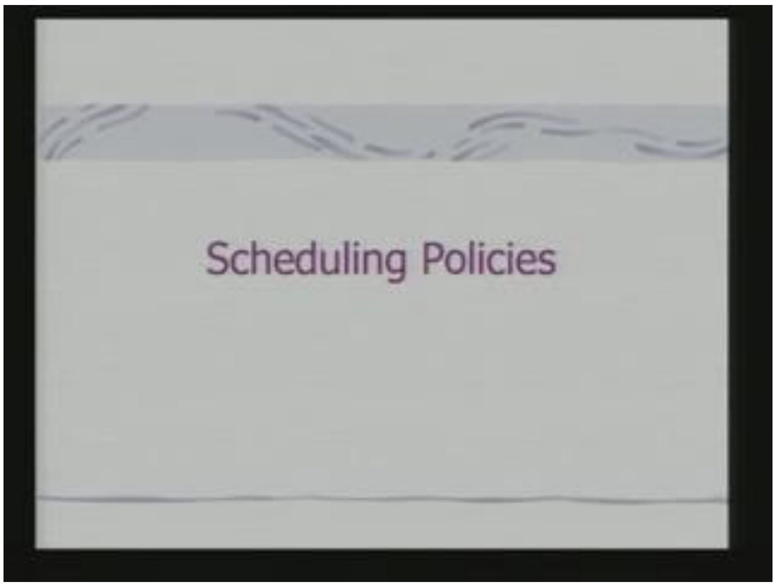
Embedded Systems
Dr. Santanu Chaudhury
Department of Electrical Engineering
Indian Institute of Technology, Delhi

Lecture - 21
Scheduling policies

In the last class, we had discussed the OS for the Embedded System and we had a brief review of the scheduling policies, because, many of the embedded system are real time systems. So, the basic job that OS kernel does, in such embedded system is to schedule jobs satisfying the deadlines. Today, we shall look at this scheduling policies in more detail.

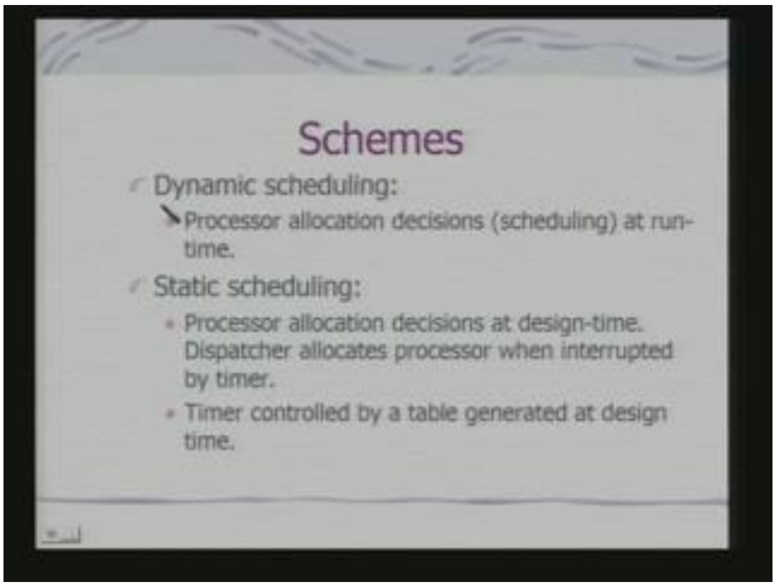
(Refer Slide Time: 01:43)



Scheduling Policies

The scheduling schemes can be broadly classified in to two basic classes.

(Refer Slide Time: 01:50)



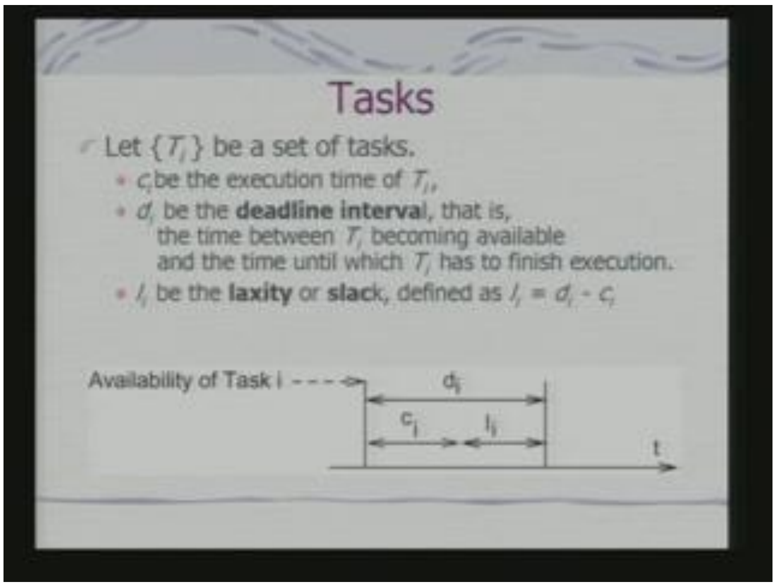
Schemes

- Dynamic scheduling:
  - Processor allocation decisions (scheduling) at run-time.
- Static scheduling:
  - Processor allocation decisions at design-time. Dispatcher allocates processor when interrupted by timer.
  - Timer controlled by a table generated at design time.

Dynamic scheduling and static scheduling. So, in dynamic scheduling processor allocation decisions or at run time. That means, when the process arrives or a scheduling point these decisions are meant. Static scheduling is start class of scheduling algorithms, why the schedule is decided. Before, hand and the decision is not need at each scheduling point. So, processor allocation decision need at design time. And dispatcher allocates processor, when interrupted by the timer.

And which process to be allocated to processor, the policy to be followed is decided a priory. And the timer is controlled by table generated at design time. Now, this is possible in an embedded system. Because, the set of jobs, that would run is fixed and known a priory in most of the cases.

(Refer Slide Time: 03:01)



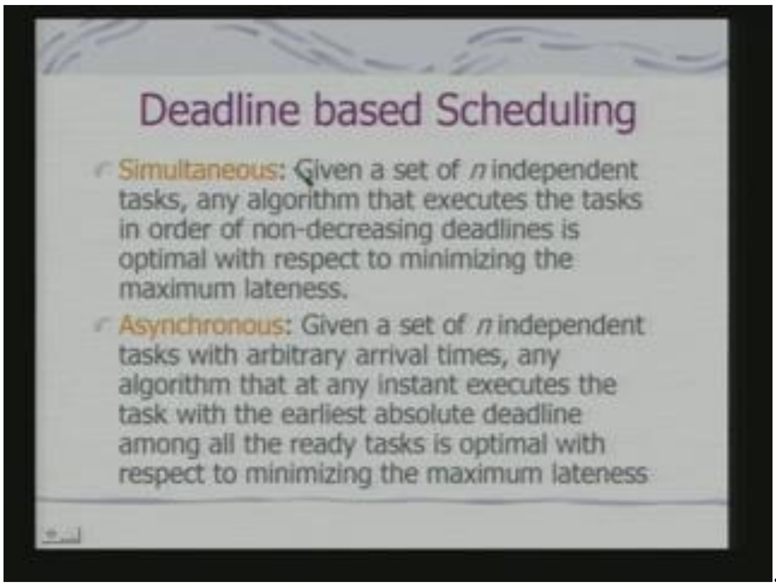
Tasks

- Let $\{T_i\}$ be a set of tasks.
  - c_i be the execution time of T_i,
  - d_i be the **deadline interval**, that is, the time between T_i becoming available and the time until which T_i has to finish execution.
  - l_i be the **laxity** or **slack**, defined as $l_i = d_i - c_i$

Availability of Task i
d_i
c_i
l_i
t

Still it review, the definition that we have looked at last in the last class for the purpose of starting the scheduling schemes. Let, T i be a set of tasks. c i is the execution time for the task. d i is the deadline interval. That is the time between T i becoming available. This is the point of the task becoming available. And the time until which T i has to finish execution. And l i is the laxity or slack defined as d i minus c i. So, these are the parameters on the basis of which the scheduling scheme will be defined. First, we shall look at deadline base scheduling.

(Refer Slide Time: 03:53)



Deadline based Scheduling

- Simultaneous: Given a set of *n* independent tasks, any algorithm that executes the tasks in order of non-decreasing deadlines is optimal with respect to minimizing the maximum lateness.
- Asynchronous: Given a set of *n* independent tasks with arbitrary arrival times, any algorithm that at any instant executes the task with the earliest absolute deadline among all the ready tasks is optimal with respect to minimizing the maximum lateness

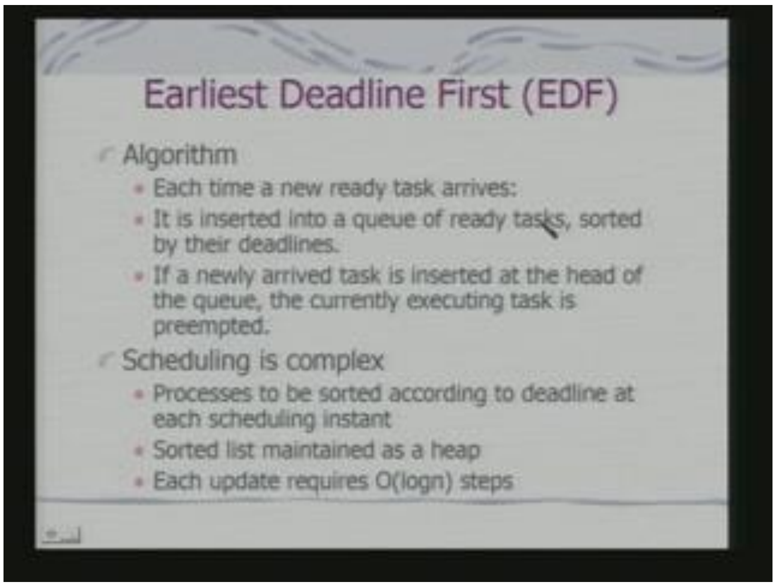
.

Now here, we are looking a two possibilities. One is simultaneous. That means, simultaneous arrival of all the job with deadlines. See, we have a set of an independent tasks. Any algorithm that executes the tasks in order of non decreasing deadlines is optimal, with respect to minimizing the maximum lateness. That means, we have all the jobs. And we know also the deadlines. So, effectively based on sorting the jobs according to the deadlines and executing them one after another. So, have the assumption is all tasks rives simultaneously.

The second possibility is the tasks may not all arrive at the same point in time. So, this is the asynchronous situation. So, given a set of n independent tasks with arbitrary arrival times any algorithm, that at any instant executes the tasks with the earlier absolute deadline. Among all the ready task is optimal with respect to minimizing the maximum lateness. So, these become a kind of a dynamic policy because, it is looking at the tasks in a dynamic fashion. Because, all of the tasks have not arrived in the same point in time.

So, this scheduling takes place as the tasks are arrived and as the deadlines of the tasks are examine at scheduling points. In fact, this algorithm is known as earlier deadlines first algorithm. And this is basically your deadline monitor algorithm. Now, what is in first in to note in that we are talking about independent tasks. That means, here tasks do not have dependence on each other.

(Refer Slide Time: 05:49)



Earliest Deadline First (EDF)

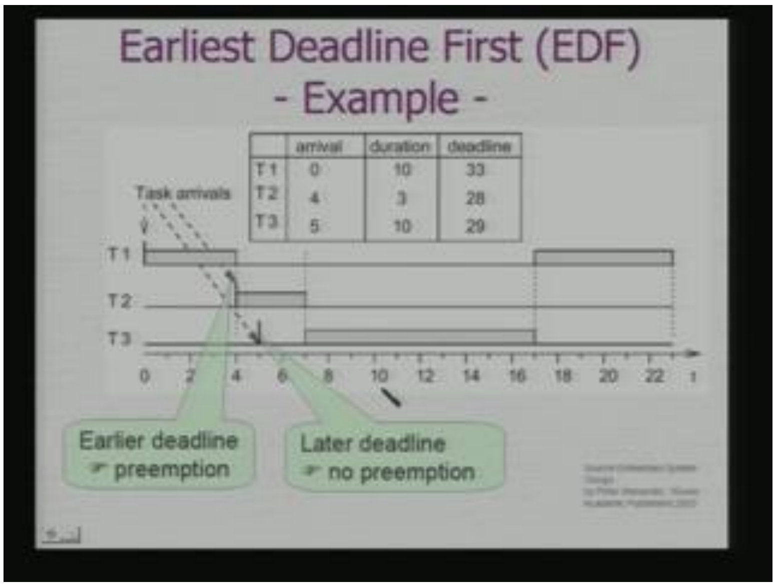
- Algorithm
  - Each time a new ready task arrives:
  - It is inserted into a queue of ready tasks, sorted by their deadlines.
  - If a newly arrived task is inserted at the head of the queue, the currently executing task is preempted.
- Scheduling is complex
  - Processes to be sorted according to deadline at each scheduling instant
  - Sorted list maintained as a heap
  - Each update requires O(logn) steps

So, let us look at this earlier deadlines, first in an algorithmic framework because, these algorithm is what is to be implemented by the scheduler. So, what happens is that, when a each time a new ready tasks arrives. It is inserted in a queue of ready tasks sorted by their deadlines. If a newly arrived tasks is inserted at the head of the queue the currently executing task is preempted. If it is suppose to have the time chunks or the time slices. Then, the time slices at the night to the current task. And time slices or allotted to the tasks, which has got the earliest deadline.

So; obviously, we can find this ordering of the queue has to be done incrementally. So, that leads to the point that scheduling is complex. Complex in terms of the time complexity, which is involved in the scheduling algorithm because, we are not really looking at the scheduling overhead. Because, if the time required for schedule becomes large then effectively what is happening, you CPU utilization is going down, CPU is not doing the tasks.

So, processes have to be sorted according to deadline at each scheduling instant. So, one we have doing is main thing the sorted list as a heap. And update requires order of login steps. Because, if you have to sort a each step. Then that complexity would be order of n square. So, you would not have to like that is, you have to use a kind of a data structure like a heap to maintain the ready queue for the purpose of scheduling.

(Refer Slide Time: 07:36)



Earliest Deadline First (EDF) - Example -

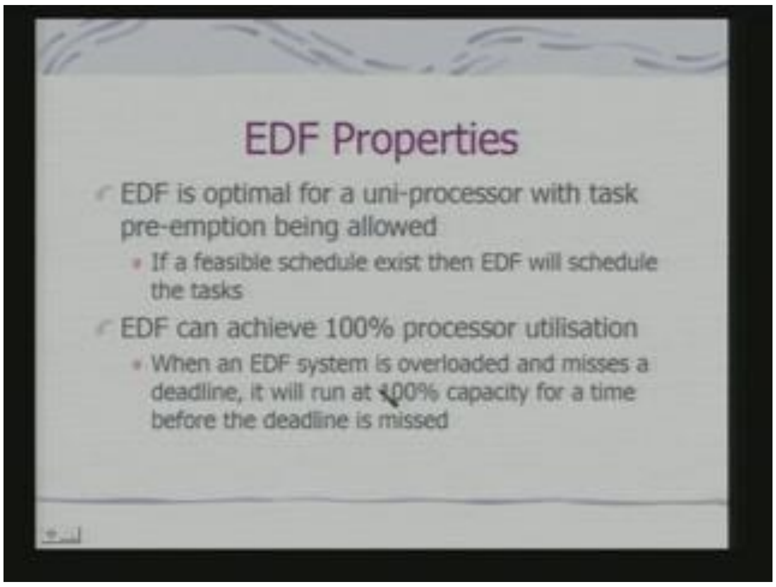
	arrival	duration	deadline
T1	0	10	33
T2	4	3	28
T3	5	10	29

So, let us take an example to understand this algorithm. This are the tasks, we are considering three tasks T 1, T 2, T 3. They have got this arrival instances and these are the durations. And these are the deadlines the duration is time taken for their computation. So, if you can do this, these are the points where the tasks arrive. So, this is where the task, T 1 arrives then task T 2 arrives then task T 3 arrives.

So, what happens first here, see these tasks T 2 arrives. And it has got a deadline, which is closer than that of T 1. So, the task T 1 will be preempted. And these will be scheduled now. Now when, the next tasks arrives that means, your task T 3. It has got a deadline later than that of T 2. That is why, T 2 is not preempted. And it continuous till it finishes it is processing job. Then only, your T 3 gets scheduled. And what you find here. T 3 gets scheduled before T 1 because, T 3 has got a deadline, which is earlier then, that of T 1.

So here, what you can find is you are scheduling in a dynamic fashion. That means, when ever these tasks is arriving depending on the deadline. The earlier task is preempted and if it has got an earlier deadline. This task will be scheduled.

(Refer Slide Time: 09:28)



EDF Properties

- EDF is optimal for a uni-processor with task pre-emption being allowed
  - If a feasible schedule exist then EDF will schedule the tasks
- EDF can achieve 100% processor utilisation
  - When an EDF system is overloaded and misses a deadline, it will run at 100% capacity for a time before the deadline is missed

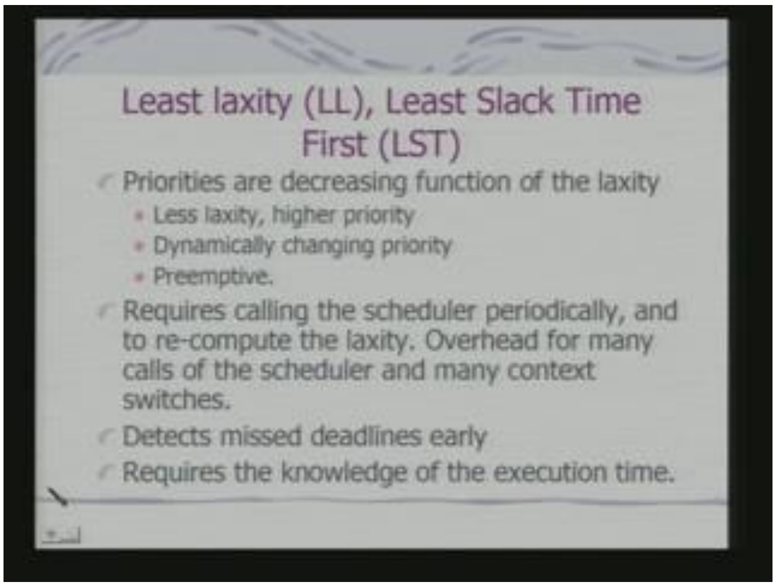
So, in that sense EDF is optimal for a uni-processor with the task pre-emption being allowed. Because, if I would not have the ability to preempt the task. The preempt the task T 1. Then it may so happen, that I might means the deadline for the other tasks. So, it points down is to that. If a feasible schedule exist, then EDF will schedule the tasks.

What does that mean that means, if somehow I can schedule the tasks. And meet deadlines, then I can do the same thing using EDF policy.

If an EDF can achieve 100 percent processor utilization obviously, when I talking about 100 percent processor utilization I am ignoring the time taken for scheduling the time taken by the scheduler and that of scheduling of the job. So, if there is a deadline needs, when I am adopting the EDF policy. What we shall find out or is that the processor is been utilized almost at 100 percent of the capacitor. That is really no time left for deadline to be meet.

So, these policy makes EDF attractive. But, the other side of it is what. That at each point you have to do scheduling. There is a contact switch, contact switch has got overhead scheduling algorithm has but overhead. And when you are talking about this processor utilization, we are not talking these overheads into account.

(Refer Slide Time: 11:18)



Least laxity (LL), Least Slack Time First (LST)

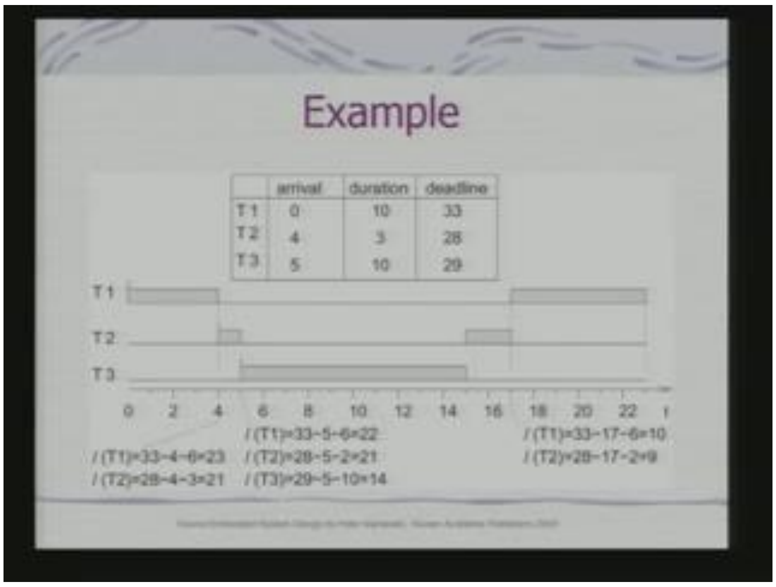
- Priorities are decreasing function of the laxity
  - Less laxity, higher priority
  - Dynamically changing priority
  - Preemptive.
- Requires calling the scheduler periodically, and to re-compute the laxity. Overhead for many calls of the scheduler and many context switches.
- Detects missed deadlines early
- Requires the knowledge of the execution time.

Related to this, you have got this algorithm, which is called least laxity or a least slack time first algorithm. Now in this case, priorities are decreasing function of the laxity. In fact, if you have seen the EDF also priority based scheduling, I am just telling how, the priority is to be determined. So here, what do you say, less the laxity higher the priority. And here also, the dynamically priority change and you would require pre-emption.

Because, these aspects will be very similar to that of your earliest deadlines based, earliest deadline based scheduling. Simply because, you laxity calculation a laxity value would also depend on the deadline. So, it request calling the scheduler periodically and re-computation of the laxity. In fact, overhead the many calls of the scheduler and many contact switches. This is overhead similar to that of the EDF.

But, what it can do, it can detect missed deadlines early because, it is looking at the laxity. So, the moment laxity becomes a negative in a kind of scheduling calculation, you can detect a possible deadline mix. But, these would require the knowledge of the execution time. In fact, EDF can be used even without the knowledge of the execution time. I just nick to know the deadline. And I can work with assumption. That, if the processor schedule it should be possible for me to, do what, finishes execution within the deadline. So, let us you get the example.

(Refer Slide Time: 13:08)



Example

	arrival	duration	deadline
T 1	0	10	33
T 2	4	3	28
T 3	5	10	29

$l(T1)=33-4-6=23$
$l(T2)=28-4-3=21$

$l(T1)=33-5-6=22$
$l(T2)=28-5-2=21$
$l(T3)=29-5-10=14$

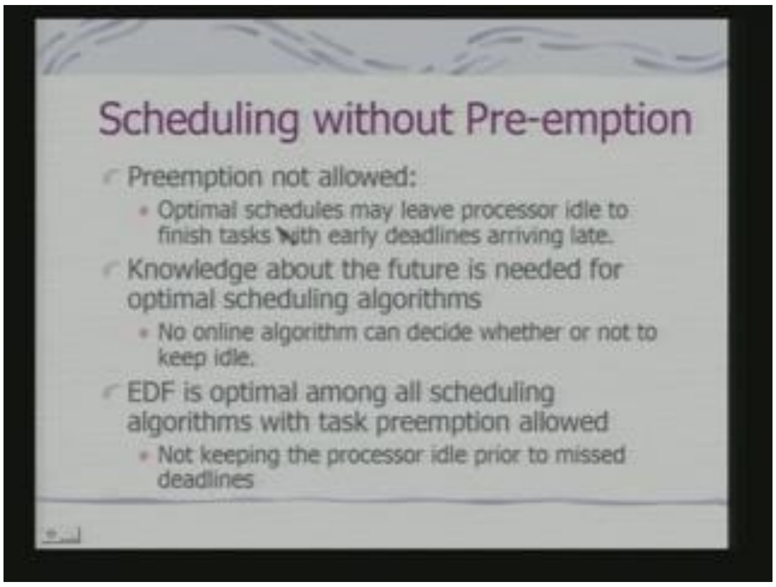
$l(T1)=33-17-6=10$
$l(T2)=28-17-2=9$

Example of least laxity first, so there are three jobs T 1, T 2 and T 3 they are the arrivals. And here, I know the duration and these duration becomes critical. So, you have got this T 1 and if I come to this point this is the point where the T 2 arrives when T 2 arrives I need to do a scheduling calculation. So, when I am doing a scheduling calculation, I shall work out the laxity.

This is the laxity, that I am calculated find 33 minus why it is 33 minus 4 minus 6. Because, I am looking at what is the current state. How much, time would you require to finish the job? And what is the deadline? So, this is the laxity for T 1, And this is the laxity for T 2. On the basis of that, what do have find. I find that laxity for T 2 is less. So, it should be scheduled. So, it get schedule at this point. Then, again at this point the T 3 arrives. I do the corresponding calculation and what I find is that for T 3 this laxity value is minimum.

Since, the laxity value is minimum, now T 3 gets scheduled and then, once T 3 finishes. Because, I have not considering arrival of any other job I shall go back for T 2. Because, the laxity here is more and accordingly, I shall come to this point. And at this point I can compute laxity and decide on which job to be scheduled next. So, this is the basic idea of less that is laxity based scheduling. But; obviously the problem here is that you need to definitely know the duration of the computations.

(Refer Slide Time: 15:07)



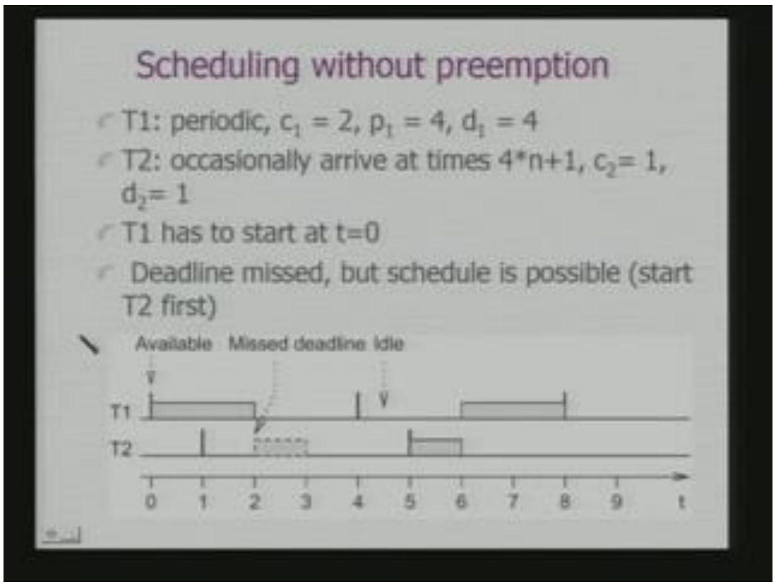
Scheduling without Pre-emption

- Preemption not allowed:
  - Optimal schedules may leave processor idle to finish tasks with early deadlines arriving late.
- Knowledge about the future is needed for optimal scheduling algorithms
  - No online algorithm can decide whether or not to keep idle.
- EDF is optimal among all scheduling algorithms with task preemption allowed
  - Not keeping the processor idle prior to missed deadlines

Now, when you really do not have a pre-emption, what will happen? If I try to talk about the optimal schedules, Optimal schedules in the sense of making sure, the deadlines are made. And the leak ness is minimized. It will leave processor to ideal to finish tasks with deadlines arriving late. So, there may be under utilization of the processor. Particularly for, case really you do not want pre-emption. And knowledge about the feature is needed optimal scheduling algorithm.

Because, without that knowledge I cannot make sure, that the deadlines will be meet. Now, I have already say that the EDF is optimal, if the task pre-emption is allowed. And it not keepings the processor that will prior to missed deadlines. Now, if pre-emption is not allowed with the deadlines. And if you really not have complete knowledge about the feature, what can happen.

(Refer Slide Time: 16:05)



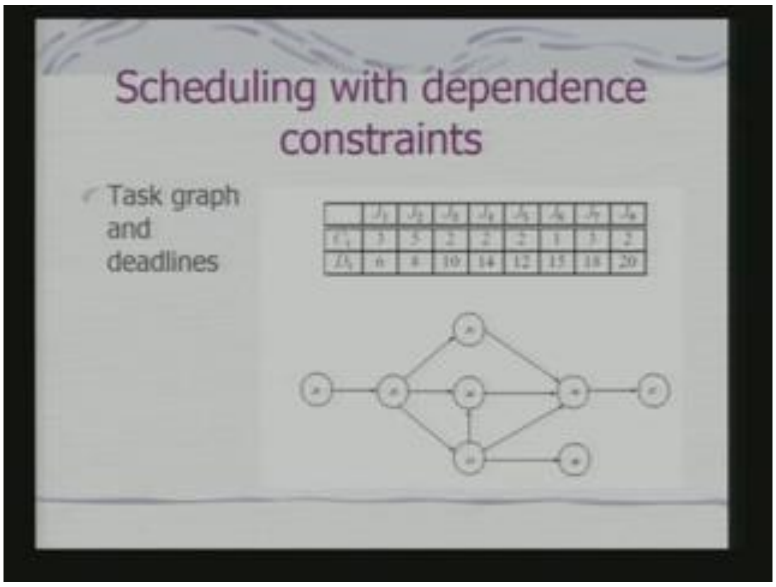
Scheduling without preemption

- T1: periodic, $c_1 = 2$, $p_1 = 4$, $d_1 = 4$
- T2: occasionally arrive at times 4*n+1, $c_2 = 1$, $d_2 = 1$
- T1 has to start at t=0
- Deadline missed, but schedule is possible (start T2 first)

Let us, look at this example. In this case, a task T 1 is periodic. Now, what is the interesting about periodic task? You know the arrival patterns. So, in a sense you know, that in future what kind of tasks are expected. But, T 2 is not strictly periodic. And we are talking about an example, where T 2 occasionally arrive at times let you say 4 into n plus 1. And it is time requirement is that is time taken for computation is 1, I needs deadline is also 1.

T 1 has to start at T equal to 0. Because, it has it is their and what happens is since no pre-emption is allowed although T 2 arrives this point my deadline will be missed. And the process will remain ideal this point. Because, I can have again I had a T 2 only at this point and time. But, where is a schedule, which can exist. Schedule can exist, if I start T 2 first. But, in that case I need to know the T 2 will come. So, if I am not having the ability to preempt the processor. I cannot really meet the deadlines if I have incomplete knowledge about the feature arrival of tasks.

(Refer Slide Time: 17:36)



Scheduling with dependence constraints

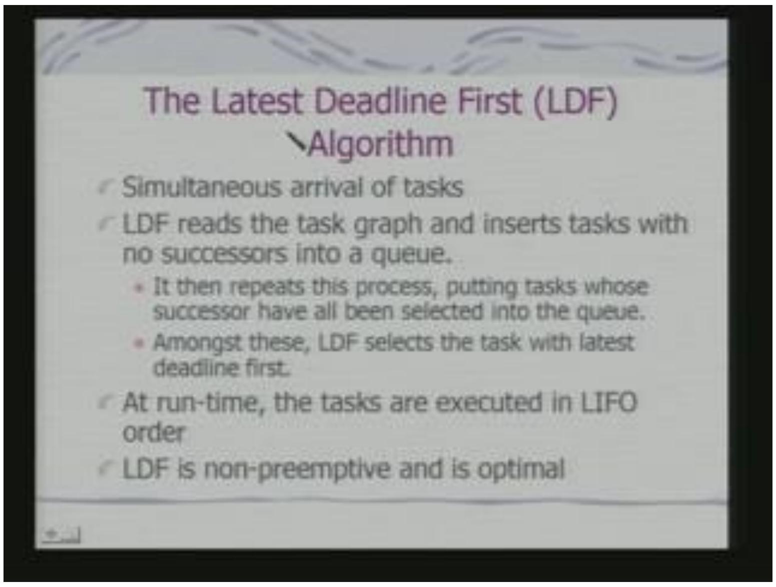
- Task graph and deadlines

	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
C_i	3	5	2	2	2	1	3	2
D_i	6	8	10	14	12	15	18	20

Then, we look at another possibility. So for, we are talking about tasks without any kind of dependences. But, know tasks can really have dependences among themselves, because of the sheared resources. Also, because of one computation may be dependent on another. So, what we represent, we represent this dependence by what is called a task graph. So, in this case, I have got the different jobs J 1 to J 8. And these are their computation times and these are their deadlines.

And, the dependence of the task is given by this task graph. The arrow says that, I cannot really have j 2 executed before j 1. So, the problem now becomes more complex. Because, I had got the deadline as well as I got the dependence. So, I cannot arbitrarily schedule the tasks without taking in to a account their dependences. So, how to do that, so in this case, you get what is called a latest deadline first algorithm.

(Refer Slide Time: 18:45)


The Latest Deadline First (LDF) Algorithm

- Simultaneous arrival of tasks
- LDF reads the task graph and inserts tasks with no successors into a queue.
  - It then repeats this process, putting tasks whose successor have all been selected into the queue.
  - Amongst these, LDF selects the task with latest deadline first.
- At run-time, the tasks are executed in LIFO order
- LDF is non-preemptive and is optimal

So, all this not that the task will be the latest deadline is getting schedule first. So, if that happens if you cannot really meet the deadlines in many cases. So, let us see what happens in the algorithm. So, you got simultaneous, this is the assumption work with a assumption it is a simultaneous arrival of tasks. So, LDF this algorithm reads the tasks graph and inserts tasks with no successors in to a queue. It, then repeat this process putting tasks, whose successor have all been selected into the queue.

In fact, it is a kind of a topological fated, but in an inverse way. So, you have got the sorted lists of tasks in the queue. What does this mean? The task, which is at the top of the queue actually, does it have a independence. In fact, the task which has been last entered into the queue will have not any dependence. So, among these in fact, at each point when you are making a choice for a task be put in to the queue. You select the tasks with the latest deadline first.

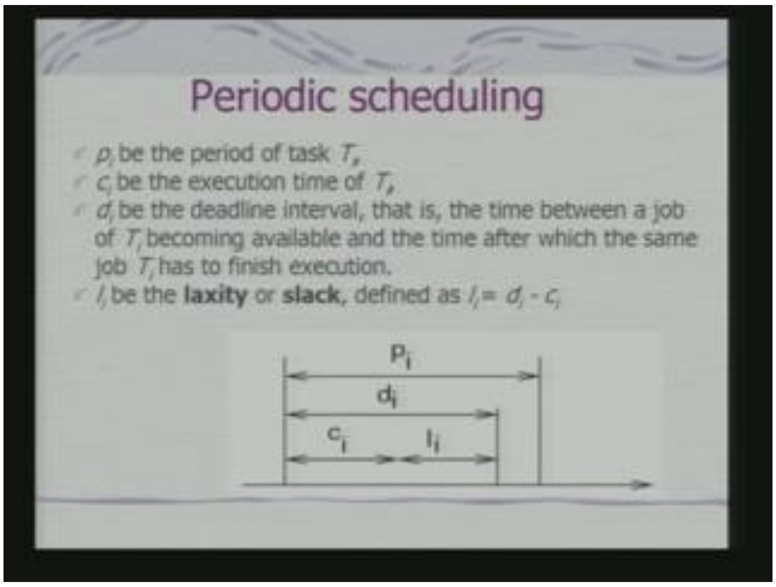
You select the task with the latest deadline first for putting in to the queue. If you know, thing about the queue what happens. In the queue at the end of the queue, you have got a task which is not depended on any other task. And if there a few tasks over their which have not dependent on any of the other tasks. Then of the top of the queue will be the task, which has got what the earliest deadline. Because, have chosen the task with the latest deadline to go before this tasks in the queue.

So, the first condition is what dependence. So, if I, if the dependence can be satisfied, there will be ties among the nodes and the task graph. I resolve ties among the nodes in the task graph on the basis of their deadline. And I select the node with the latest deadline, first to go in to the queue. At run time, what do I do, I remove the tasks, but I executes the tasks in last in first out order. So, the task having earliest deadline, an having no dependence will be executed first. And that order would be preserved at this entire queue.

Now, obviously we can realize here that, this is possible only if you have got a simultaneous arrival of tasks. That is you know this tasks, as well as the dependence as the deadlines before you have drawn up the scheduling policies. So, LDF in a sense as non preemptive when is optimum. Why non pre-emptive? Because, you can define a schedule and that, schedule would be decided by what, by the task graph and the deadlines.

So, consequently you actually need not preempt the process. Because, you are assuming that no tasks are arriving, while the task corresponding to the task graph is in execution. Next, we shall look at periodic jobs. Because, so for we have not strictly considered, the periodicity of the tasks for the purpose of scheduling.

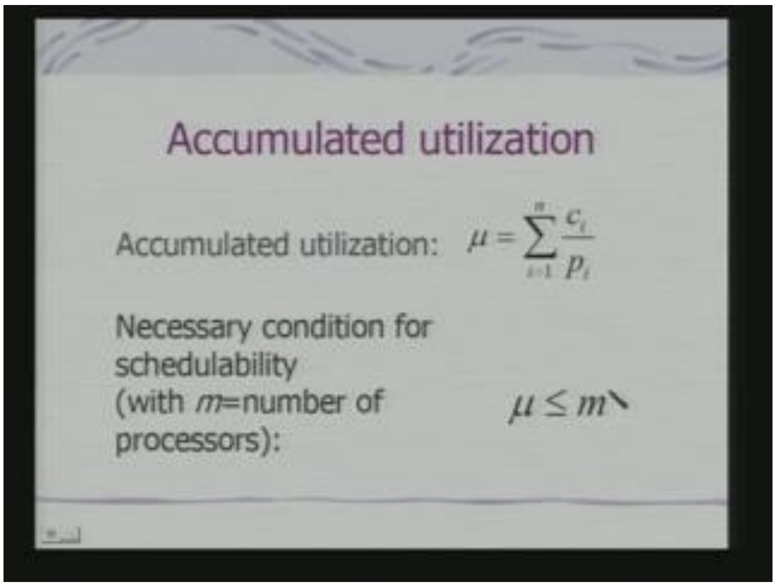
(Refer Slide Time: 22:27)


Periodic scheduling

- p_i be the period of task T_i
- c_i be the execution time of T_i
- d_i be the deadline interval, that is, the time between a job of T_i becoming available and the time after which the same job T_i has to finish execution.
- l_i be the **laxity** or **slack**, defined as $l_i = d_i - c_i$

So, when we are looking at periodic scheduling, the parameters which will be important are obviously, c i l i which we have already defined d i. Now, added to that we shall have period P i. Now, here what we have seen now looking at is a situation where my deadline is not same as that of the period. In fact, deadline can be before that of the period. Deadline can be after that of the period, deadline can be actually coincident with that of the period. There are different possibilities of the deadline. Many of the scheduling policy is for the periodic job, that we shall consider for them we shall assume that my deadline is actually coincident with that of the period.

(Refer Slide Time: 23:23)



Accumulated utilization

Accumulated utilization: $\mu = \sum_{i=1}^{n} \frac{c_i}{p_i}$

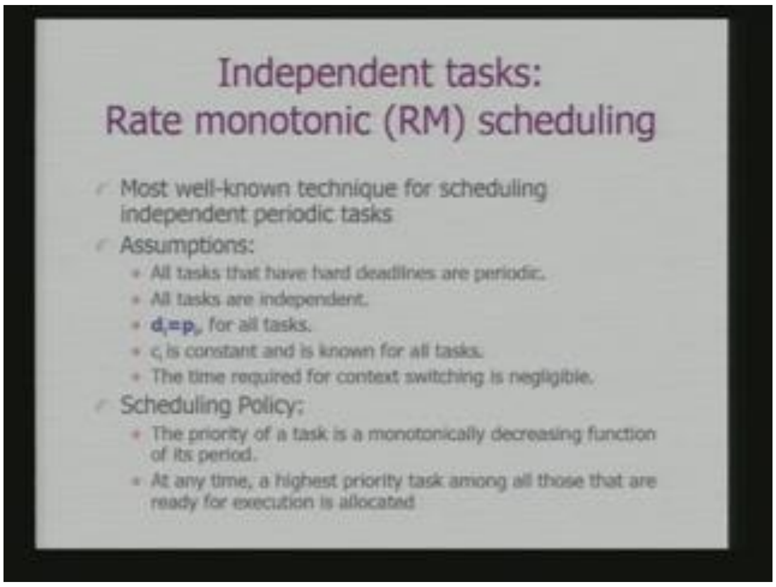
Necessary condition for schedulability (with m=number of processors): $\mu \leq m$

Now, for this kind of scenario, we can look at the processor utilization. And the necessary condition for schedule ability. Because here, if you look into it what you are getting. That, if I have c i by P i. P i is the period, c i is the computation time, if I know the computation time required for each task. Then, give me the fraction of the period, which is occupied by the actual computation of the task. If I sum it up your all such tasks, all such periodic tasks which are to be executed.

There are sum cannot exceed 1, if they are sum exceeds 1. That means, it cannot be accumulated by the processor. So, and the accumulated utilization will be mu. And mu is schedulable is mu is less than equal to n. If n is a number of processors. In a uni-processor system typically, your embedded system m would be 1. Now, therefore, what we are getting, we are getting a schedule ability test condition.

Now, why schedule ability important? Because, if I have the periodic set of task, we know the nature of features arrivals on the basis of the feature arrival pattern I can decide, whether these set of tasks schedulable or not. What is meant by schedulable? Whether, I can run this tasks and meet the deadline or not. In fact, if we do the analysis, what will find these gives me a loose lower bound. You can even exceed this and still can do the scheduling. But, this gives me a kind of a bound corresponding to the schedule ability. It is a weaker test to check, whether you can schedule a set of periodic tasks.

(Refer Slide Time: 25:20)



Independent tasks: Rate monotonic (RM) scheduling

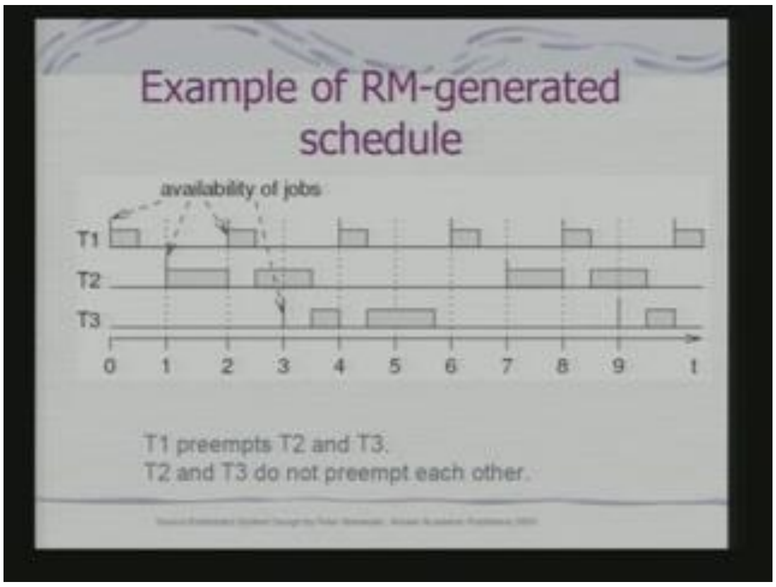
- Most well-known technique for scheduling independent periodic tasks
- Assumptions:
  - All tasks that have hard deadlines are periodic.
  - All tasks are independent.
  - **$d_i = p_i$**, for all tasks.
  - c_i is constant and is known for all tasks.
  - The time required for context switching is negligible.
- Scheduling Policy:
  - The priority of a task is a monotonically decreasing function of its period.
  - At any time, a highest priority task among all those that are ready for execution is allocated

So, for the periodic task the most well known algorithm is rate monotonic scheduling algorithm. Now, here what are the assumptions. We are assuming, that all task that have hot deadlines a periodic. All tasks are independent. And d i is equal to P i. That means, deadline coincides with that of your period. c i is constant, that is from period to period the computation overhead a computation requirement does not vary, it is constant then is known a priory for all tasks.

The time required for contact switching also we are assuming will be negligible. To way we have done an analysis for EDF. So, the scheduling policy is what. The priority of the task is a monotonically decreasing function of it is period. And that is preciously reason why, we called is rate monotonic scheduling, the name comes from there. So, at any time a high priority task among all those, that I ready for execution is allocated the processor.

So, obviously here, what is the assumption, assumption you know the periods. So, you know the arrival patterns, you know their computation times your deadline coincides. So, you can schedule right at the beginning. And you need not to the scheduling and each periodic arrival of tasks.

(Refer Slide Time: 26:56)



Example of RM-generated schedule

T1 preempts T2 and T3.
T2 and T3 do not preempt each other.

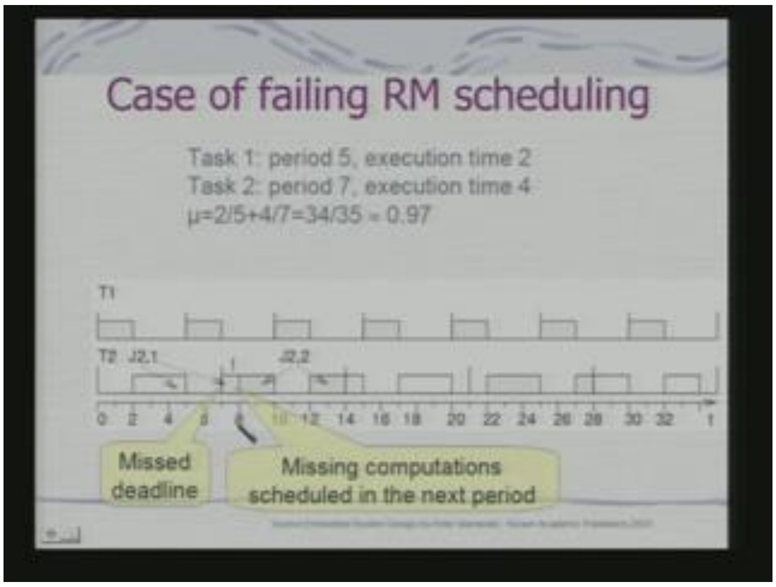
So, this is an example, so if you look in to it. Here, what we are showing, that T 1 is, is at this point. And there are three tasks. And availability of jobs is T 1, here it is T 2 and here at this point you have got the T 3. Now, if you at deciding on the schedule, when the why get the pre-emption you look in to it. That T 1 preempts T 2 and T 3. Now, why this is ping preempted. See, if you look in to it T 1, when it arrive. It has been decided on a scheduling priority.

When your T 2 arrives what happens, corresponding to the T 2, what is being found is that T 1 period is smaller. So, it will have a higher priority. If it has a higher priority because, why it is arrives, see, T 2 arrives here first and then T 2 arrives again at this point. T 2 arrives again at this point what happens, the period for T 1 is smaller than that of T 2. So, T 1 will have a higher priority than that of T 2. So, when T 1 is scheduler this point, it has finished it job.

So, when T 2 arrives the T 2 will schedule at this point. But, if the T 1 next time T 1 arrives, T 1 has got a higher priority than that of T 2. So, T 1 will preempt T 2. T 1 will preempt the T 2. Now, again when the T 3 arrives, T 2 and T 3 what we say, do not preempt each other. Why? Because, the periodic is different and the priority wise they are schedule in such a way, that they do not preempt each other.

Now, what is to be noted here is also important, that here it is with the R M it is note, a non preemptive policy, it is a preemptive policy. And what we are telling is the movement, I have got an arrival of periodic stream of jobs I assign it a priority. And then, the scheduling takes place on the basis of this priority.

(Refer Slide Time: 29:25)



Case of failing RM scheduling

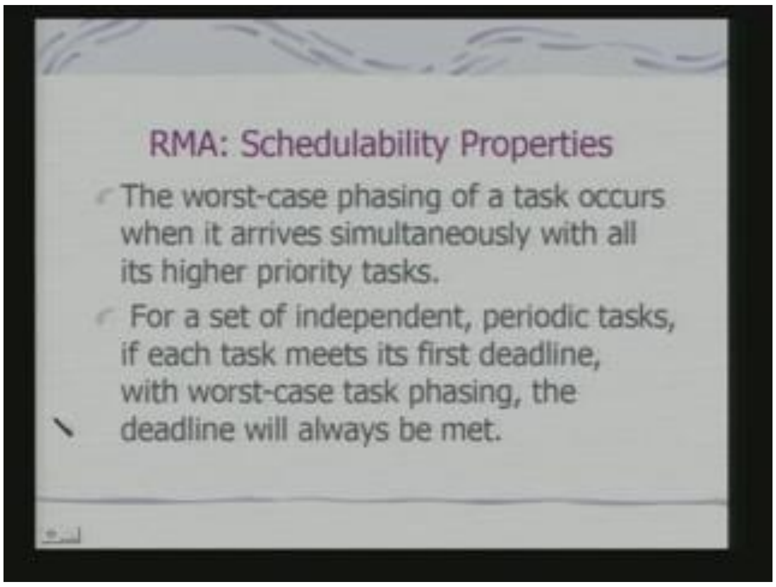
Task 1: period 5, execution time 2
Task 2: period 7, execution time 4
$\mu=2/5+4/7=34/35 \approx 0.97$

T1
T2
J2,1
J2,2
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 t
Missed deadline
Missing computations scheduled in the next period

Next, what we look at is a issue where R M is failed. So, what we are looking at task 1 is task 1 is period 5 execution time equal to 4. Task 2 is period 7 execution time is 4 and if I to the summation to the utilization, although we are getting a 0.97. But, if I am doing a pure R M scheduling, there could be a deadline mix. So, what happens here? Here the task T 1 has got a priority, which is higher then that of T 2.

Now, because of that what is happening? T 1 is scheduled here, T 1 finishes it job. Then only, T 2 can be actually scheduled. If T 2 scheduled here, now T 2 effects execution time is 4. So, it is completing what three periods. After it completes three periods, what happens, your T 1 next job of T 1 arrives. Since, T 1 has got a higher priority, then this will be pre-empted. This T 1 will be executed. Still T 1 will be executed. I shall miss the deadline and missing computations getting schedule in the next period. So obviously, it is not that R M can in the sense meet always a deadline.

(Refer Slide Time: 31:10)


RMA: Schedulability Properties

- The worst-case phasing of a task occurs when it arrives simultaneously with all its higher priority tasks.
- For a set of independent, periodic tasks, if each task meets its first deadline, with worst-case task phasing, the deadline will always be met.

So, on the basis of this rate monotonic scheduling the next thing that come a kind of rate monotonic analysis. So, rate monotonic analysis is specifically for the purpose of deciding schedule ability of a set of tasks. Whether, a set of tasks can be scheduled meeting the schedule ability construct. Now, schedule ability for that purpose, why we called as a rate monotonic analysis. I can adopt may be an different scheduling policy. But, what we are looking at, we are looking at that if they are scheduled according to the rate for a set of periodic tasks. How we can commend up on their schedule ability?

So, we need to understand, what is the concept of all the worst case phasing of a task? The worst case phasing of a task occur, when it arrives simultaneously with all it is higher priority tasks. So, that means, if it simultaneous with all it is higher priority task. That means, the higher priority task will be given the processor share. So, that would be the worst case for this, so called lower priority task. And for a set of independent periodic task, if each task meets its first deadline with worst case task phasing the deadline will always be met.

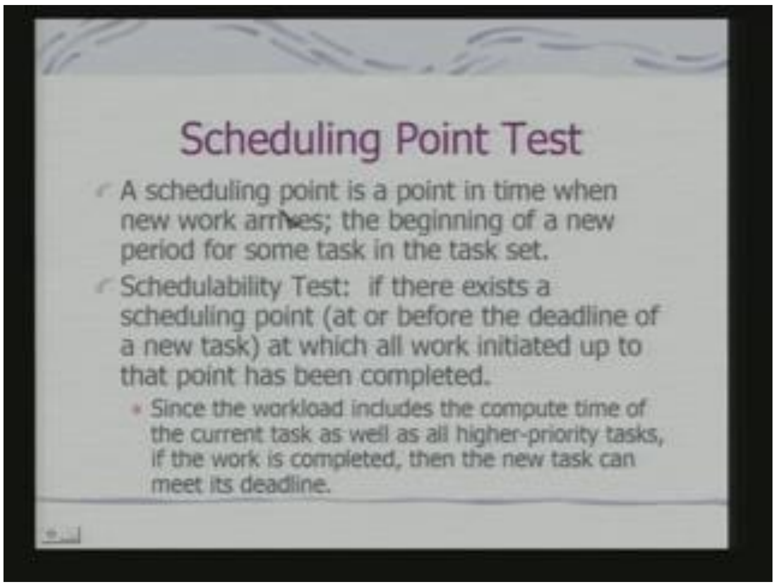
In fact, this has been formally shown. And this in test for deciding account schedule ability. Now, what did you implies in this cases also, that you can decide a priority on based up on factor not just periodic. There will be a other reasons to associate higher priority to processes. So, if you consider the mix of the processors is such, that all higher priority processors are existing. When, I consider the scheduling of a lower priority process, we land up in to what is called worst phase.

And if in the worst phase, we can show that a task can be scheduled meeting it is first deadline. Then, it will continue to meet for all subsequent periods because, I have done the worst case analysis. So, for the purpose of deciding, whether a set of task can be scheduled because, these becomes a very important requirement when we designing an embedded system. Because, I am telling you that in many of the cases, the set of task the embedded system will handle or known a priority.

If they known a priory, then if I can do a worst case schedulable in a analysis. Then only I can say, then given the current hardware, given the current wires. This mix of tasks can be supported on that embedded appliance, such that the deadlines are meet. So, what kind of schedule ability test, that we should carry out for this kind of worst case phasing. One such test is known as scheduling point test. There have various test case we have proposed.

And they have the various property we have test. We shall not go in to them, we just look in to one such test. Because, it is intuitively easier to understand, this is called scheduling point test.

(Refer Slide Time: 34:37)



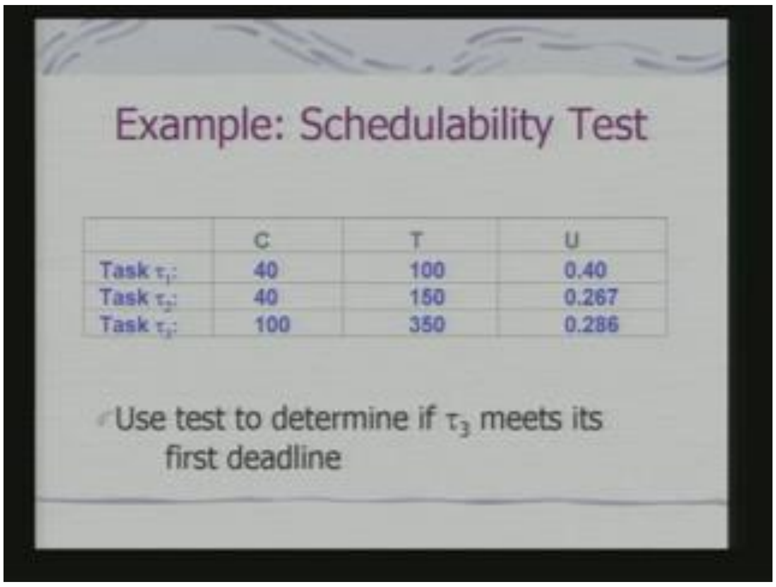
Scheduling Point Test

- A scheduling point is a point in time when new work arrives; the beginning of a new period for some task in the task set.
- Schedulability Test: if there exists a scheduling point (at or before the deadline of a new task) at which all work initiated up to that point has been completed.
  - Since the workload includes the compute time of the current task as well as all higher-priority tasks, if the work is completed, then the new task can meet its deadline.

A scheduling point is a point in time, when new work arrives or the beginning of a new period for some task in the task set. We have seen, that is exactly the point where the task set schedule. So, such point we are referring to us scheduling point. And what is the exact schedule ability test if there exists scheduling point at or before the deadline of a new task. At which, all work initiated up to that point has been completed.

Then, the new task is schedulable. Because, there is nothing really ((Refer Time: 35:18)). So, what we says, since the work load includes the compute time of the current task as well as all higher priority tasks. If the work is completed then the new task can meet it is deadline. So, at which all work initiated up to that point has been completed. If there exists a schedule ability point, at which all work initiate up to that point has been completed. That means, I can really meet the deadline. In fact this is the very intuitive way to do the checking. Selects look at graphically, how it is look like.

(Refer Slide Time: 35:54)



Example: Schedulability Test

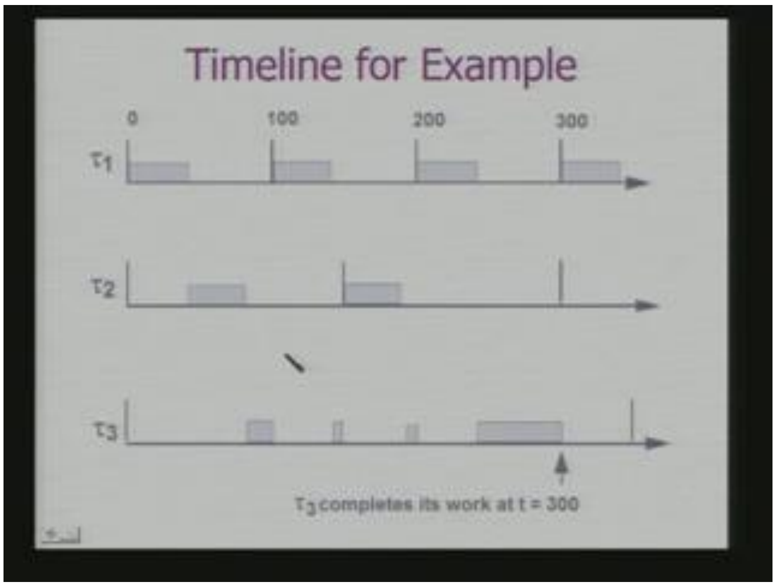
	C	T	U
Task τ_1:	40	100	0.40
Task τ_2:	40	150	0.267
Task τ_3:	100	350	0.286

- Use test to determine if τ_3 meets its first deadline

Consider an example, where have got mix of three task, task 1, 2, 3. And each one of them has computation 1 and 2, 40 time computation time 40 and this 100. And their units specifically you look in to, it is a same common unit. This is basically the time, corresponding to the deadlines. And this is it will be and here what we have consider the period and the time. For, the deadline may be actually coincident. And this is the utilization, if you look at corresponding to the periods. That is consumed by the computation if they are schedule.

Now, I would like to use the schedule ability test to determine, if the third task meets it deadlines, first deadline. In fact, if you look in to the RMA, depending on the RMA are rate monotonic scheduling policy. Obviously, what I shall have task T 1 higher priority, task T 2 next priority, then I get really the task T 3.

(Refer Slide Time: 36:59)



Timeline for Example

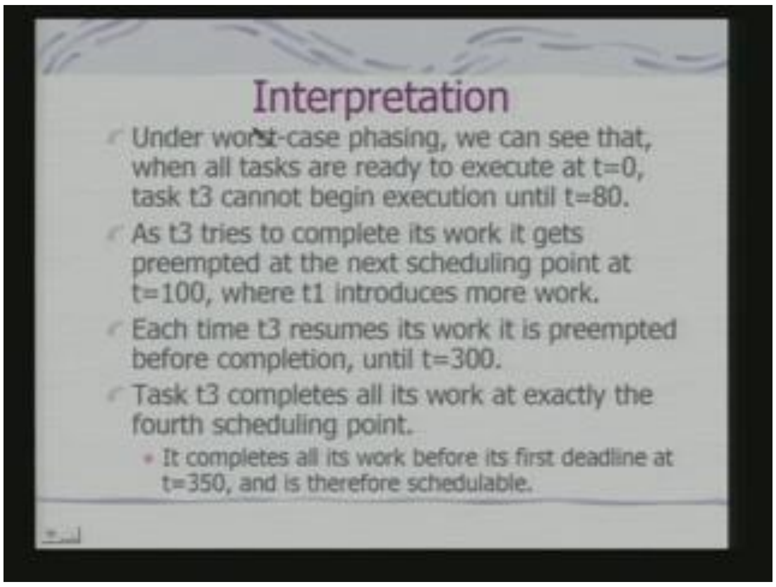
Say, look at the time line for example. So, in this time line if you see, here exactly the task T 1 is getting scheduled. And I assume the t is the periodicity as well as the deadline. The time T 2 would be scheduled only at this point. And what we are looking at. Only when, both T 1 and T 2 is not there. I can schedule T 3. So, T three is getting schedule at this point for some period. Then, T 1 arrives again T 3 is getting scheduled. It again T 1 is getting scheduled.

For small interval T 3 can come in. Now, T 2 the task 2 comes in how to second task comes in. So, the task T 3 get preempted. It comes back here. Again it comes back and finishes the task at this point. Say, if you look at this, this is a scheduling point. This is a scheduling point, because the other two tasks are arrived. And if I look at this scheduling point, by this scheduling point all the jobs, which had been started have finished. That means, I can have my what, complete task 3, complete it is work by t equal to 300, at time 300 and hence it meets the deadline.

So, this is become the schedule ability point. If such a schedule ability point exists, for a meets of a task then I say, that a task is schedulable meeting the deadline. That such a schedule ability point should exist at a time point, which is less than the deadlines for that task. And in fact, you can realize that seems this periods and known a priory the computation time are known as priory. I can carry out the schedulable test, before we actually what do you say, admit a job.

Admitting a job means whether a job would join the ready queue. If I join ready queue, if the jobs joins the ready queue then only it becomes a candidate for scheduling. If it fails schedule ability test, the job will not join the ready queue. And the system will flag that this request cannot be serviced. So, this is basically an interpretation of what we had already look that.

(Refer Slide Time: 39:25)

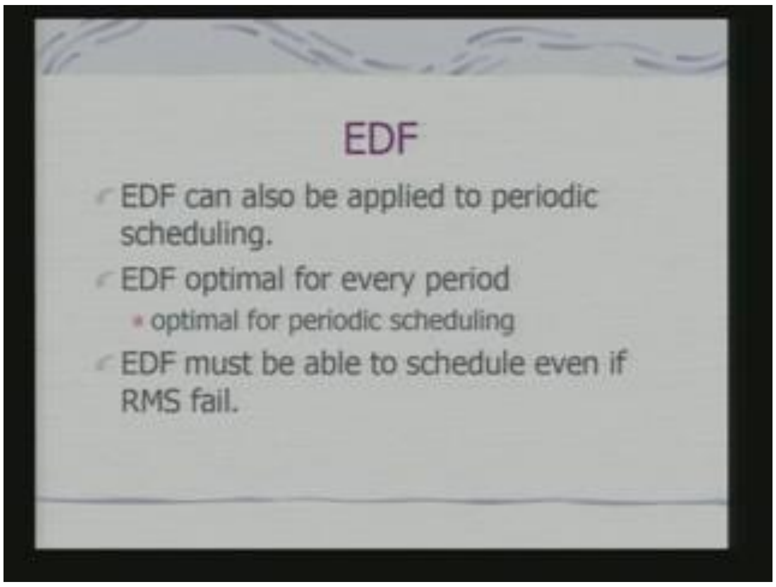


Interpretation

- Under worst-case phasing, we can see that, when all tasks are ready to execute at t=0, task t3 cannot begin execution until t=80.
- As t3 tries to complete its work it gets preempted at the next scheduling point at t=100, where t1 introduces more work.
- Each time t3 resumes its work it is preempted before completion, until t=300.
- Task t3 completes all its work at exactly the fourth scheduling point.
  - It completes all its work before its first deadline at t=350, and is therefore schedulable.

So, what we say the each time T 3 resumes it is work. It is preempted before completion until t equal to 300. And task T 3 completes all it is work at exactly the fourth scheduling point. It completes all it is work before it is first deadline at t equal to 350, and is therefore schedulable. And you also considered what, the worst phase scenario, because all task with higher priorities existing when we where, considering schedule ability of the task T 3.

(Refer Slide Time: 40:00)

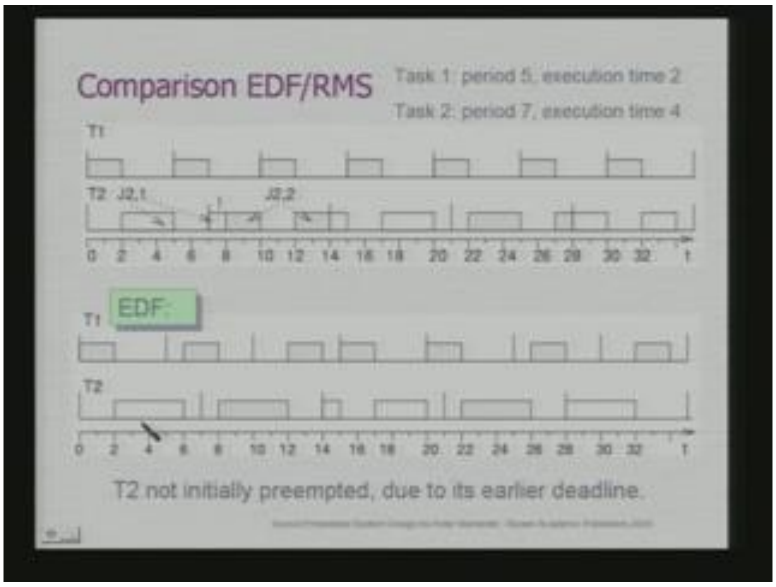


EDF

- EDF can also be applied to periodic scheduling.
- EDF optimal for every period
  - optimal for periodic scheduling
- EDF must be able to schedule even if RMS fail.

In fact, earlier deadline first can also be applied to periodic scheduling. And in fact, what you say the EDF since it is a optimal with the pre-emption is optimal also for periodic scheduling. And EDF must able to schedule, even if RMS fail. In fact, that is the basic idea you find because, EDF can actually schedule according to the deadlines. And it can preempt the schedule in an dynamic fashion.

(Refer Slide Time: 40:32)



Comparison EDF/RMS

Task 1: period 5, execution time 2

Task 2: period 7, execution time 4

T1
T2
J2,1
J2,2
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 t
EDF:
T1
T2
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 t

T2 not initially preempted, due to its earlier deadline.

So, let us at look at this example. This an example where, there is a deadline miss here. There is a deadline miss here and in here already see this example next schedule it in the subsequent phase. But, what is EDF do here. EDF at this point prevents T 2 been pre-empted. Why? Because, it is now using at this scheduling point, because what happens, when this T 1 arrives it recalculates the priority of the processes. Since, it recalculate priority of the processor, now what happens the T 2 has earlier deadline. So, T 2 will not be pre-empted.

So, T 2 will continue to finish, it will meet the deadline. Then only T 1 will get scheduled. So obviously, EDF can meet the deadline in cases where R M fails. Now, but what is the Advantage of R M rate monotony. Advantage is this priority is decided right at the point at when the task arrives. So, the priority is decided only once. But, in this case, in case of EDF what you find. That at every scheduling point, priority is being determined dynamically.

In that sense, EDF is dynamic scheduling algorithm compare to that of rate monotonic. And that is why EDF is also got a more over head in terms of scheduling type. So, but I can this example illustrate there are cases, where R M would failed. But, EDF will meet the requirement.

(Refer Slide Time: 42:25)

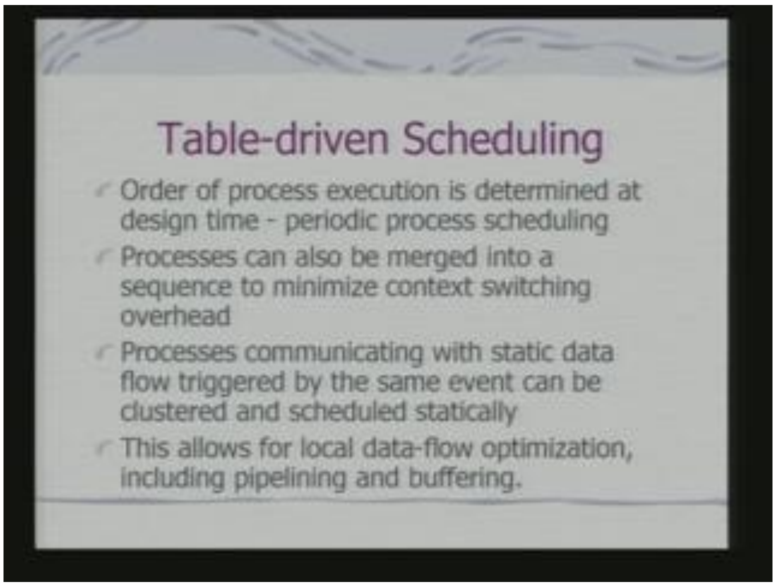


Table-driven Scheduling

- Order of process execution is determined at design time - periodic process scheduling
- Processes can also be merged into a sequence to minimize context switching overhead
- Processes communicating with static data flow triggered by the same event can be clustered and scheduled statically
- This allows for local data-flow optimization, including pipelining and buffering.

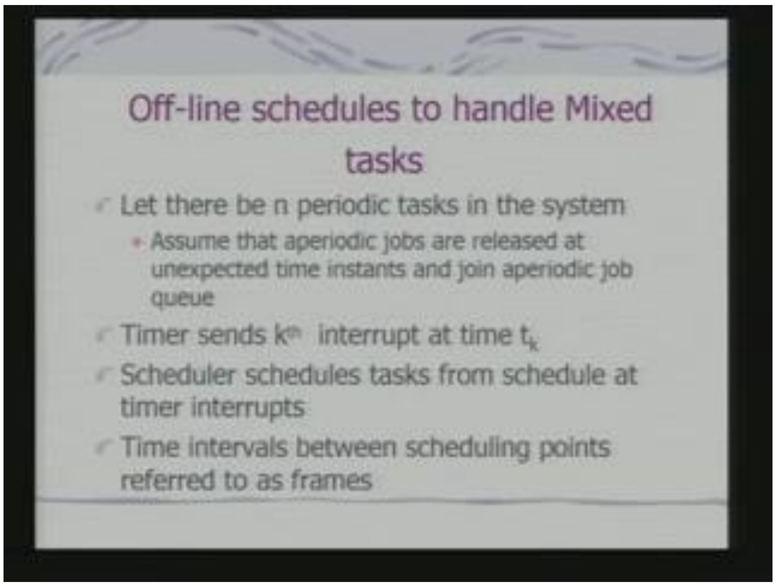
Now, what we are coming to is a static scheduling scheme, static scheduling is scheduling done completely at the time of designing and not really, in a run time. So, we can look at, this table driven scheduling is a way of implementing that. See order of process execution is determined at design time. So, in fact this can be done for a strictly periodic process scheduling, if I know strictly the periods of the processes.

Processes can also be merged into a sequence to minimize context switching overhead. Because, this you are deciding a priory during design time itself. And the processes communicating with static data flow, triggered by the same event can be clustered and schedule statically what does it mean, processes so I am talking of now depended processes. Because, there has been data flow between the processes triggered by the same event can be clustered and schedule statically. That means, they would work in a sequence because, it is a data flow between them.

And this allows for local data flow optimization including pipelining and buffering. Because, this data flow becomes critical because, of the pipelining. Because, the pipelining if pipeline is not completely full. I cannot really exploit the pipelining architecture. So, in this case if I know the pattern of the data flow, between the dependent tasks. And if I can do the scheduling at the design time, I can make sure the data flow is search that the pipeline is getting optimally utilized.

So, theses are possible, if you really know the job mix at the design time. You can take care of the pipelining scheduling issue. And cluster the jobs and decide on the static schedule. Then, next thing what we look at, so what we have look that, looked at periodic, we have looked at task which and not necessarily periodic.

(Refer Slide Time: 44:29)

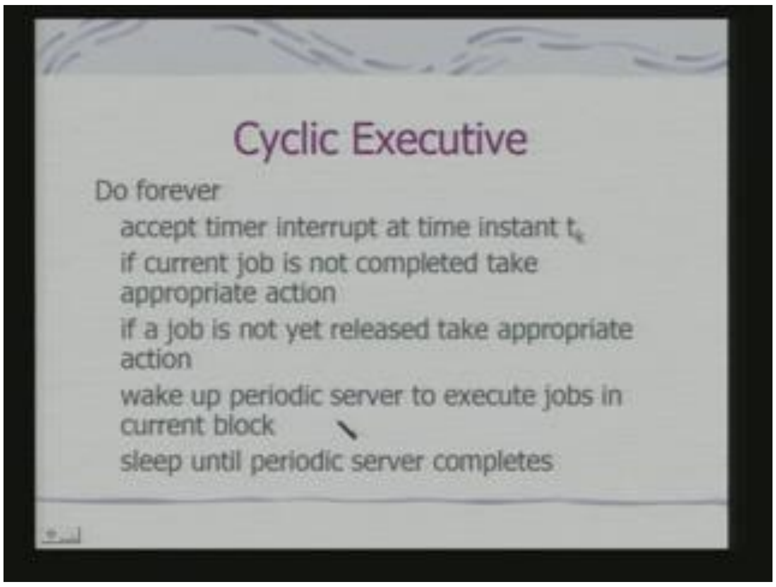


Off-line schedules to handle Mixed tasks

- Let there be n periodic tasks in the system
  - Assume that aperiodic jobs are released at unexpected time instants and join aperiodic job queue
- Timer sends k^{th} interrupt at time t_k
- Scheduler schedules tasks from schedule at timer interrupts
- Time intervals between scheduling points referred to as frames

Now, we shall look at kind of a mixed scenario where there are may be some task, which a periodic, some tasks which are non periodic. So, the situation becomes more complex. So, let there be n periodic tasks in the system and assume that periodic jobs are released, at unexpected time instants and the join a periodic job queue. Timer sends a kth interrupt at time t k. And scheduler, schedule tasks from schedule at time interrupts. And time interval between scheduling points referred to as frames. So, obviously, the scheduler coming into play at a fixed instance and the fixed instances is decided by the timer.

(Refer Slide Time: 45:13)



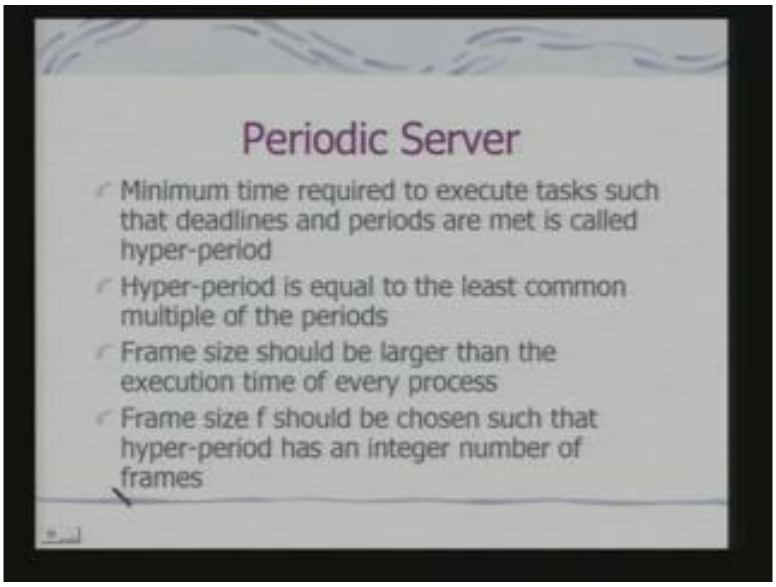
Cyclic Executive

```
Do forever
    accept timer interrupt at time instant $t_k$
    if current job is not completed take
    appropriate action
    if a job is not yet released take appropriate
    action
    wake up periodic server to execute jobs in
    current block
    sleep until periodic server completes
```

So, here what we use a kind of a cyclic executive algorithm. And this is also from the basis of the knowledge of the kind of job mix that you expect to get. So, what you do, you accept timer interrupt at time instant t k. If current job is not completed take appropriate action that means, I need to save its status. So, that can be schedule next if job is not yet released take appropriate action that means, if the job is waiting not utilizes means what, the job will now join the ready queue.

Now, wake up next what you do, you wake up periodic server to execute the jobs in current block. That means, now if there is a some periodic jobs. So, I use a periodic scheduler to execute the periodic jobs. And you sleep the scheduler will sleep until periodic server completes.

(Refer Slide Time: 46:14)



Periodic Server

- Minimum time required to execute tasks such that deadlines and periods are met is called hyper-period
- Hyper-period is equal to the least common multiple of the periods
- Frame size should be larger than the execution time of every process
- Frame size f should be chosen such that hyper-period has an integer number of frames

What does periodic server do, periodic server can schedule the jobs following strictly the right monotonical algorithm. It can follow the right monotonical algorithm. We can follow the EDF algorithm, depending on which algorithm and time to use. But, that can be a static schedule drawn as well. How the static schedule can be drawn in this case? And which becomes very simplified, if you look in to the tasks minimum time required to execute tasks such that deadlines and periods are met is called hyper period.

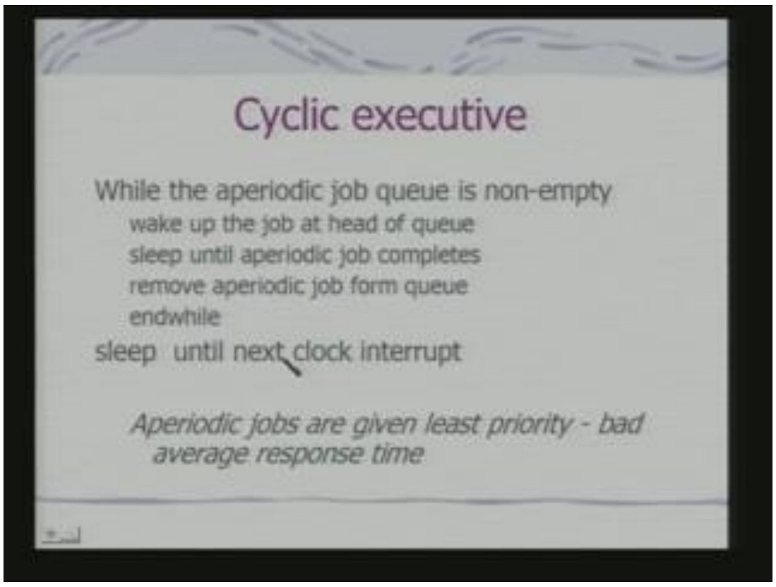
If I look at a set of tasks, such that deadlines and periods are met. If we consider my deadlines and periods coincident, then this becomes strictly a period. What would be the hyper period? That hyper period would be basically what, the least common multiple of the periods, periods of the tasks, periodic tasks which the periodic server is serving. And if we consider an offline scenario that means, you already know the periodicity. We can always compute the hyper period.

And other issue is what, frame size is what, frame size is time interval between the scheduling instance because, the timer is scheduling the scheduler between the time instance. Obviously, if periodic server to work what do you require, you require the frame size should be larger than the execution time of every processes. That means, it should be more time that of max of the execution type of all the processes, which have under consideration.

If it is less than that, you cannot really drop a fix schedule for the periodic server. And frame size should be chosen, such that hyper period as an integer number of frames. Hyper period is the LCM of all the period. So, if your frame size f is what, is included in the hyper period integral number of time. Then what it means, all periodic job can actually scheduled, because, at the frame intervals the scheduler is coming in to play. And it is not coming in to play between, because it is a integral number.

So, depending on the periodicity of the jobs, at each scheduling instant the jobs can be scheduled. And all this computations can be done if you really know, a priory the set of periodic jobs.

(Refer Slide Time: 48:53)



Cyclic executive

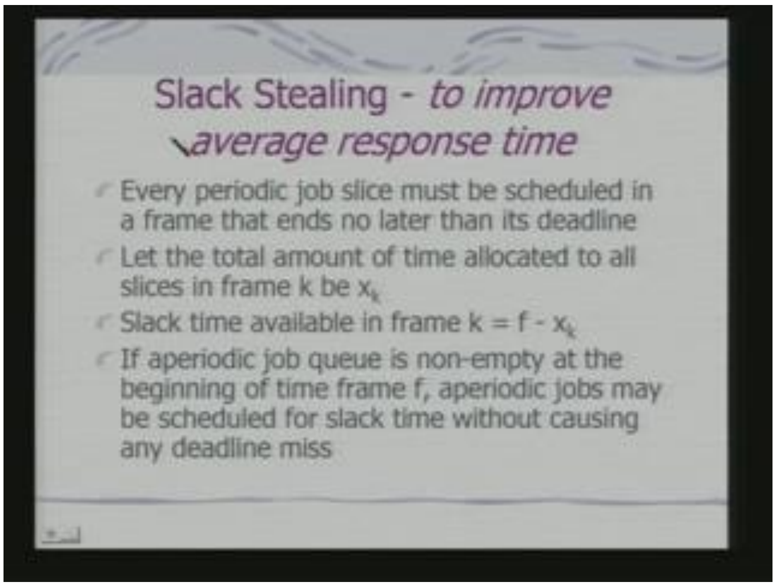
```
While the aperiodic job queue is non-empty
    wake up the job at head of queue
    sleep until aperiodic job completes
    remove aperiodic job form queue
    endwhile
sleep until next clock interrupt
```

Aperiodic jobs are given least priority - bad average response time

Next comes, what happens to the periodic jobs, when the periodic server has released. That, while the periodic job is non empty, wake up the job at the head of queue sleep until a periodic job completes. Remove a periodic job from queue and while and sleep until next clock interrupt. This, become a very crucial kind of a scenario. So, in this case obviously, what we have finding is a periodic jobs, have been given list periodic. This can only be done only when, the periodic server has finish it is job.

So, appear a periodic jobs at given the bad average response time if you remember the definition of the periodic jobs and jobs, which occur occasionally. But, they really do not have deadline as associated with it. And that is the reason why, I can wait the periodic server completes the job, that look at that a periodic jobs. But, can we improve up on the average response time. This lead show what is called slack stealing, to improve average response time.

(Refer Slide Time: 49:54)



Slack Stealing - *to improve average response time*

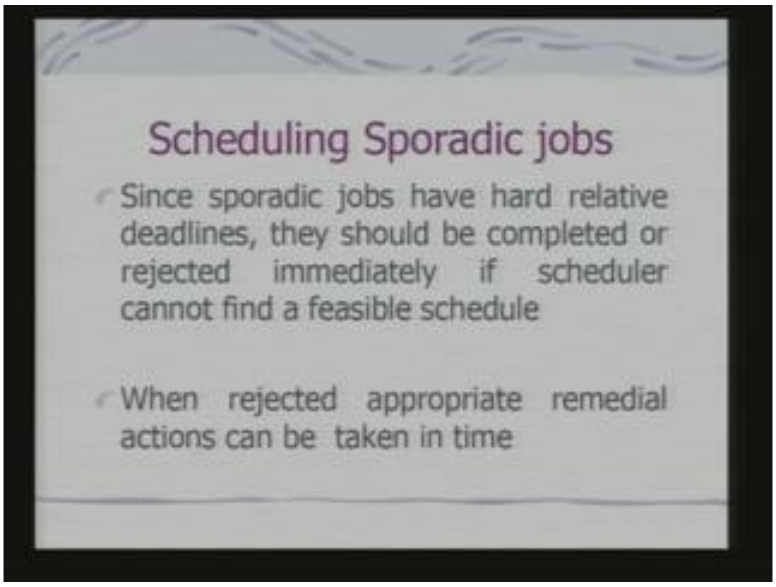
- Every periodic job slice must be scheduled in a frame that ends no later than its deadline
- Let the total amount of time allocated to all slices in frame k be x_k
- Slack time available in frame k = f - x_k
- If aperiodic job queue is non-empty at the beginning of time frame f, aperiodic jobs may be scheduled for slack time without causing any deadline miss

Every periodic job slice must be scheduled in a frame that ends no later than it is deadline that is the basic requirement, we have already looked at from the periodic server. And let us, the total amount of time allocated to all slices in frame k be X k. Now, what would be the slack time available, f minus X k. Depending on the job beaks that is available, depending on the frame time that has been chosen. Now, I have already told you what, the frame time should be chosen in such a way.

That is integral number of frame time in hyper period and hyper period is known a periodic. So, the frame time is so chosen and such that after allocation of the time to the jobs with deadline with the periodicity, I have k slack time. Then each in each frame, I allocate these slack time to the, a periodic jobs. This would improve up on your average response time for the periodic jobs.

So, this is a typical lee handled, when you have a mix of a periodic jobs with deadline with a periodic jobs, which really does not have deadlines. But, the problem becomes more critical sporadic jobs. Sporadic jobs are what, jobs with deadlines they arrive in an a periodic fashion, but they have got strict lines.

(Refer Slide Time: 51:24)

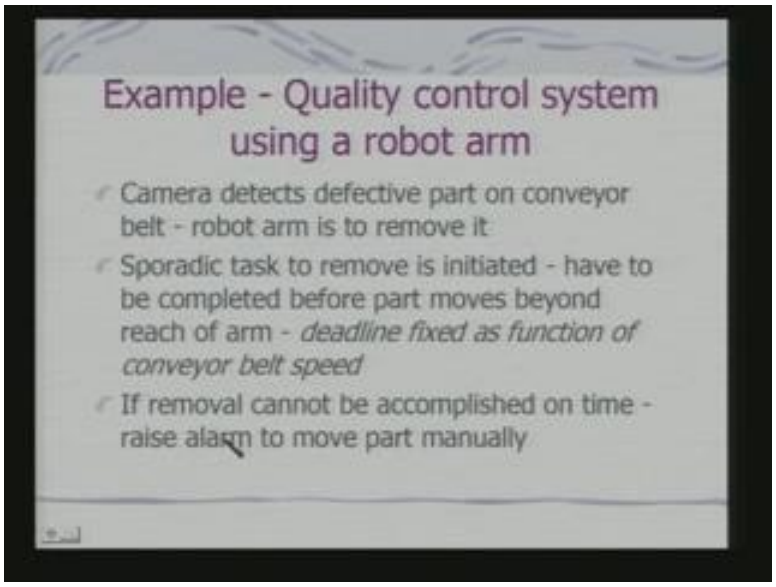


Scheduling Sporadic jobs

- Since sporadic jobs have hard relative deadlines, they should be completed or rejected immediately if scheduler cannot find a feasible schedule
- When rejected appropriate remedial actions can be taken in time

So, what we say, since the sporadic job have hard relative deadlines, they should be completed or rejected immediately. If scheduler cannot find a feasible schedule and when rejected appropriate remedial action can be taken in time.

(Refer Slide Time: 51:41)

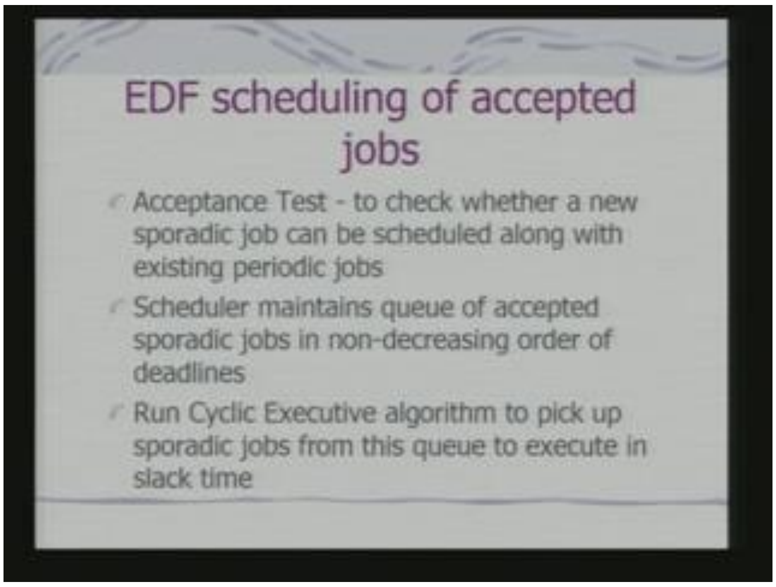


Example - Quality control system using a robot arm

- Camera detects defective part on conveyor belt - robot arm is to remove it
- Sporadic task to remove is initiated - have to be completed before part moves beyond reach of arm - *deadline fixed as function of conveyor belt speed*
- If removal cannot be accomplished on time - raise alarm to move part manually

Let us, look an example is a quality control system using a robot arm. Camera detects defective conveyor belt. And robot arm is to remove it. So, sporadic task, is what to remove is a sporadic task and it is to be removed with in a time interval. Deadline is fixed as a function of conveyor belts and if the removal cannot be accomplished with this sporadic task cannot be scheduled and deadline can be made. It is to be detected. If I can detect it I can reason a lab. That manual operator can remove it. This is how in most of the cases the sporadic scheduling taken care of.

(Refer Slide Time: 52:21)

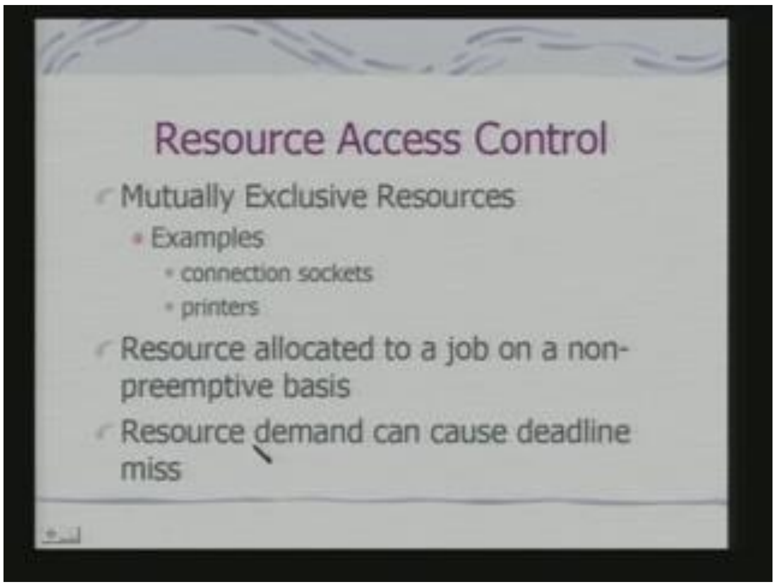


EDF scheduling of accepted jobs

- Acceptance Test - to check whether a new sporadic job can be scheduled along with existing periodic jobs
- Scheduler maintains queue of accepted sporadic jobs in non-decreasing order of deadlines
- Run Cyclic Executive algorithm to pick up sporadic jobs from this queue to execute in slack time

Obviously, for sporadic job scheduling the basic policy to be used will be EDF earliest deadline first. So, use an acceptance test to check, whether a new sporadic job can be scheduled along with existing periodic jobs. Scheduler maintains the queue of accepted sporadic jobs in non decreasing order of deadlines. Run your cyclic executive algorithm to pick up sporadic jobs from the queue in slack time. So, you cannot need the deadline for a given slack time. Then you do not accept the job and you can always calculate the slack time, because, you know the mix of the jobs sporadic.

(Refer Slide Time: 53:03)

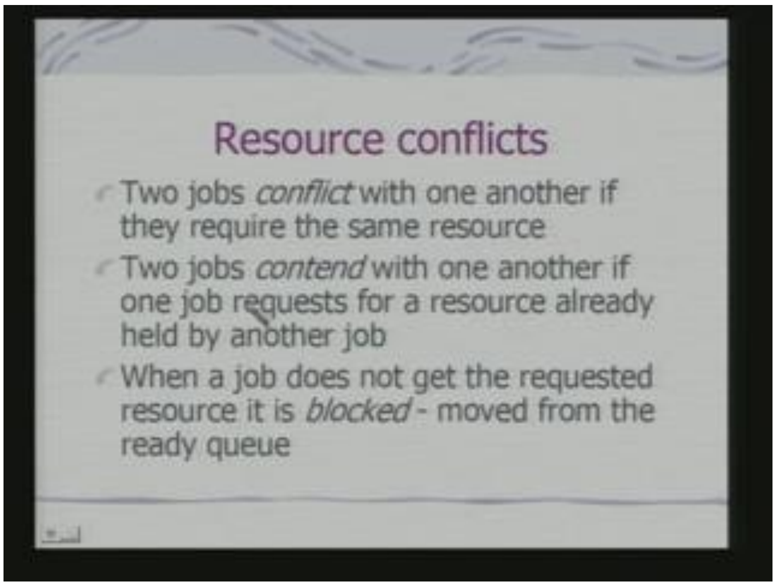


Resource Access Control

- Mutually Exclusive Resources
  - Examples
    - connection sockets
    - printers
- Resource allocated to a job on a non-preemptive basis
- Resource demand can cause deadline miss

Now, these processes do share in many cases resources. And this resource, in many cases can be mutually exclusive resources. Example sockets printers and these resources are allocated to a job on a non preemptive basis. Since, they are allocated on a non preemptive basis another process, which is required in this resource cannot work. If that be the situation, then this resource demand can cause a deadline miss or under that condition, you scheduling policies to be revised.

(Refer Slide Time: 53:44)

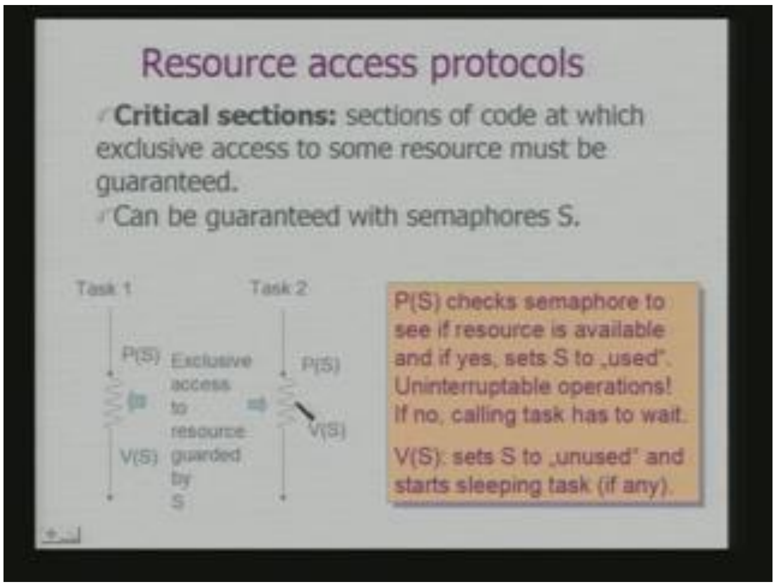


Resource conflicts

- Two jobs *conflict* with one another if they require the same resource
- Two jobs *contend* with one another if one job requests for a resource already held by another job
- When a job does not get the requested resource it is *blocked* - moved from the ready queue

So, we say that two jobs conflict with one another if they require the same resource. Two jobs content with one another if one job request for a resource already held by another job. And when the job does not get the requested resource it is blocked, moved from the ready queue.

(Refer Slide Time: 54:02)



Resource access protocols

- **Critical sections:** sections of code at which exclusive access to some resource must be guaranteed.
- Can be guaranteed with semaphores S.

Task 1
Task 2
P(S)
V(S)
Exclusive access to resource guarded by S
P(S)
V(S)

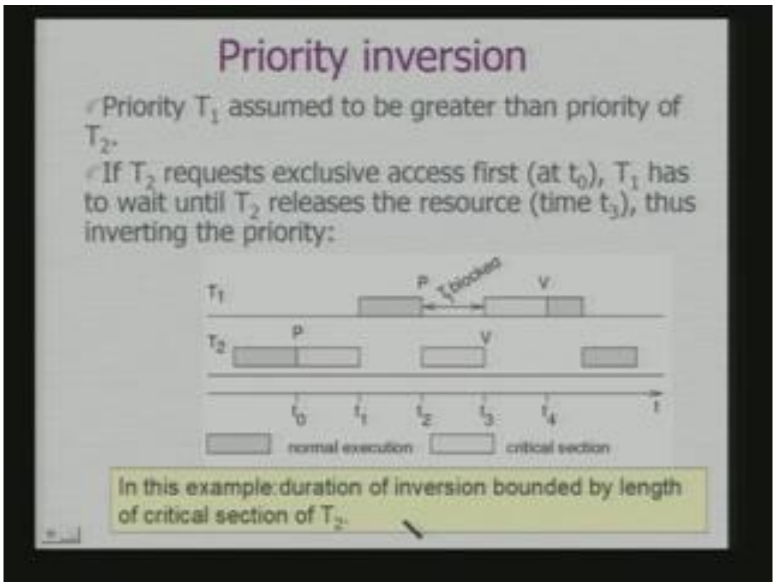
P(S) checks semaphore to see if resource is available and if yes, sets S to „used". Uninterruptable operations! If no, calling task has to wait.

V(S): sets S to „unused" and starts sleeping task (if any).

If it is move from the ready queue, there can be a actually a deadline mix. So, typically when there is a exclusive access, you use semaphore are critical sections for the managing the exclusive access. Here, I am showing P and S the two operators what happens is P S checks semaphore to see, if resource is available if P S sets s to use and once it is inside the task is uninterrupt.

V S sets S to unused, sets S to unused, when you come out of that critical section used V S, And you starts, sleeping task if we any, if your task is sleeping. Because, of these then that will be wake up that is the basic semaphore mechanism.

(Refer Slide Time: 54:48)



Priority inversion

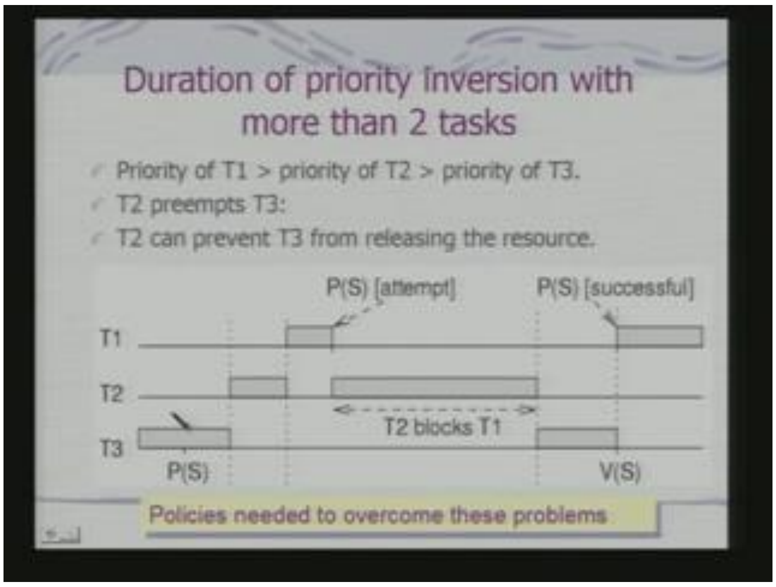
- Priority T_1 assumed to be greater than priority of T_2.
- If T_2 requests exclusive access first (at t_0), T_1 has to wait until T_2 releases the resource (time t_3), thus inverting the priority:

T_1
T_2
P
T_1 blocked
V
P
V
t_0
t_1
t_2
t_3
t_4
t
normal execution
critical section

In this example:duration of inversion bounded by length of critical section of T_2.

Now, what happens under those conditions if you see, if a task is blocked here. Then, the priority actually gets inverted. T 1 has got a greater priority. But, since T 1 is blocked now T 2 works. So, this is what is known as priority inversion. And priority inversion period in this case is bounded by the length of the critical section.

(Refer Slide Time: 55:13)



Duration of priority inversion with more than 2 tasks

- Priority of T1 > priority of T2 > priority of T3.
- T2 preempts T3:
- T2 can prevent T3 from releasing the resource.

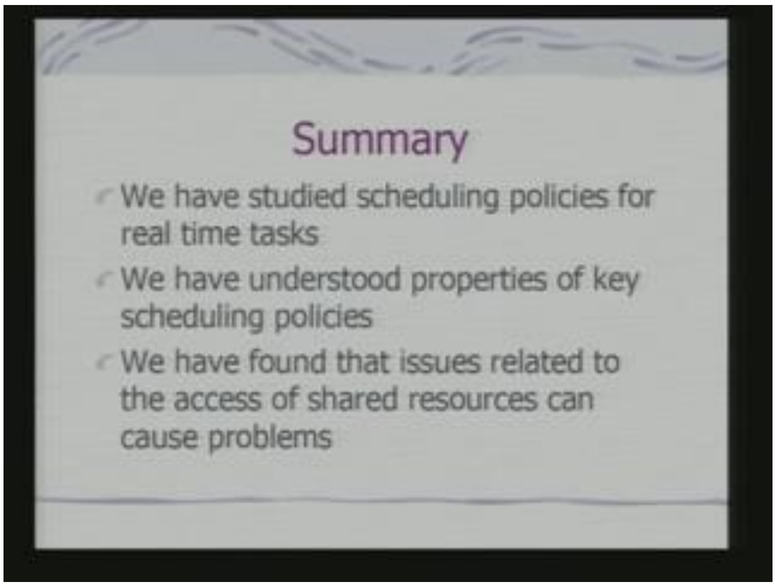
P(S) [attempt]
P(S) [successful]
T1
T2
T2 blocks T1
T3
P(S)
V(S)

Policies needed to overcome these problems

Now, they can be other problems the duration of priority inversion can be more than two tasks because why, because here you got T 2 preempts T 3. And T 2 can preempt the T 3 can from releasing the resource because, T 2 has got a higher priority than that of T 3.

So, in that case, this priority inversion can be of longer priority. So, you need policies to overcome these problems. So, I have got in to the more complex scheduling scenario. And these, is really a complex problem in the context of various and better systems.

(Refer Slide Time: 55:55)



Summary

- We have studied scheduling policies for real time tasks
- We have understood properties of key scheduling policies
- We have found that issues related to the access of shared resources can cause problems

So, what we studied today, there is the scheduling policies for real time tasks. We have understood properties of key scheduling policies. And we have found that issues related to access of shared resources can really cause problems. There would be, what is called a priority inversion. A higher priority task would wait, because some other tasks is actually in execution. In fact, there has been various real life problem, we shall discuss that in the next class in terms of your mass path find the robot of failure occurred. Because, of your priority inversion. So, how this priority inversion problem has to be taken care of we shall discuss in the next class. Any questions?

Student: ((Refer Time: 56:45)) EDF to did not take care of n the lax time and in we just take care of the execution time and Student: ((Refer Time: 57:00)).

See, the point is the question is whether in earliest deadline first scheduling algorithm you take care of the slack time. When we are looking at, earliest deadline first algorithm we really do not considered a slack time. Because, I can really do deadline first scheduling without knowing the computation time. What I need to know is my arrival time and the deadline. And the scheduling is on the basis of absolute deadlines. The job having what, earliest absolute deadline is given the maximum priority.

So, actually I do not take care of the slack time. In the slack time first scheduling, I do this priority assignment on the basis of slack time. So, the requirement there is to know the time required for computation of the associated with the task.