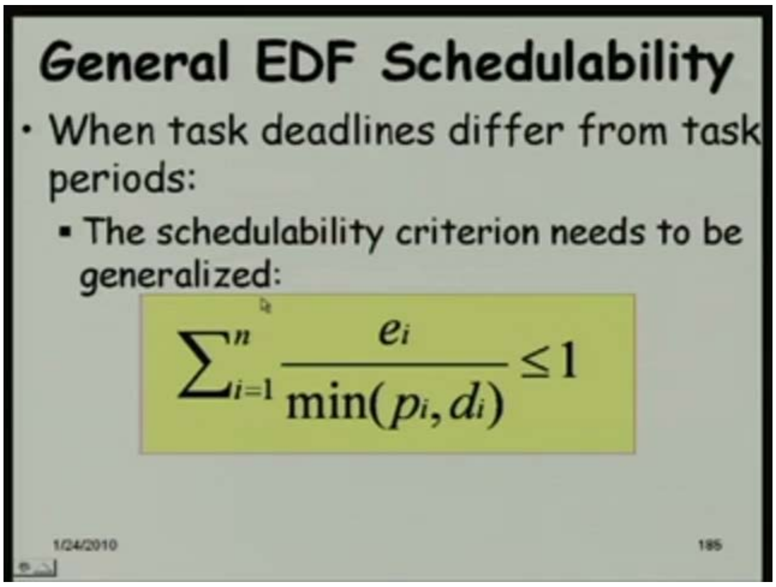
And one data structure which is can implements most efficiently a priority queue let me just ask this question, which is the most efficient data structure? Which can efficiently implement a priority queue?

National (( )),

Hip is the, if you revise your data structure course you will see that, hip is the one where a priority queue can be most efficiently maintained and using a priority queue the complexity of maintaining the set of tasks in the order of the deadline. I mean each time you are able to select the shortest deadline you will be logged to a log on a step. And that you need to do each time listen it, each time a task arrives and if the number of tasks waiting there is large it becomes very inefficient. Now, the third disadvantage as we are saying that this is the most severe disadvantage is poor resource handling poor resource handling.

So, if we in a practical situation the tasks need to share many resources for example, data structures and in operating system literature we will see these are called as critical sections. Later, we will discuss in more detail about that, how resource sharing is supported among real time tasks? And if you try to support resource sharing among tasks you see that becomes extremely inefficient. Very likely tasks will miss their deadlines and we will discuss this in more detail later, but possibly this the most severe disadvantages, because in real life situation tasks will need to share resources and using E D F it becomes very difficult.

(Refer Slide Time: 23:09)



General EDF Schedulability

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

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

1/24/2010 185

Now, what? If the deadline and the period are different see earlier you had said that the schedulability condition is e i by p i and assuming that p i was equal to d i. But we can have some examples of tasks, where the deadline is different from the period possibly, smaller than the period or larger than the period. So, we need to generalize the schedulability criterion. So, this is the criterion actually if d i is the deadline and p i is the period for the task t i and e i is the execution time requirement then for every task we will have to find out e i by minimum of p i by d i. If d i is ten and p i is fifty. So, deadline is 10 millisecond only after the task has arisen and the period is 50 millisecond will. So, consider 10 because by ten it should complete. Or in other words we are converting the task which had a 50 millisecond period to a 10 millisecond period e i by minimum of the p i d i it is as if that task had a 10 millisecond period. Yes please,

(( )) disadvantages (()) about the E D F,

We did not mention anything about the schedulability criteria that the task may not be schedule if we are using the E D F? Yes if the task are not meeting the e i by p i criteria.

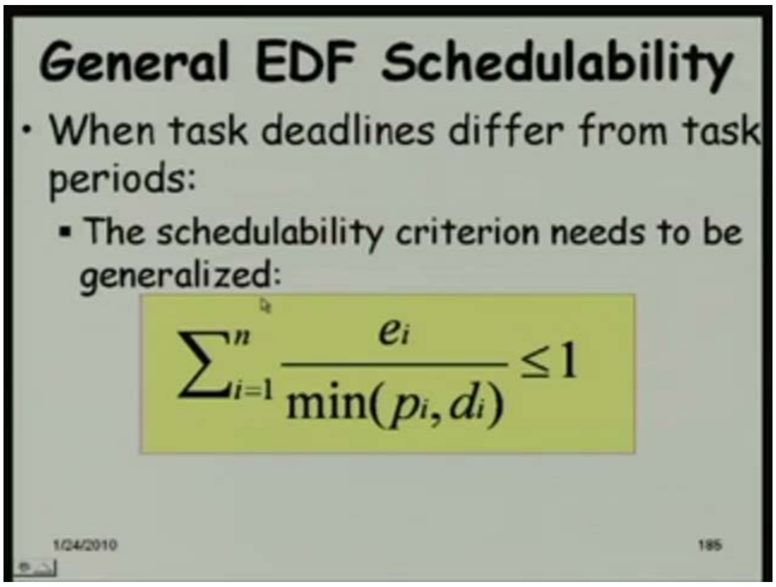
No.

See the question is that whether the constraint that you were saying e i by p i summation of that. So, this satisfies less than should be equal to less than one for a task set to be schedulable it is not (( )) only disadvantage for this algorithm actually none of the algorithm ever run a tasks set whose e i by p i is greater than one impossible.

(( )) similar to all that,

It is for all scheduler it is true because a processor cannot run we are assuming at the same time 2 tasks it cannot have you know utilization more than one we are considering uniprocessors running only single thread, single task at any time. So, let us proceed.

(Refer Slide Time: 25:42)



General EDF Schedulability

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

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

1/24/2010 185

So, you are saying that in some situations p i might be different from d i and then we need to consider minimum of that, but just see here one problem is that it is pessimistic. In the sense that if the task set which is having a deadline of 10 millisecond, but the period is let us say 50 millisecond it is not really occurring every 10 millisecond its occurring at after 50 millisecond listen it. But, here implicitly we are assuming that it is occurring every 10 millisecond convert it basically e i by minimum of p i by d i.

So, made it e i by d i right. So, you can say that it is likely conservative it may run if even if this criterion is satisfied. So, this is the sufficient condition. But not a necessary condition can you as a bonus mark construct an example where this condition is not satisfied, but the task set runs successfully using the E D F please try that. So, d i is less than p i d i is less than p i.

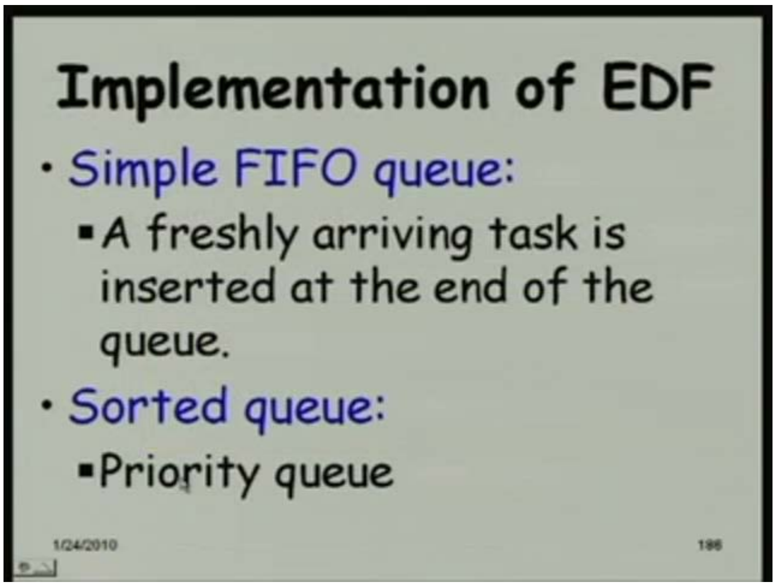
And this condition is not satisfied, but still the task set will successfully run please construct such an example. Now, the other situation is p i less than d i becomes a sufficient condition not a necessary condition similar thing again. So, please checks the characteristic of this algorithm where this is a sufficient or a necessary condition please proves it with i mean constructing examples.

Right please try that look at the two situations p i is less than d i and d i is less than p i please check this two conditions for both this conditions please see if you are able to construct examples which will.

(( ))

No, there are two condition one is the deadline is occurring after the period. So, that is this one we have written here deadline occurring after the period or p i is less than d i the other one is d i is less than p i. So, for both this conditions please check whether it is a sufficient condition, but not a necessary condition or it is both sufficient and necessary please check that that is a bonus problem please work it out and submit it to me.

(Refer Slide Time: 28:39)



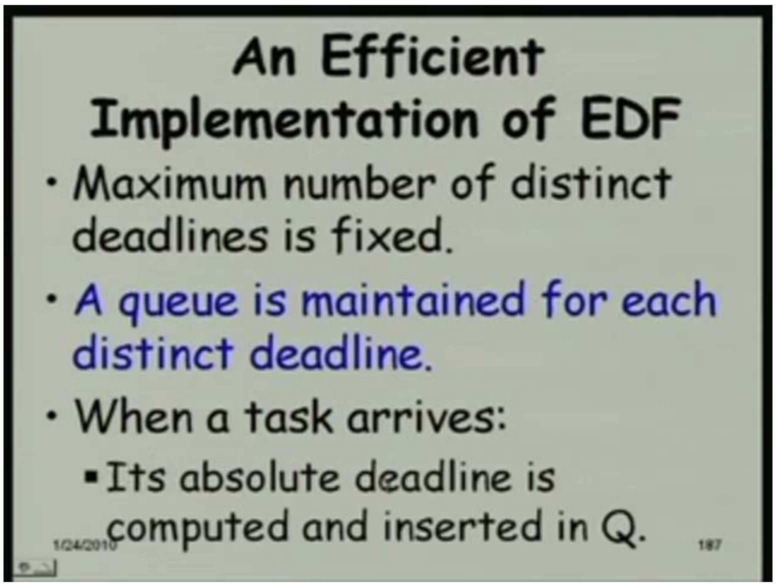
Implementation of EDF

- Simple FIFO queue:
  - A freshly arriving task is inserted at the end of the queue.
- Sorted queue:
  - Priority queue

1/24/2010 186

How does one implement E D F? Our saying that, you would be able to write schedulers and you can plug into the operating system it will use your own custom scheduler. But, if you want to use E D F? What will you have to do we had said that a simple FIFO queue become very inefficient where you insert it at the end of the queue. So, the inserting is trivial listen it one time just insert it at the end of the queue, but each time you want to select a task you will have to scan the entire queue (( )) each time all end tasks you will have to see and you would not want to do that like to use a sorted queue.

(Refer Slide Time: 29:28)



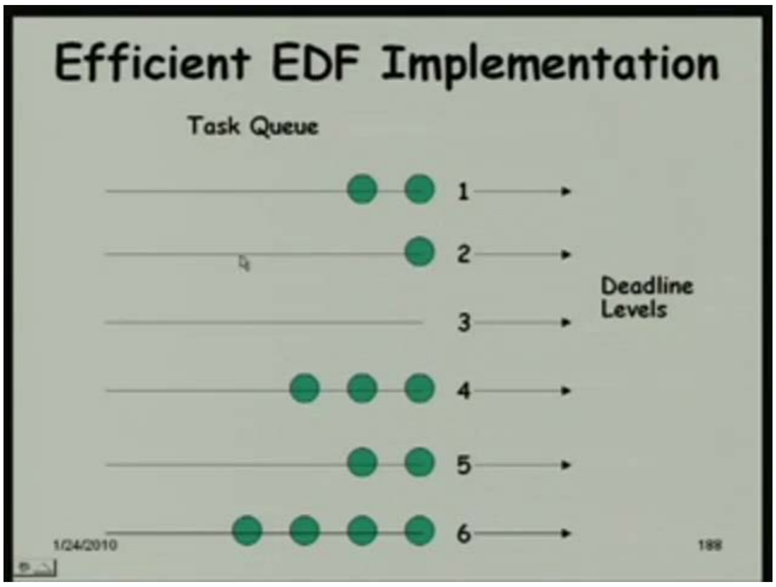
An Efficient Implementation of EDF

- Maximum number of distinct deadlines is fixed.
- A queue is maintained for each distinct deadline.
- When a task arrives:
  - Its absolute deadline is computed and inserted in Q.

1/24/2010 187

But we had said that sorted queue can be implemented using a priority queue etcetera, but can a more efficient implementation of E D F we take. Yes, the implementations of E D F those who would like to use it possibly can be like this that you consider several queues each one holds the tasks within certain deadline for example, 10 to 20 millisecond one queue let us say 21 to 30 millisecond another deadline and so on. So, a queue is maintained for each distinct deadline and a when a task is it arrives it is put into the appropriate queue inserted in the appropriate queue.

(Refer Slide Time: 30:22)

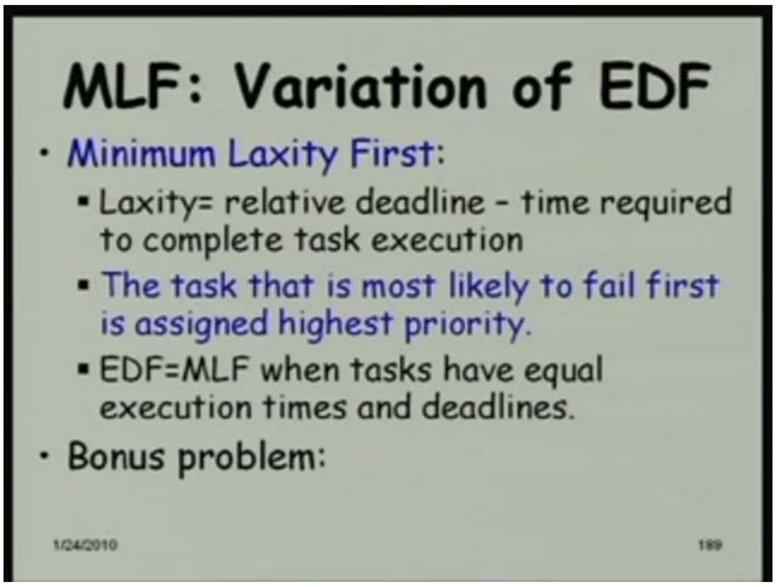

Efficient EDF Implementation

So, these are the deadline levels. So, each time and within it is a key of course, you will have to keep them sorted. Right, and each time also you would have to pick the task you just pick from the top queue here this has a shortest deadline right. Otherwise if this queue is empty check here and so on. So, please do an analysis of this efficient E D F implementation again another bonus mark.

Please check out, what is the complexity? Of this compared to a priority queue simply priority queue where you have a set of distinct deadline levels. The time and space complexity please compare and check whether it is an efficient implementation and what kind of benefit shall we get by doing this kind of an implementation of E D F. So, that is a bonus mark as somebody submits a good answer i will discuss that here please try to submit. There is no time limit for when you can submit, but please try to submit as soon as you can think of it search all the literature that is available think about it consult anybody whatever does not matter.

But only thing is that the one who has submitted and appears to have done the work he will be awarded the mark for the same for answer two will not get the mark. Because, this not a assignment it is a bonus problem you need not submit it, but those who submit they i will just check what is the thinking they have done what kind of effort they have put searching the literature etcetera right based on that will award marks.

(Refer Slide Time: 32:17)



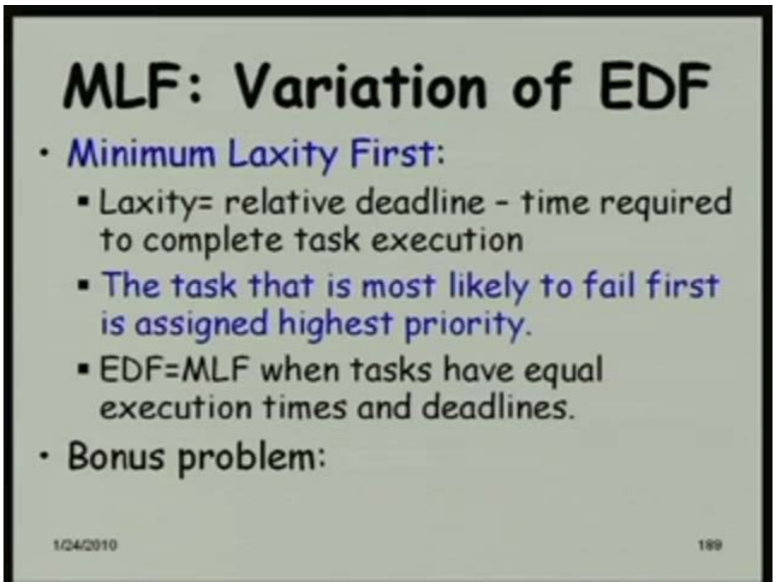
MLF: Variation of EDF

- 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.
- Bonus problem:

1/24/2010 189

Now, the E D F that we discussed there are several variations of that as i was saying that this the basic algorithm and a large number of them are proposed possibly on input and variation is the m l f or the minimum laxity first, see in the earliest deadline first at any instant of time we are just checking which task. Has the shortest deadline, but in the minimum laxity first algorithm we check what is the amount of work remaining for a task and how much is the deadline. So if a task deadline is let us 10 millisecond.

(Refer Slide Time: 32:59)



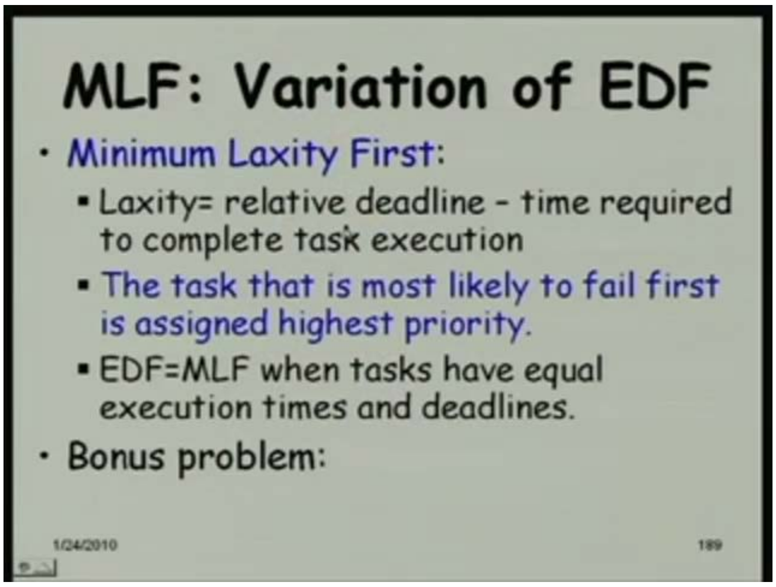
MLF: Variation of EDF

- 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.
- Bonus problem:

1/24/2010 189

Right and another task and the 10 millisecond task have completed 50 percent of its work. So, let us say another only 2 millisecond is remaining and it has 10 millisecond is the deadline. 2 milliseconds work remaining and 10 millisecond is the deadline and another task which has 12 millisecond is deadline, but 6 millisecond of work remaining. So, that algorithm will be taken up even though its deadline is later, but it has more work remaining. So, that one will be taken up this the minimum laxity first.

(Refer Slide Time: 33:42)



MLF: 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.
- Bonus problem:

1/24/2010 189

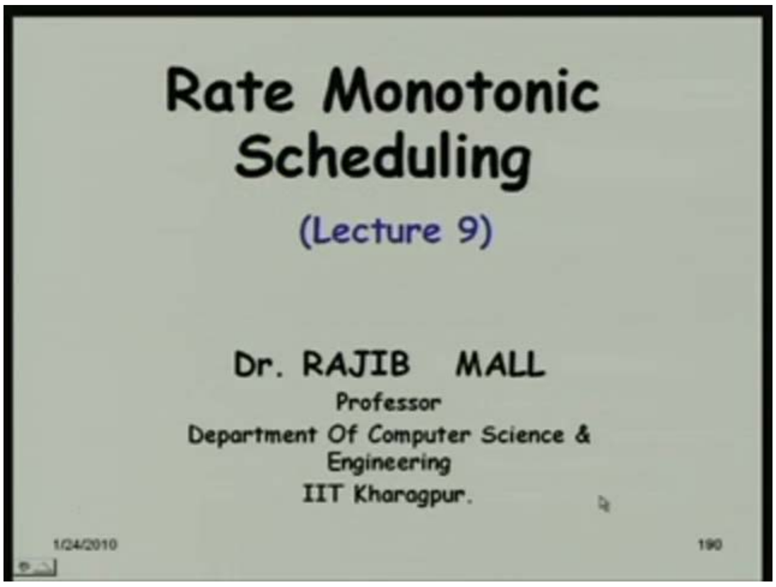
Right the laxity as written here is the deadline minus the time required to complete the task execution. So, how much it has completed sum? So far and how much more computation it needs. So, the deadline was 10, but it had already completed two out of the total requirement of four. So, only two was required for this. So, 8 is the laxity for that for the other task that we discussed the second task twelve was the relative deadline from that point. But it had not completed any work and it required 6 millisecond to complete. So, its laxity is 6, where as the other one having deadlined ten the laxity was eight.

So, the idea behind this algorithm a variation of E D F is that the task that is most likely to fail first is assigned highest priority. So, it is the task if it seems it has the least time lax time to complete it is the one under overload or whatever situation that is the one which is likely to miss its deadline for example, another higher priority task arise right.

But of course, it is obvious that when the execution time and deadline is the same all the tasks are equal execution time is a deadline.

Then E D F is the same as minimum laxity first problem. So, the bonus problem here is that analyze the performance of minimum laxity first and E D F. So, three bonus problems are already announced today. So, these are good opportunities to get extra mark from teachers' assessment. So, analyze the performance of m l f with respect to E D F check which one is does more efficient which one can schedule a set up task not schedulable by others. These two points please compare them. So, now, let us look at the rate monotonic scheduler.

(Refer Slide Time: 35:56)



Rate Monotonic Scheduling

(Lecture 9)

Dr. RAJIB MALL

Professor

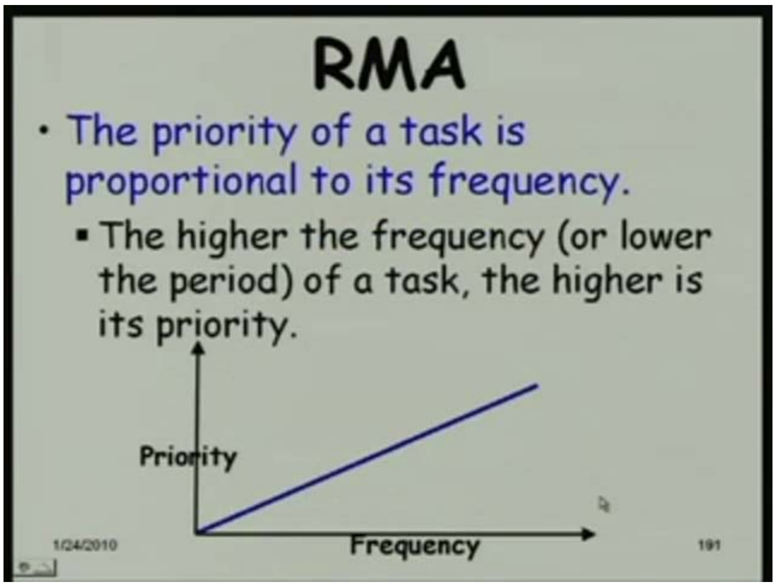
Department Of Computer Science & Engineering

IIT Kharagpur.

1/24/2010

190

(Refer Slide Time: 35:58)



RMA

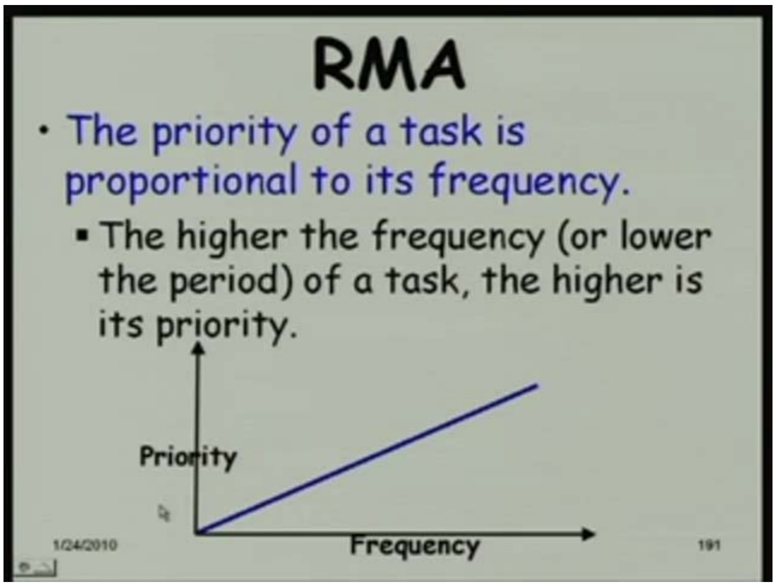
- The priority of a task is proportional to its frequency.
  - The higher the frequency (or lower the period) of a task, the higher is its priority.

Priority
Frequency
1/24/2010
191

So, this was actually the focus of this hour’s lecture, but while reviewing the E D F that we had discussed last time, we actually you had lot of questions and we spent quite a bit of time there now let us look at the rate monotonic algorithm this is the one which is most widely used in all real time applications. So, here the algorithm is again simple the programmer needs to assign priority here see earlier we had no notion of a priority in E D F there was no notion of a priority the scheduler just looked at the deadlines. We the only thing that we possibly needed to do for the scheduler was to declare the deadlines and the execution times right, but here.

The programmer has to assign the prioritative tasks. So, how does the programmer assign prioritative tasks? It will check, what is the frequency? With which the task occurs or in other words what is the period. If a task has lets a period of 100 millisecond and another one has period of 10 millisecond then the frequency of the 10 millisecond task is high isn’t it by the time one with 100 millisecond occurs this 10 millisecond task could have occurred ten times right. So, the 10 millisecond task could be given higher priority by the programmer if he wants to run it under rate monotonic algorithm. So, this is the plot actually. To be done (( ))

(Refer Slide Time: 37:38)



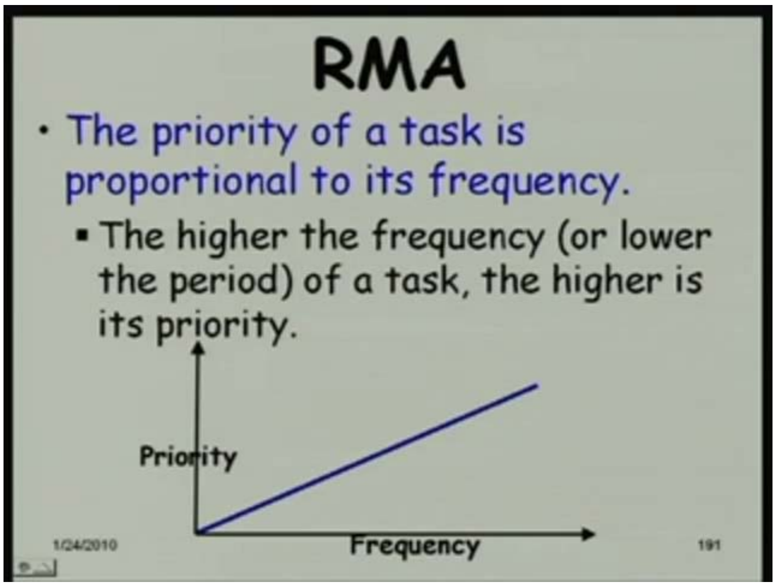
RMA

- 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
1/24/2010
191

Before the program runs actually, the programmer should have decided by looking at the task characteristics what is the priority of which task done statically before the program starts done by the (( )) is that ok. Now, let us proceed.

(Refer Slide Time: 38:00)



RMA

- 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
1/24/2010
191

So, the priority value increases monotonically with the frequency or the rate is the frequency.

(( ))

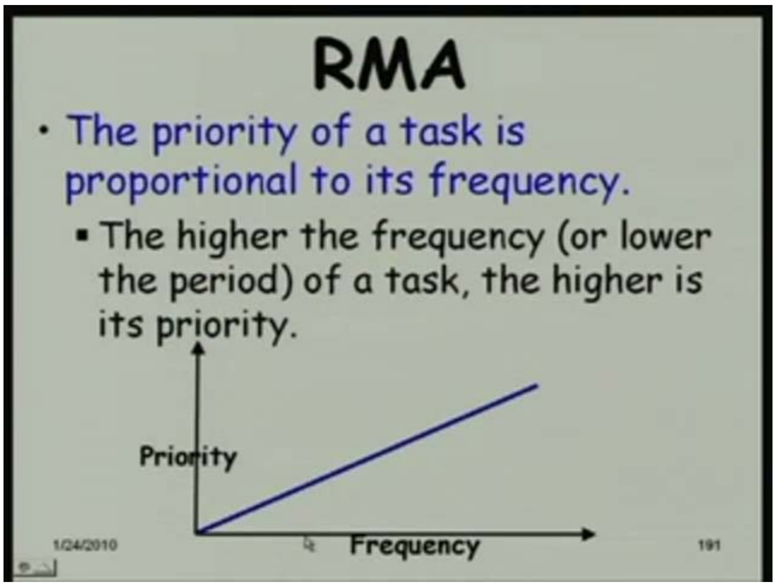
Before hand yes of course,

And priority is also give a (( )),

Yes.

Why there are (( )) means we have two variables and they are dependent on each other then we should decide. No the priority is dependent on the rate.

(Refer Slide Time: 38:30)



RMA

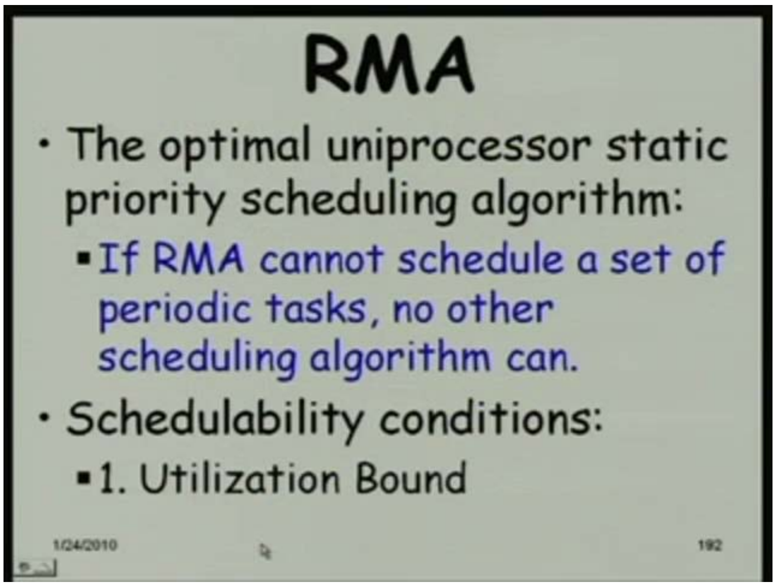
- The priority of a task is proportional to its frequency.
  - The higher the frequency (or lower the period) of a task, the higher is its priority.

Priority
Frequency
1/24/2010
191

The rate is basic characteristic of a task, priority is not the characteristic of a task the priority is a dependent variable on the rate. So, the programmer has to decide either the priority or the rate. No, programmer cannot decide the rate see for example, let us say a temperature needs to be sampled every hundred millisecond for a let us say an air conditioner to work right.

So, that rate is fixed by the task characteristic, what the programmer can do? Is look at hundred millisecond and check if another task needs two hundred millisecond every two hundred millisecond needs to be sampled. So, then the hundred millisecond task will get a higher priority or the programmer has to assign that as a higher priority. So, the rate is the independent variable here the rate or the frequency and the priority depends on the rate and this is fixed by the programmer monotonically the priority should increase with the rate that is the algorithm their priority should monotonically increase with the rate is that.

(Refer Slide Time: 39:40)



RMA

- The optimal uniprocessor static priority scheduling algorithm:
  - If RMA cannot schedule a set of periodic tasks, no other scheduling algorithm can.
- Schedulability conditions:
  - 1. Utilization Bound

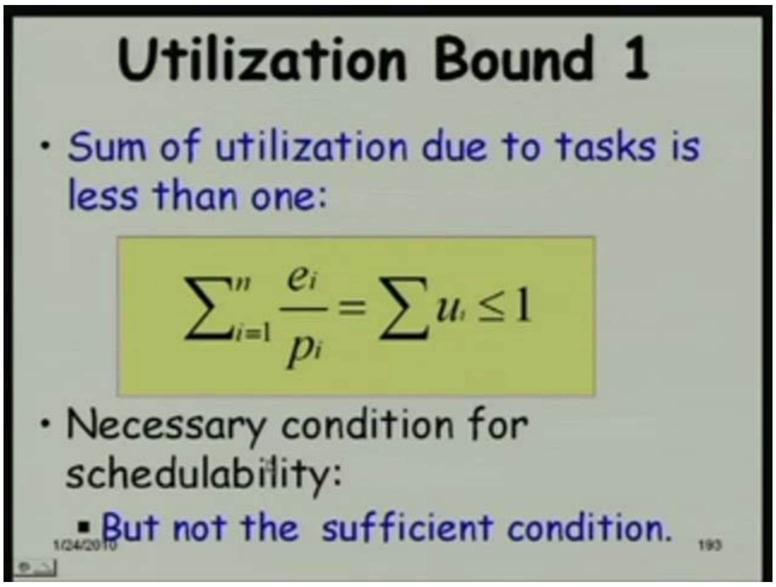
1/24/2010 192

Simple algorithm and it is also the optimal uniprocessor static priority algorithm see in the E D F, we said that it was a dynamic priority algorithm because the same task at different points of time it will get different priorities. Or it after it waits for sometime it just gets precedence over other tasks and so on. But here, once you fix that priority as long as there is a higher priority task even if it is waiting for long time it will not run, but it is an optimal static priority algorithm.

What it really means is that as long as you assign priorities statically if R M A cannot schedule then no other static priority algorithm can schedule. But of course, a set up task which r m a cannot schedule E D F can possibly schedule because E D F is optimal algorithm including all dynamic and static priority algorithms, but R M A is the static priority algorithm it is optimal static priority algorithm something not schedulable by R M A can be schedulable under E D F.

So, how do you check the schedulability? One is of course, the utilization bound because the processor cannot run two tasks at the same time running only on a uniprocessor.

(Refer Slide Time: 41:25)



Utilization Bound 1

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

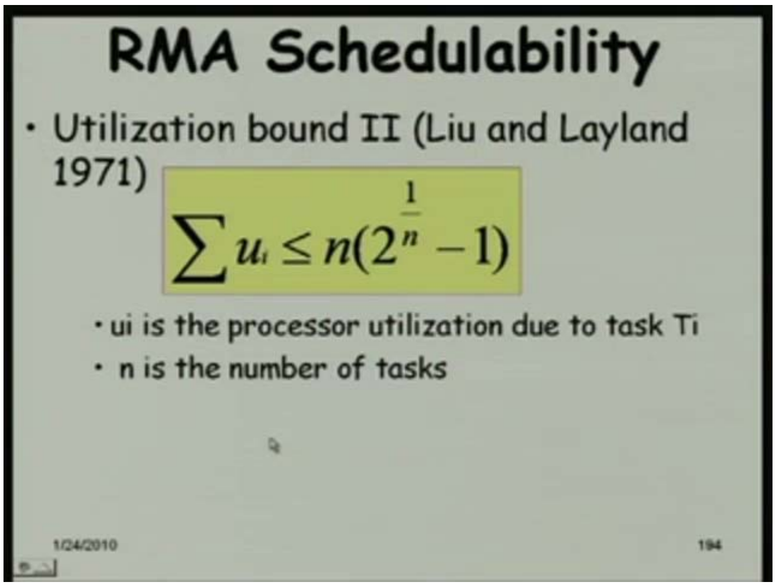
$$\sum_{i=1}^{n} \frac{e_i}{p_i} = \sum u_i \leq 1$$

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

1/24/2010 193

So, this is the same thing we had for E D F the same restriction holds here, but we will have additional restrictions. So, the necessary condition is that the utilization due to tasks is less than one, but even if this is satisfied a task may not run successfully under R M A it is just a necessary condition it is not a sufficient condition.

(Refer Slide Time: 41:51)



RMA Schedulability

- Utilization bound II (Liu and Layland 1971)

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

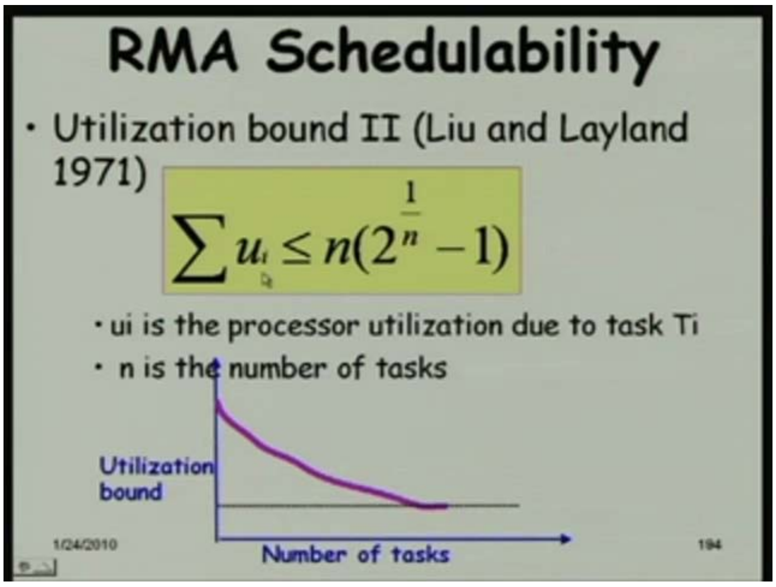
- ui is the processor utilization due to task Ti
- n is the number of tasks

1/24/2010 194

So, the sufficient condition was given the liu and layland nineteen seventy one as i was saying that most of the research in these areas were done in nineteen seventies sixties. So, this was one of the very prominent result i think any real time scheduler or any real time book we will have a on the liu layand algorithm very basic algorithm. We are just calling it as the utilization bound two, if u i is the utilization due to the task that is sigma e i by p i. So, to be less than n is the number of tasks and into two to the power one by n minus one.

So, if this condition is satisfied then the task set will definitely be schedulable. So, this is the sufficient condition, the liu layland condition is a sufficient condition. You can take a set of tasks and then check whether this is satisfied, n into two to the power one by n minus one as long as it is greater than the utilization of the set of tasks then the tasks set will be schedulable.

(Refer Slide Time: 43:14)



RMA Schedulability

- Utilization bound II (Liu and Layland 1971)

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

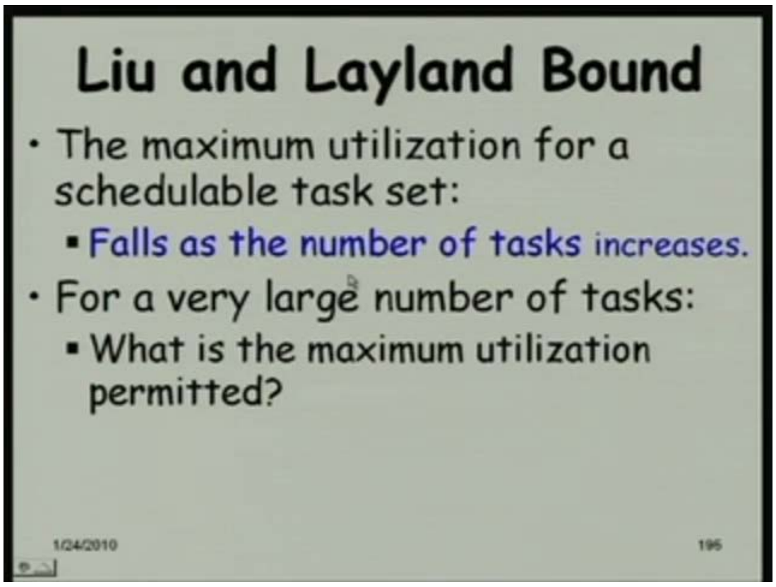
- ui is the processor utilization due to task Ti
- n is the number of tasks

Utilization bound
Number of tasks
1/24/2010
194

Now, if we plot this utilization bound see this is the utilization bound, anything having utilization more than this will not run under e d f sorry under R M A. So, if you plot this value with respect to n you will get a plot like this. When n is equal to one see, here n is equal to one see the horizon here is one actually because one is the minimum number of tasks we will consider. So, how much is it coming? When n is one what is the maximum utilization? One. So, one into two to the power one by one minus one. So, this is one into two minus one is equal to one right. So, 100 percent utilization can be achieved when just one task runs.

But what about when two tasks run. So, two into two to the power one by two minus one will be two into 1.414 minus 1.414 into two is 0.82. So far, one task it is 100 percent for two tasks it will be 81 percent three tasks will be less and then it comes here and finally, saturates when the number of tasks is large, what about when n is infinity?

(Refer Slide Time: 44:47)

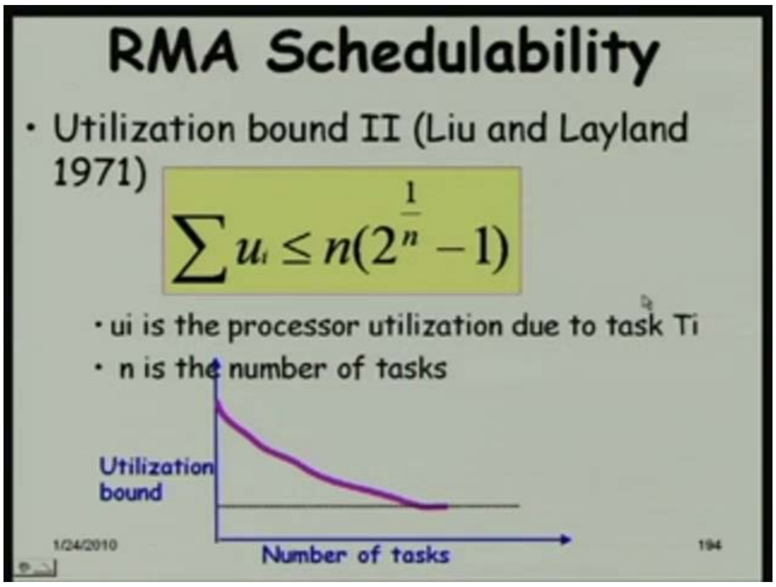


Liu and Layland Bound

- 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?

1/24/2010 195

(Refer Slide Time: 44:54)



RMA Schedulability

- Utilization bound II (Liu and Layland 1971)

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

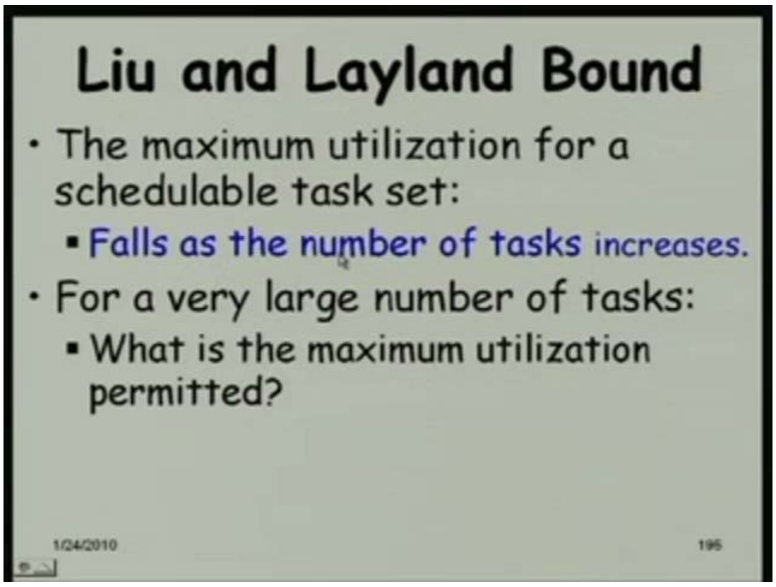
- ui is the processor utilization due to task Ti
- n is the number of tasks

Utilization bound
Number of tasks

1/24/2010 194

S o when n is infinity; it will be infinity into two to the power one by infinity minus one and that is a indeterminate form listen it. Because, two to the power one by infinity is one two to the power zero is one. So, one minus one is zero and infinity into zero form

(Refer Slide Time: 45:16)



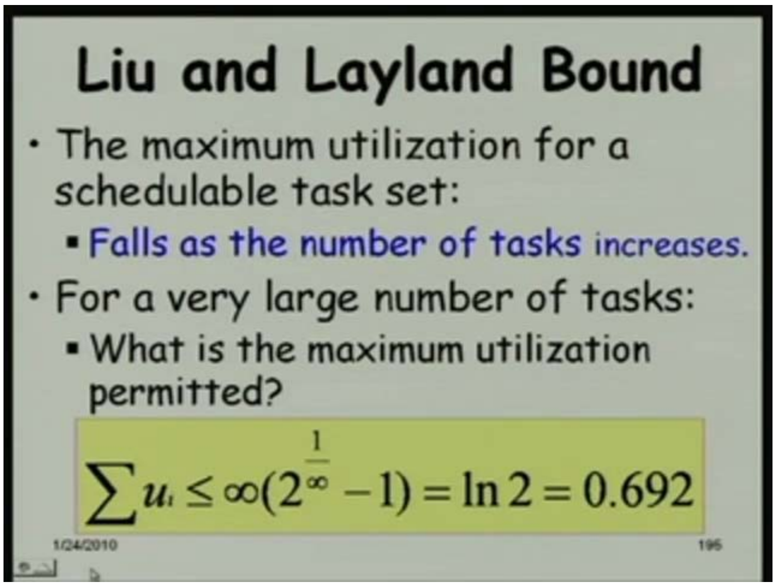
Liu and Layland Bound

- 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?

1/24/2010 195

let us look at that, the maximum utilization for a schedulable task set falls as the number of tasks increases for one task we can go up to hundred percent utilization of the processor and still the deadline will be met. But, when we have two tasks we can go up to 81 percent for three tasks we can go up to some 78 percent. Four tasks 75 percent and so on, but for a very large number of tasks.

(Refer Slide Time: 45:45)



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?

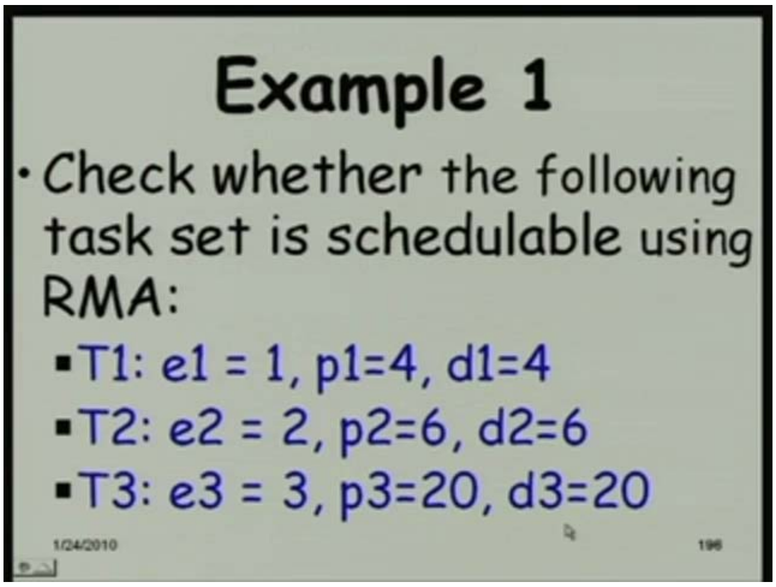
$$\sum u_i \leq \infty(2^{\frac{1}{\infty}} - 1) = \ln 2 = 0.692$$

1/24/2010 195

We have infinity into two to the power one infinity minus one which is which is an indeterminate form. Now, to find the value of a indeterminate form we will have to use some l'hospital's rule and so on from basic mathematics and you will find that it is log two to base e L n 2 which is 0.692. So, if the number of tasks is reasonably large ten twenty or something like that you can think of that if the utilization is less than 0.7 or. So, it will run.

But, if utilization achieved for the processor is more than 70 percent the task set will not run and 70 percent is not a large value. The processor is remaining idle 30 percent of the time listen it. So, this is the bound the lieu lay land bound is that as long as the number of task set is large and the utilization is less than 70 percent it will run let us take an example. Check whether the fallowing task set is schedulable using R M A. So, please work it out.

(Refer Slide Time: 46:55)

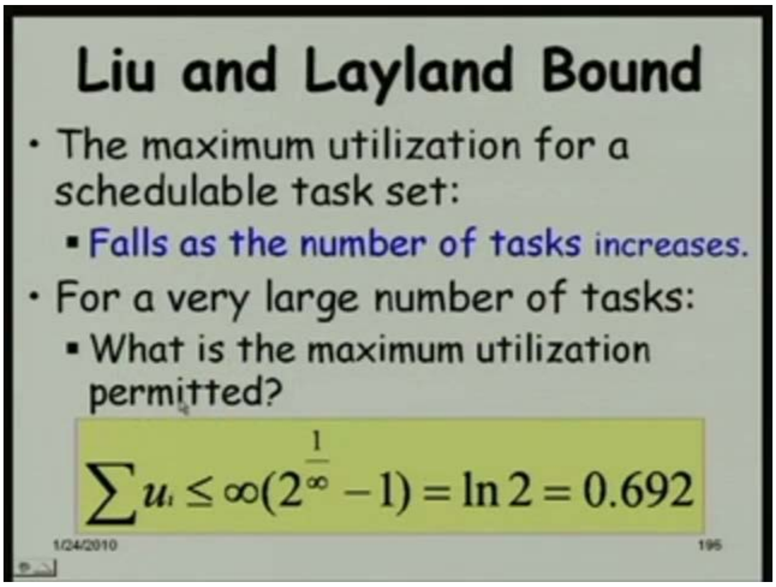


Example 1

- Check whether the following task set is schedulable using RMA:
  - T1: e1 = 1, p1=4, d1=4
  - T2: e2 = 2, p2=6, d2=6
  - T3: e3 = 3, p3=20, d3=20

1/24/2010 196

(Refer Slide Time: 47:07)



Liu and Layland Bound

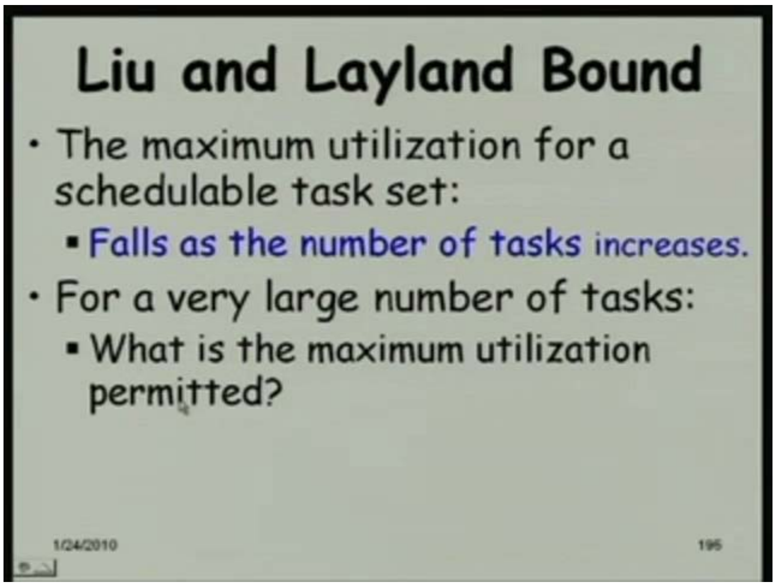
- The maximum utilization for a schedulable task set:
  - Falls as the number of tasks increases.
- For a very large number of tasks:
  - What is the maximum utilization permitted?

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

1/24/2010 196

This condition for schedulability.

(Refer Slide Time: 47:10)

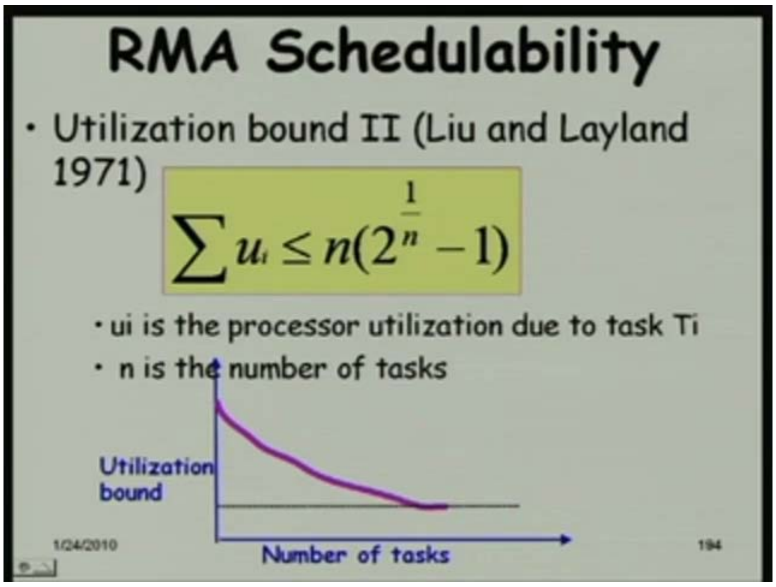


Liu and Layland Bound

- 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?

1/24/2010 196

(Refer Slide Time: 47:12)



RMA Schedulability

- Utilization bound II (Liu and Layland 1971)

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

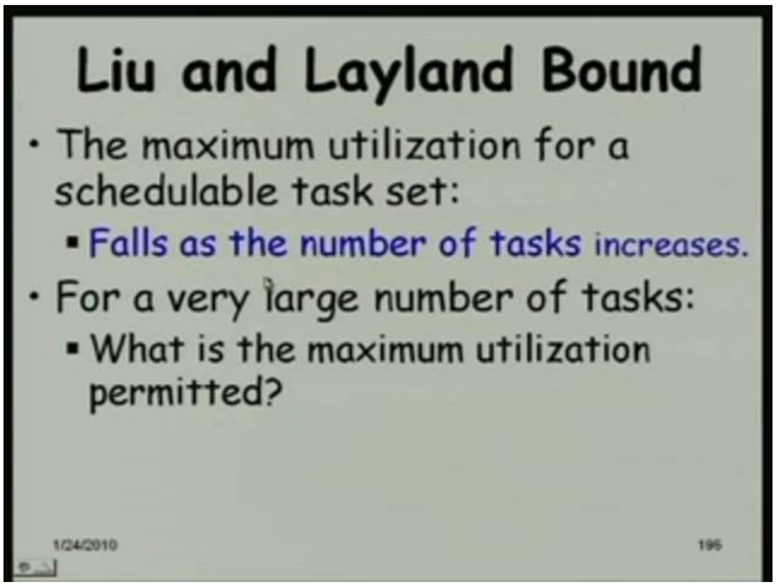
  - ui is the processor utilization due to task Ti
  - n is the number of tasks

Utilization bound
Number of tasks

1/24/2010 194

Have forgotten the expression look at it. N into two to the power one by n minus one as long as that is greater than the utilization achieved the tasks set should be schedulable.

(Refer Slide Time: 47:24)

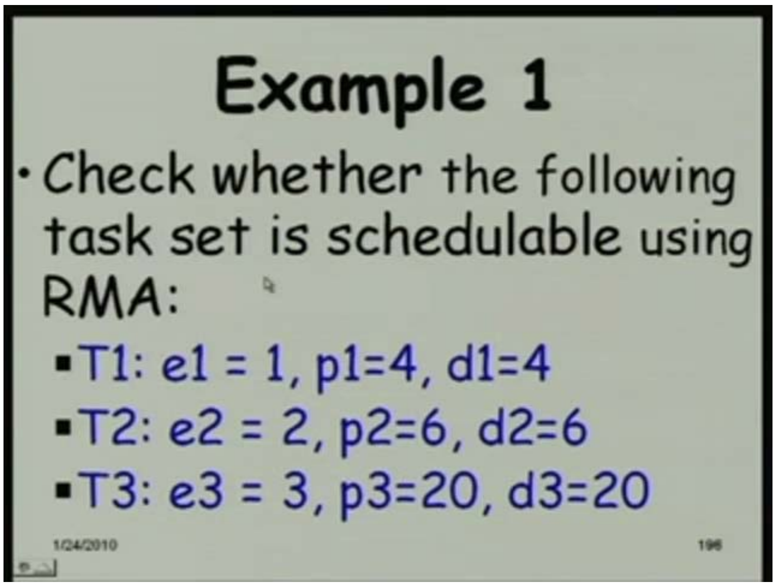


Liu and Layland Bound

- 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?

1/24/2010 195

(Refer Slide Time: 47:26)



Example 1

- Check whether the following task set is schedulable using RMA:
  - T1: e1 = 1, p1=4, d1=4
  - T2: e2 = 2, p2=6, d2=6
  - T3: e3 = 3, p3=20, d3=20

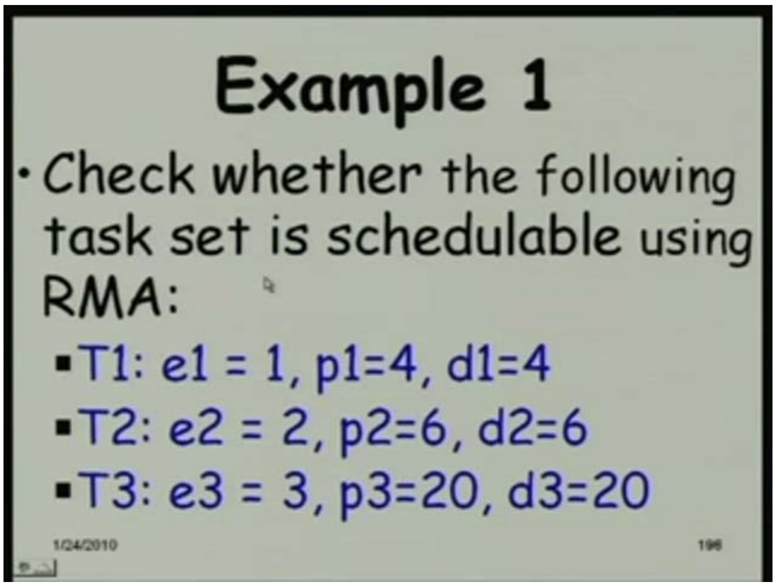
1/24/2010 196

So, the example there are three tasks first tasks the first task execution time is 1 millisecond and the period is 4 millisecond second task execution time 2 millisecond period 2 millisecond third one. Execution time 3 millisecond period 20 millisecond. So, first tell what about the priority which one will have highest priority under r m a t one will have the highest priority because its deadline.

(( )),

Yes, its period and deadline which are the shortest the second one is second highest third one is the third highest. Now, tell me whether this task set will run under E D F please work out.

(Refer Slide Time: 48:34)



Example 1

- Check whether the following task set is schedulable using RMA:
  - T1: e1 = 1, p1=4, d1=4
  - T2: e2 = 2, p2=6, d2=6
  - T3: e3 = 3, p3=20, d3=20

1/24/2010 196

So, we will have to first find the utilization and check whether it is less than the utilization bound. So, what is the utilization you are getting?

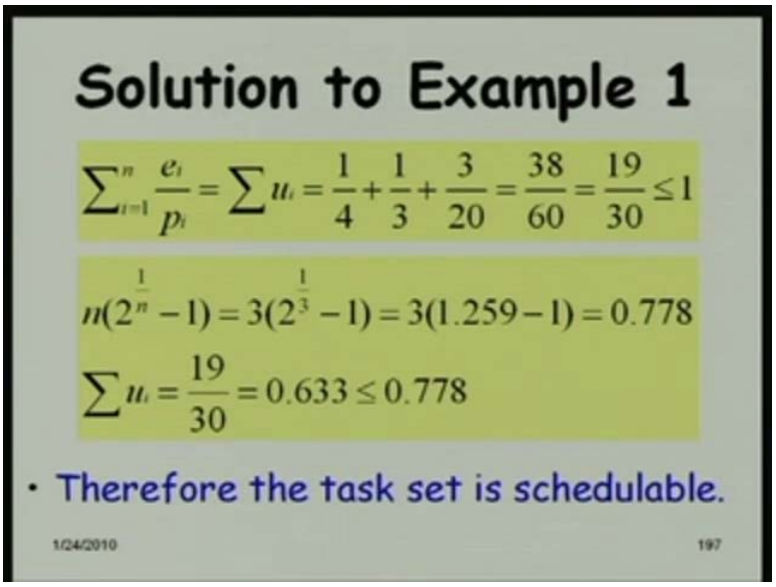
(( )) eleven by fifteen.

Eleven by fifty three is it.

Fifteen,

Eleven by fifteen is everybody getting that eleven by 15.735. So, is eleven by 15.73, yes it is. So, eleven by fifteen and what about the utilization bound? three tasks are there. So, n into two to the power one by n minus one

(Refer Slide Time: 49:32)



Solution to Example 1

$$\sum_{i=1}^{n} \frac{e_i}{p_i} = \sum u_i = \frac{1}{4} + \frac{1}{3} + \frac{3}{20} = \frac{38}{60} = \frac{19}{30} \le 1$$

$$n(2^{\frac{1}{n}} - 1) = 3(2^{\frac{1}{3}} - 1) = 3(1.259 - 1) = 0.778$$

$$\sum u_i = \frac{19}{30} = 0.633 \le 0.778$$

- Therefore the task set is schedulable.

1/24/2010 197

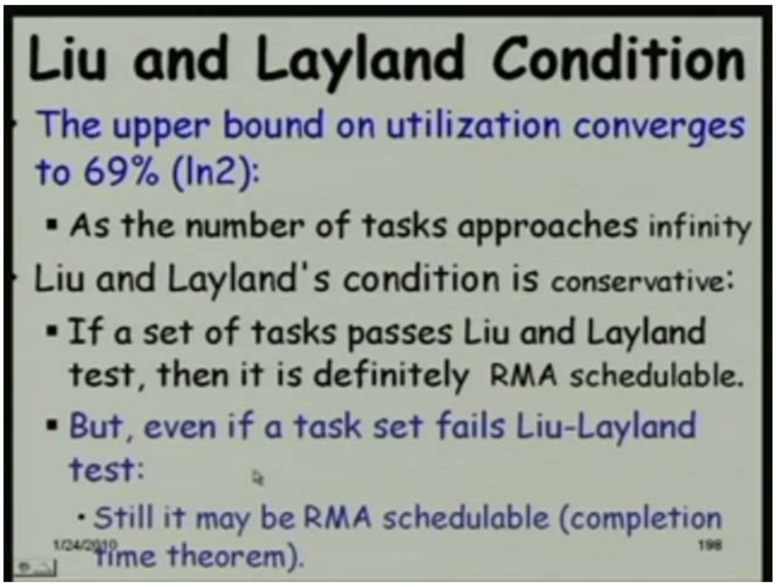
See here there is a solution see one by four plus one by three listen it.

(( )),

One by three,

One by four plus one by three plus three by twenty is thirty eight by sixty is that fifteen plus twenty is thirty five plus thirty eight. Now, the first condition is satisfied eleven by fifteen is less than one now the second utilization bound we have to check n to the power n into two to the power one by n minus one is if you simplify this you will get 0.778. And 0.778 is less than 0.73 listen it, 11 by 15 is 0.73. So, 0.73 actually there is a mistake it should have been 11 by 15. So, 0.73 is less than 0.778. So, the task set is schedulable.

(Refer Slide Time: 51:05)



Liu and Layland Condition

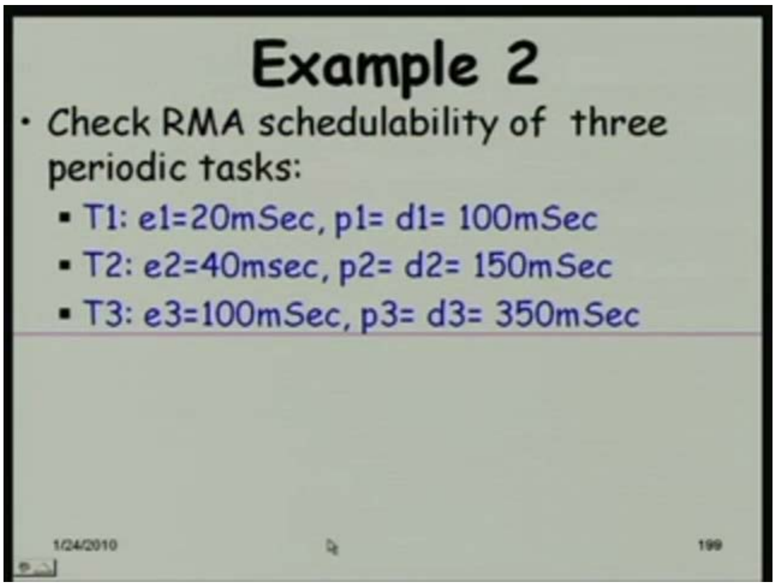
- The upper bound on utilization converges to 69% (ln2):
  - As the number of tasks approaches infinity
- Liu and Layland's condition is conservative:
  - If a set of tasks passes Liu and Layland test, then it is definitely RMA schedulable.
  - But, even if a task set fails Liu-Layland test:
    - Still it may be RMA schedulable (completion time theorem).

1/24/2010 198

Now, as we are saying that the utilization bound 69 percent for large number of tasks is actually does not look a good utilization for a processor. It was proved later in end of 1980s or. So, mid of eighties that liu and layland's condition is actually conservative. If it a set of task passes the liu layland condition then definitely it will be schedulable, but if it fails it, but still there is chance that it will be schedulable.

So, liu and layland is a sufficient condition, but not a necessary condition even if it fails liu layland still you can try and the completion time theorem was proposed to check whether the said task set is schedulable. Even if it fails the liu layand condition you should check for the completion time theorem before saying that whether the task set is schedulable or not.

(Refer Slide Time: 52:14)



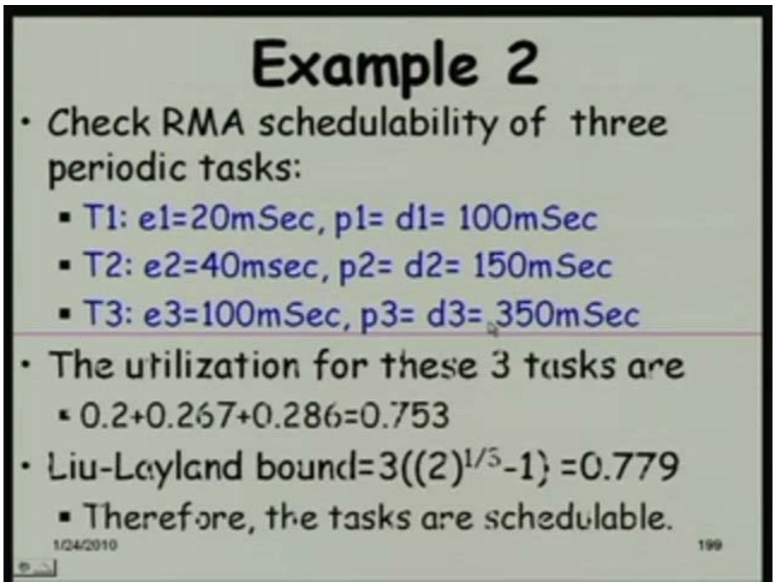
Example 2

- Check RMA schedulability of three periodic tasks:
  - T1: e1=20mSec, p1= d1= 100mSec
  - T2: e2=40msec, p2= d2= 150mSec
  - T3: e3=100mSec, p3= d3= 350mSec

1/24/2010 199

Now, just to take an example let us look at these three tasks, task t one twenty and hundred task t two is forty by hundred fifty and task three is hundred millisecond is the run time requirement execution time requirement and three hundred millisecond is the period. So, if you try it out

(Refer Slide Time: 52:43)



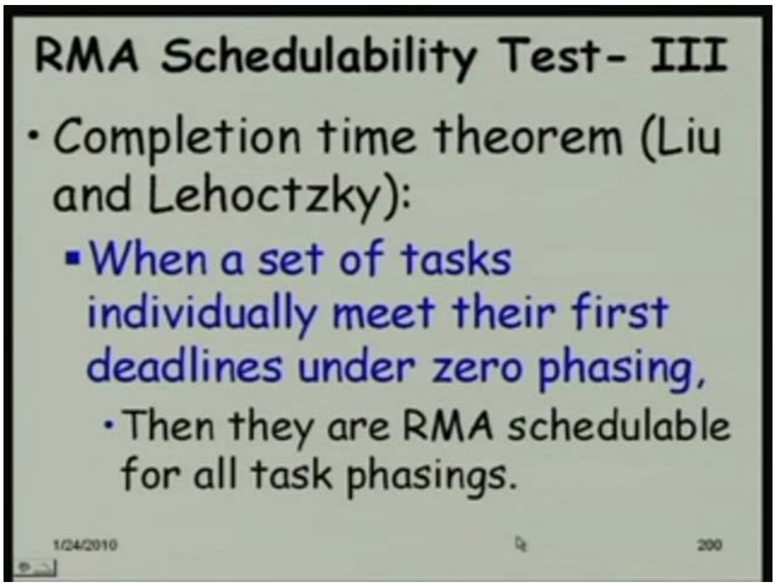
Example 2

- Check RMA schedulability of three periodic tasks:
  - T1: e1=20mSec, p1= d1= 100mSec
  - T2: e2=40msec, p2= d2= 150mSec
  - T3: e3=100mSec, p3= d3= 350mSec
- The utilization for these 3 tasks are
  - 0.2+0.267+0.286=0.753
- Liu-Layland bound=$3((2)^{1/3}-1)$ =0.779
  - Therefore, the tasks are schedulable.

1/24/2010 199

You will say that the utilization of the three tasks, the first one is 20 by 100 is 0.2. Second one is 40 by 150 is 0.267 and the third one is 350 is 0.286. Is it if you sum it up it will be 0.753 and three tasks the liu layland bound or the utilization bound is 0.779. So, this task set is also schedulable.

(Refer Slide Time: 53:22)

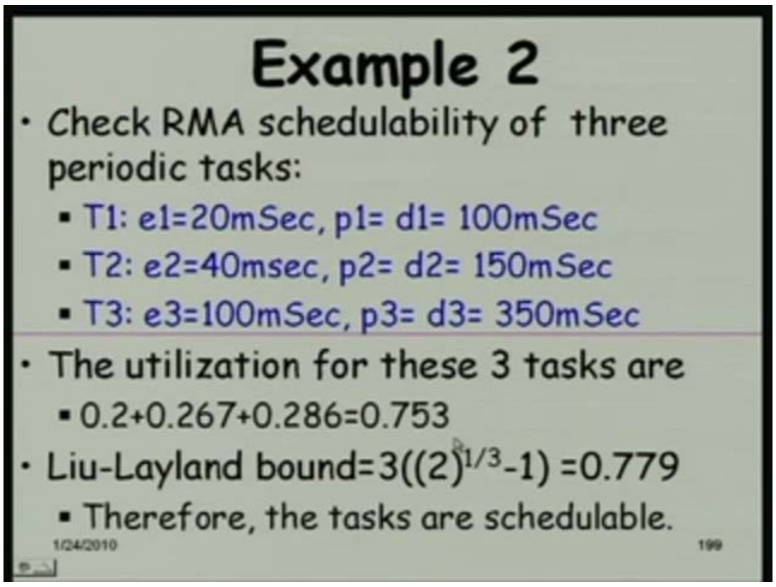


RMA Schedulability Test- III

- Completion time theorem (Liu and Lehoctzky):
  - When a set of tasks individually meet their first deadlines under zero phasing,
    - Then they are RMA schedulable for all task phasings.

1/24/2010 200

(Refer Slide Time: 53:27)



Example 2

- Check RMA schedulability of three periodic tasks:
  - T1: e1=20mSec, p1= d1= 100mSec
  - T2: e2=40msec, p2= d2= 150mSec
  - T3: e3=100mSec, p3= d3= 350mSec
- The utilization for these 3 tasks are
  - 0.2+0.267+0.286=0.753
- Liu-Layland bound=$3((2)^{1/3}-1)$ =0.779
  - Therefore, the tasks are schedulable.

1/24/2010 199

Please try that out again and yes.

Because if liu layland bond.

Yes.

Expression,

Yes.

(( )),

Yes. So, the liu layland’s paper that they published gives the exact derivation they have theoretically shown the result. It is a complex proof actually if we discuss the proof it will take several hours here and since the result is well established and used everywhere we have not bother to really prove it here give this proof here. Because it is a well accepted one and the proof is available those who really want to go through it many books actually give that proof.

But the book that we are fallowing even skip this proof saying that is actually a complex proof and not really central to learning real time tasks scheduling. But those who are theoretically inclined want to look at the proof yes you can look at it in several of the books and the original paper which is still available on the net just give the liu layland schedulability paper you will get the paper on the net available.

It is a nineteen seventy one paper. So, we will look at the completion time theorem which tells us whether a task set will schedulable even if it fails the liuland layland test. So, today we are running out of time now we will not discuss that because it will take some time fifteen minute or. So, so in the next class we will discuss about the completion time theorem it goes by the name liu and livosky’s theorem, we will stop here.