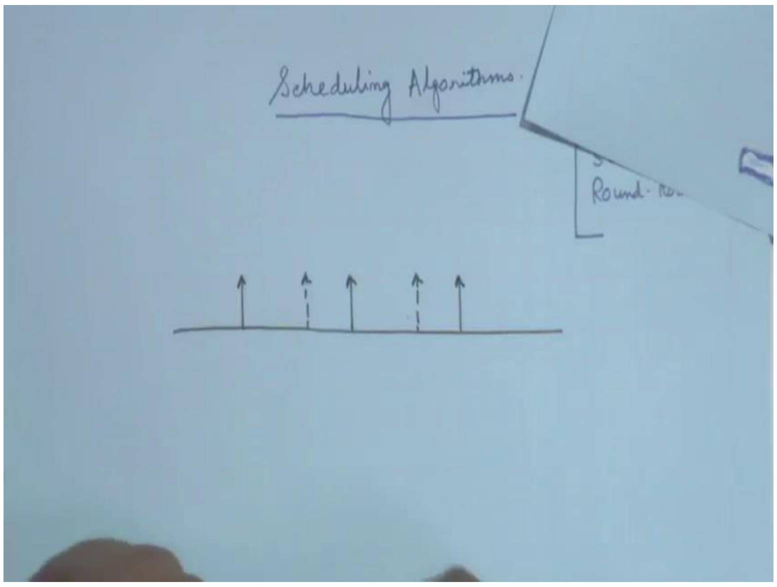
Embedded Systems Design
Prof. Anupam Basu
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 28
RMS Algorithm

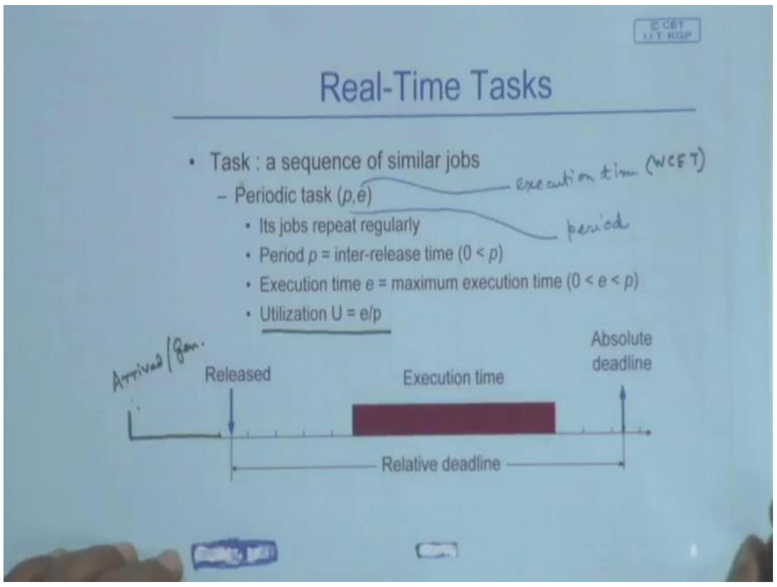
So in the last class, we have seen the distinction between preemptive scheduling and non preemptive scheduling. So, in preemptive scheduling the operating system can preempt a task when a higher priority task arrives. In non preemptive scheduling, once a task has been allocated a resource then that will continue till completion or that task itself requests for some IO. Now we will keep and also we discussed that shared access to resource can block the different tasks, but will come back to that point a little later. Today what we will be doing is we will be looking at the different scheduling algorithms that are used for real time operating systems.

(Refer Slide Time: 01:19)



We are aware of the different scheduling algorithms that are used for normal non embedded system or non real time system type of scenarios. We are aware of the first come first served, shortest job, next round robin etcetera, but under preemptive scenario when we try to schedule within a deadline then we need to be a little bit more careful.

(Refer Slide Time: 02:08)



Real-Time Tasks

- Task : a sequence of similar jobs
  - Periodic task (*p*,*e*) execution time (WCET)
    - Its jobs repeat regularly period
    - Period *p* = inter-release time (0 < *p*)
    - Execution time *e* = maximum execution time (0 < *e* < *p*)
    - Utilization U = e/p

Arrival / Gen.
Released
Execution time
Absolute deadline
Relative deadline

So, if we look at a real time task here for example, here we are showing a real time task, you can see a task is characterized by a period p and an execution time e; e is the execution time. So, this is the execution time and p is the period; period means the delay or the frequency at which a particular job arrives. So, right now we are considering only periodic tasks, we are not considering aperiodic tasks or sporadic tasks. Aperiodic tasks are tasks which arrive, not at fix periods; for example, if this be my timeline and a task arrives at regular delay say 5 units; this is one arrival, this is another arrival, this is another arrival. A task arrives in regular interval then it is a periodic task; aperiodic task, this task could have been aperiodic, if it had some time appeared here; some with the delay 3 may be and sometime appeared here like that.

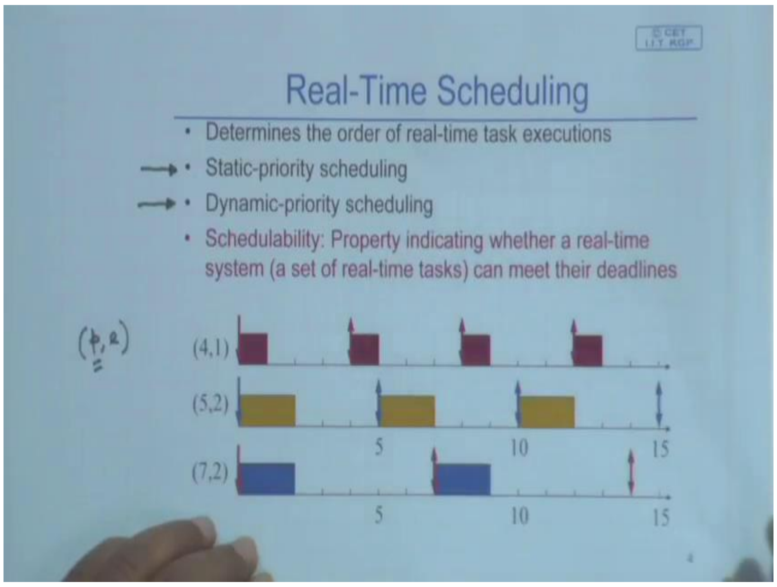
Now, we are concerned with the periodic tasks only for the time being and a periodic task therefore has a period; that is a delay gap after which it appears and it has got an execution time. Now couple of things need to be mentioned about this execution time apriori often we do not know, what will be the exact execution time of a task because that may depend on the arrival of some events or that may also depend on some particular data. Therefore, when we talk of execution time we talk of worst case execution time. Usually we talk of worst case execution time, now since its periodic the jobs will repeat regularly.

Now, the period is the inter release time; now the point is say the job has been released at this point of time and a job comes with a deadline. The deadline is the absolute deadline that come what may; the job must be finished by this time and the worst case execution time of the job has been shown here. So, this is known as the relative deadline, relative deadline means may be that the job has actually been generated here; might be the job has been generated here that is the arrived or generated, but I released it from this point.

Now, in real time scenario that is why this is known as the relative deadline because this is a deadline with respect to the time when the job is being released or the job is being; now these two usually are the same for the case of an embedded system, but in general we can have see that a job has been arriving in the ready queue, but it has been taken to some other queue and all those things. Therefore, with respect to the deadline with respect to the release, so the absolute deadline is given in terms of some time that it must be completed within 5 millisecond; now that 5 millisecond will be computed from this point, the job is that is released.

Now, the execution time is the worst case execution time or the maximum execution time. Now it cannot be more than the period; obviously, in that case there will be problem and we are talking of something called the utilization. Utilization of the processor, see for example during this time; this is a period and during this period e units of time are being utilized for executing this job. Therefore, the utilization is e divided by p, so now we are associating a job with respect to; till now we have not talked about priority, but along with that each job that comes will come as a with some priority. So, with this notion of execution time, period, utilization and the deadline let us move to the issue of real time scheduling.

(Refer Slide Time: 07:43)



Real-Time Scheduling

- Determines the order of real-time task executions
- Static-priority scheduling
- Dynamic-priority scheduling
- Schedulability: Property indicating whether a real-time system (a set of real-time tasks) can meet their deadlines

(p, e)
(4,1)
(5,2)
(7,2)
5
10
15
5
10
15

A real time scheduling or real time scheduler; it will determine the order in which the real time tasks will be performed. There can be two types of scheduling; one is static priority scheduling; that means, the priority of the jobs are decided apriory, I know that this job will have this priority. Now what the priority will be that will vary; how the priority will be associated that will vary from scheduling algorithm to scheduling algorithm. The other thing is dynamic priority scheduling where we change the priority; the priority gets changed, so we will study both this type of scheduling.

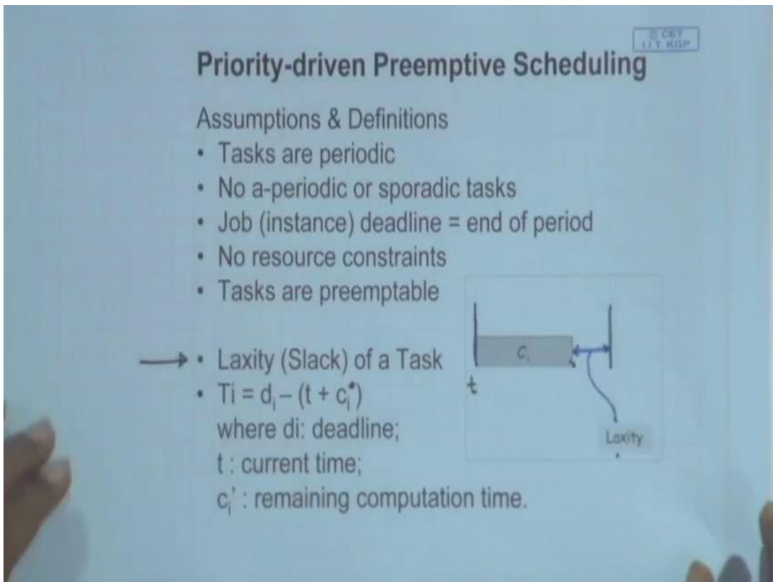
Another very important concept that we must understand is the schedulability. Schedulability means that given a set of tasks, so set of tasks has been have been given to you; now this tasks have come with their execution time and with their deadline. Now given this bag of tasks; are they schedulable? Does there exist a schedule? Whether I can find it out or not is a different question, but does there exist a schedule that can schedule all this tasks, so that the deadlines are met that is known as the schedulability; that is the property indicating whether a real time system or a; that means, a real time system in the set of real time tasks can meet their deadline.

So, here we are showing a 3 task scenario and let us try to understand this diagram. This is task 1, this is task 2 and this is task 3; task 1 as we can see has got the execution time is 1. So, each of these tasks are being denoted as p comma e, where p is the priority and e is the execution time. So, now this priority or rather sorry I am sorry period and the execution time; you see that for this job the period is 4; that means, it is appearing here next after 4 units of time it is appearing again here are my time charge.

So, it is appearing here, then here, then here, then here after 4 units of delay, this one has got an execution time of 2 units and its period is 5, 1, 2 next appearance, next appearance, next appearance and why are these arrows if you can see them; these arrows are bidirectional, the reason is this is also their deadline. The priority and the deadlines of the jobs are this, for the second job it must complete within the period otherwise; obviously, you understand this is the simplest scenery, this is the deadlock the deadline can be less than this, but cannot be more than this because then the second instance of this job will appear, similarly for this the period is 7 and the execution time is 2.

We will come back to this scenario again; we will explain or try out different scheduling algorithms on this set of tasks. Now we first start with now this is the task scenario and we will have to decide on a particular type of scheduling first we will study static priority scheduling.

(Refer Slide Time: 12:06)



Priority-driven Preemptive Scheduling

Assumptions & Definitions

- Tasks are periodic
- No a-periodic or sporadic tasks
- Job (instance) deadline = end of period
- No resource constraints
- Tasks are preemptable

- Laxity (Slack) of a Task
- $T_i = d_i - (t + c_i')$

  where di: deadline;

  t : current time;

  c_i' : remaining computation time.

See here; we have considering static priority scheduling where it is a preemptive scheduling based on the priority. Our assumptions are as before the tasks are periodic, no aperiodic or sporadic tasks are coming that it is coming all (Refer Time: 12:30) and never begin the job deadline is the end of the period that is what we are assuming here that since it is a static priority, we are assuming that the deadline is the end of the period and a very important thing is here we are not considering any resource constraints. In the last lecture, we are talking about mutual exclusion and shared resource we are needing to share a resource because there were not ample number of resources but here we are assuming that no job will be delayed because of non availability of resource; the tasks are preemptable.

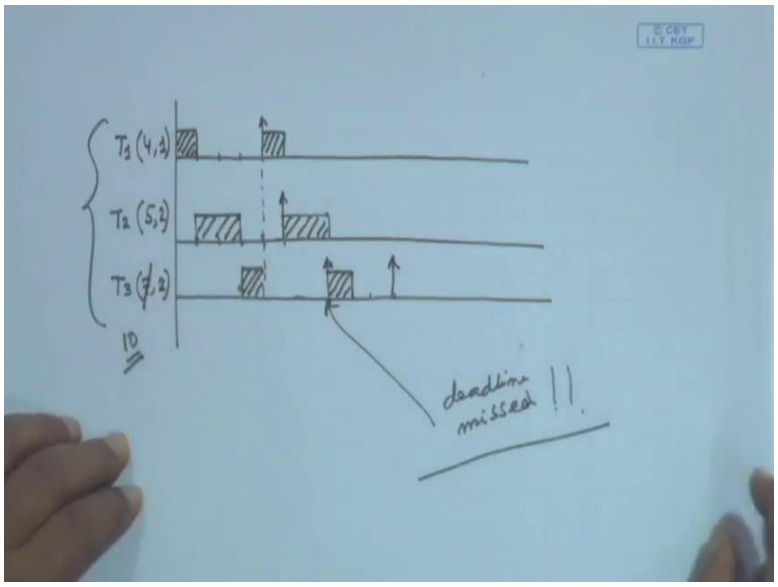
We introduce another term here that is the laxity of the tasks; laxity or slack; it goes by two names laxity or slack of a task. For example, suppose the task arrives here and the task deadline is here and the tasks execution time, it finishes within this time the c i is the execution time, then this is the period that I have got at my disposal this is known as laxity. So, laxity is the T i the time that is if I take the deadline and the current time plus how much time; this is the current time this is t and is taking so much time.

So, t plus c i here is no need of c i time c i and this red line and if I subtract this then I will get the slack, so this term will also come in handy the laxity later. So, with these definitions and ideas let us move to the first scheduling algorithm which is the Rate Monotonic Scheduling algorithm or the R M S. Again we come back to the task that we had seen earlier, now here it is a static priority scheduling, the priorities are known beforehand.

Now what are the priorities? It assigns the priority according to the period and we know that we have assumed that the period is the same as the deadline for the time being. So, a task with shorter period has a higher priority; that means, its deadline is less. So, it will execute select the job with the shortest period, now I would request all of you to note this task specification down and we will see what happens as we proceed with a scheduling.

Now, you can see you just note down 4, 1; 5, 2; 7, 2 from there you can reconstruct this. Now if I look at this point, which one will have the highest priority; the one whose period is the shortest. So, the period is shortest for task T 1; if I name them T 1, T 2, T 3, so therefore I will schedule this one first. So, as I schedule this one; the other two will be delayed although they arrived here, they will be pushed back, so what will be the scenario?

(Refer Slide Time: 16:57)

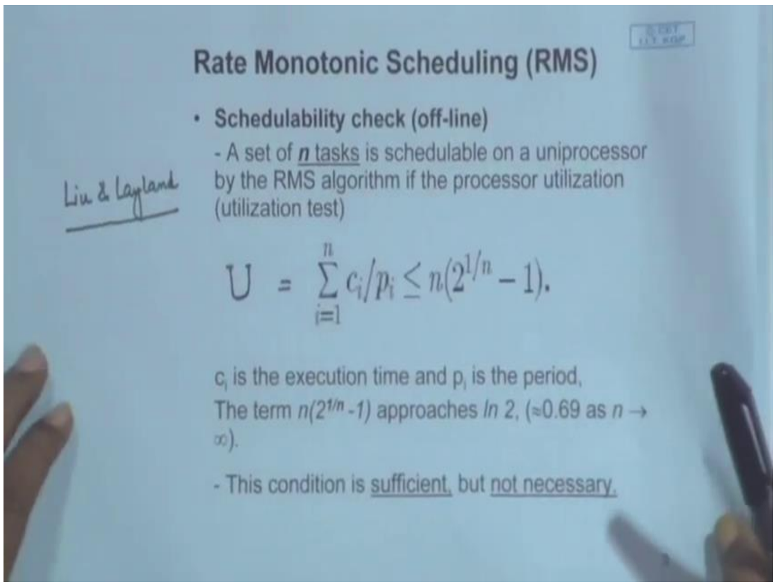


If we do this, so the first task T 1 which was having 4, 1 and whose 1, 2 let me just put the timeline first 4; this is its second arrival time this 1, 2, 3, 4. Now this task has finished here and task T 2 which was having 5, 2 was waiting here. So, again 1, 2, 3, 4, 5 it will again appear here right task 2.

Now, the second task will start from this point onwards because this has; I am assuming by the way a single processor. Therefore, this task will start and its priority is highest, so it will take two units of time and will run here. Now task T 3 which equals 7 period is 7; therefore, 5, 6, 7; now this can be started here. Now so I can start this task here, it will take two units, so I start it here, I go on doing this, but something happens at this point; what happens at this point; task 1 again arrives and according to my priority task 1 has got the higher priority. So, task one will execute here till then till this task one arrived this one has run, but it is supposed to get the other slot; it needed two units; one unit it has got, it will need another unit and right now it cannot take.

So, now task T 1 has finished, but task T 2 has arrived therefore, task T 2 will now go for two units and this task can only come here and; that means what? This task has missed the deadline, so the deadline has been missed. Now, suppose I just change the scenario, I make this period instead of 7; I make this 10. Now what happens? Fine, so this is 2, 1 this was 7, 8, 9, 10; its actual period was here. Therefore, its deadline is also this therefore, now both of them are schedulable. Therefore, the question is how do I know that given a pack of tasks, whether I will be able to schedule them or not? That is something we have to understand. So, we will try to find that out; it has been found is the problem cleared to all of you? This problem is cleared?

(Refer Slide Time: 21:47)


Rate Monotonic Scheduling (RMS)

- **Schedulability check (off-line)**
  - A set of ***n*** <u>tasks</u> is schedulable on a uniprocessor by the RMS algorithm if the processor utilization (utilization test)

Liu & Layland

$$U = \sum_{i=1}^{n} c_i/p_i \leq n(2^{1/n} - 1).$$

c_i is the execution time and p_i is the period,
The term $n(2^{1/n}-1)$ approaches $\ln 2$, (≈ 0.69 as $n \to \infty$).

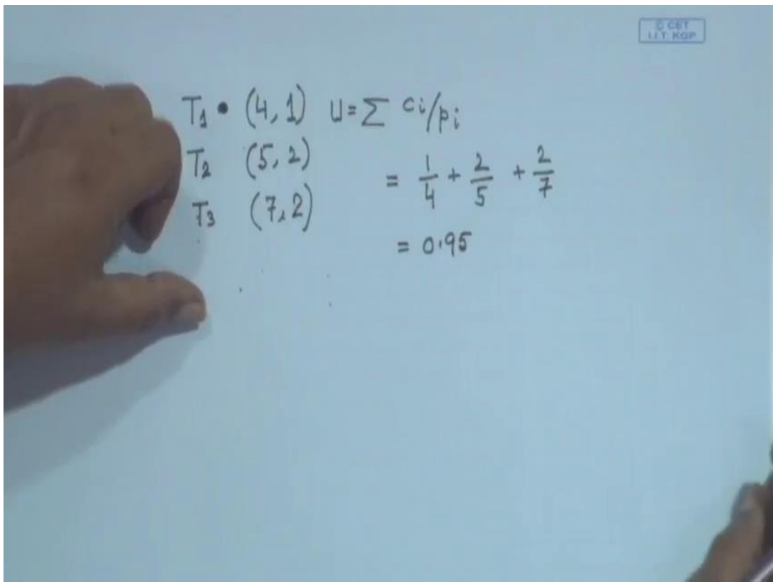
- This condition is sufficient, but not necessary.

So, but with this I do not meets the deadline, so it is schedulable. Now it is said that a set of n tasks is schedulable on a uniprocessor by the rate monotonic scheduling algorithm, if the processor utilization is this. What is this? c i is the execution time of the task of one task p i is the period of that task if and we have seen that c i by p i we had given a name of that; that was utilization.

So, the total utilization for all the tasks, this is a total utilization is less than equal to n times 2 raised 1 by n minus 1. If this condition is satisfied then we can schedule the tasks, I am not going into the derivation of this; this derivation can be found out from the; there is a famous paper by Liu and Layland those of you are interested can look up the paper and it actually counts from the response time analysis and from there we get this we will illustrate the significance of this condition shortly.

Now, this condition is sufficient but not necessary; that means, if this condition is satisfied then certainly you can schedule it; there exists such schedule, but if it is violated then a schedule may exist or may not exist; I do not know beforehand.

(Refer Slide Time: 24:03)



T_1 (4, 1) $U = \sum c_i/p_i$
T_2 (5, 2)
T_3 (7, 2)
$= \frac{1}{4} + \frac{2}{5} + \frac{2}{7}$
$= 0.95$

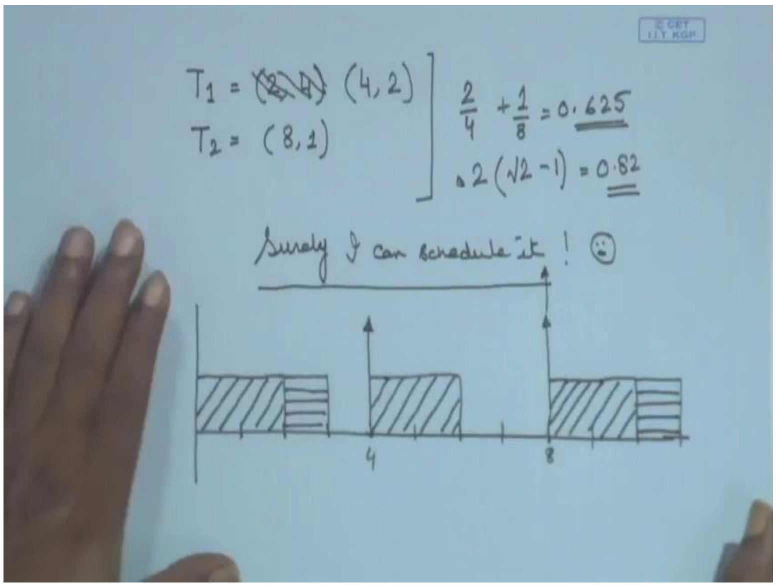
So now let us take a case I will come back to this; let us take a task scenario where I have got a task T 1, whose time is say 4 is the period, 1 is the execution time, T 2 is another task 5, 2 a little different I have done from the earlier one you know and T 3 is 7,2; there is a same thing the earlier task it failed, the task that we had just now done; this scenario this task scenario that failed. We are trying to analyze that with respect to the utilization criteria, now what will be my sigma U; I have got sigma U or U is sigma c i by p i and that will be here 1 by 4 plus 2 by 5 plus 2 by 7, so that is coming to around 0.5 little more than that little less than that; however.

Now, we know that this should be less than equal to n 2 raised 1 by n minus 1. Now for us n is n 2 raised 1 by n minus 1 that is 3 tasks, I have 3 tasks 2 raised 1 by 3 minus 1 that comes to around 3 into 1.26 minus 1 is equal to 0.78. So, you see this condition is not maintained is violated; that means, what does this thing, what is the inside that you get here; the inside that you get here is that if there be 78 more than 78 percent utilization of the CPU, then you cannot schedule more than 3 tasks here; now under this scenario.

Now, you can say for example, if I make it 10, 2; what will this thing become? This part will be modified to 1 by 4; 2 by 5 plus 2 by 10. So, there is coming to 0.25, 0.4, 0.2; this coming to; still this is more, but I could schedule. Therefore, this condition is sufficient but not necessary even with 10, 2; I could do this, but if this is violated I can say I mean if this is satisfied then I am sure that I can schedule that.

So, let us take another example of this, suppose I have got pardon any questions.

(Refer Slide Time: 28:30)



$T_1 = \cancel{(2,4)}\ (4,2)$
$T_2 = (8,1)$
$\frac{2}{4} + \frac{1}{8} = 0.625$
$2(\sqrt{2}-1) = 0.82$
Surely I can schedule it ! ☺
4
8

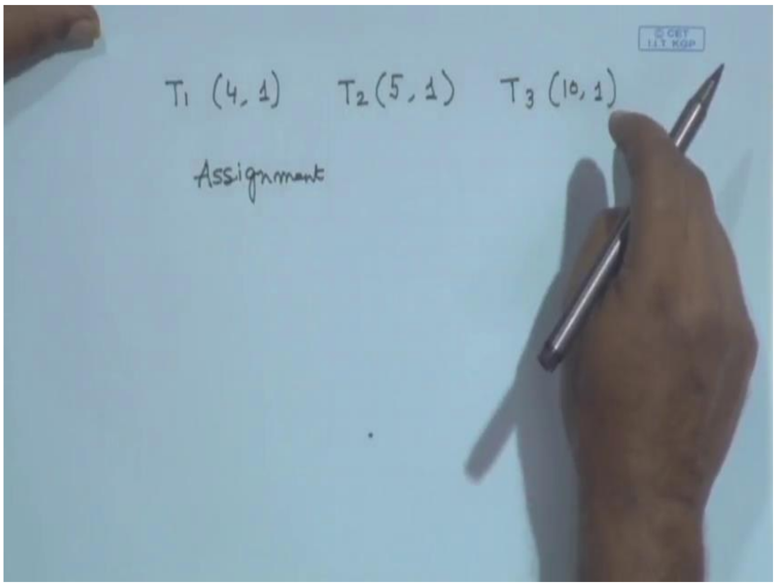
So, let us take another example a task say T 1 which has got is 2, 4; I am sorry writing it in a different way it is 4, 2 and T 2 is another task that is 8, 1. Now let me do the schedulability check here first; what is the schedulability check? Tell me; the left hand side of utilization will be 2 by 4 plus 1 by 8 that is 0.625 whereas, the other side will be 2 times root 2; 2 to the power half minus 1 that will be 0.82 this is less. So, I am sure I will be able to schedule it; I can just say surely I can schedule it.

Therefore, now let me try to schedule it here which one what will be that is the revision of the rate monotonic scheduling algorithm I am trying to do here. So which one has got the higher priority T 1, so T 1 will come in and T 1 will take two execution units say this is 1, this is 2, this is 3, this is 4, 5, 6, 7, 8. So, first T 1 will take place, so T 1 will take 2 units; T 1 has executed. Now T 2; now T 1 comes again here this is the arrival time of T 1 after 4 units again 1, 2, 3, 4 again T 1 comes here and T 2 comes after 8 units so and takes one execution time, so 1, 2, 3, 4, 5, 6, 7, 8.

So, T 2 will also come here at 8 and this is 4, now T 2 is being scheduled here and it will take one unit T 2 has executed here.

Now, during this time nothing is there now at this point which one is the active task T 1? So, T 1 will take again two units 1, 2; now what will happen? Now all the tasks are complete T u, T 2 and T 1 both are complete; next in sense when will T 1 come? T 1 is arriving here and T 2 is also arriving here. So, which one will have priority T 1. So, T 1 two units it takes two units, T 2 arrived here, but T 2 can T 2's deadline is much later. So, T 2 can very well get its place here and no one misses the deadline that is because the spread, the T 2's period was so large therefore, I can certainly schedule it that is the significance of that condition. So, given this now if in the earlier example I had just made a change you just do this.

(Refer Slide Time: 33:09)

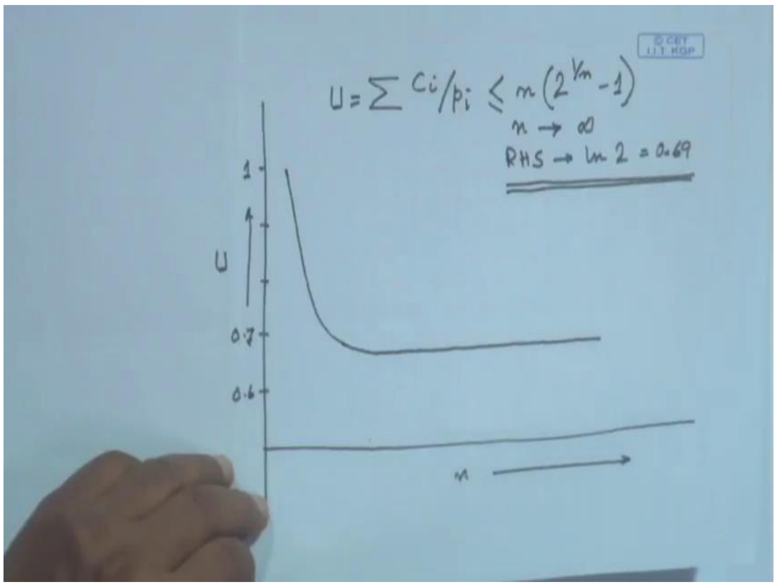


T_1 (4, 1) T_2 (5, 1) T_3 (10, 1)

Assignment

That I am giving again 3 tasks T 1 is 4, 1, T 2 is 5, 1 I am changing it little bit T 3 is 10, 1; it is an assignment I am keeping it an assignment; test the scheduling criteria whether it schedulable or not if it is schedulable find out a schedule.

(Refer Slide Time: 34:00)



Under R M under rate monotonic scheduling; now so what happens is if I just draw a curve, we have got this sigma c i; p i should be less than n 2 raise 1 by n minus 1. Now this as n tends to infinity, this part tends to RHS tends to log of 2 that is around 0.69; that means, so the curve if I have the number of tasks n going on this side and the utilization U is a utilization; U on the other side then if I start from 1 then gradually; it 1 say this is 0.9, 0.8, 0.7, 0.6 like that it comes out.

The curve will come down to something like this; that means, the maximum number of tasks that I can accommodate and schedule is limited by the utilization I can as the utilization comes down the number of tasks; as I increase the number of tasks the utilization comes down and so up to this level. So, if it be below 70 percent we can certainly schedule it; that is under rate monotonic scheduling. Even beyond that it depends on the idiosyncrasies of the tasks, in the following lecture we will continue with another scheduling algorithm that is earliest deadline first.