

Real-Time Systems
Dr. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Module No. # 01
Lecture No. # 07
Cyclic Scheduler

Good morning, let us get started. Today, we will discuss about cyclic schedulers. We had looked at clock driven scheduling, the basic table driven scheduling we had seen. But today we will see that table driven scheduling has actually some difficulties. We mentioned that last time. Does anybody remember, what is the difficulty of a table driven scheduler?

No, you mean the length of the table.

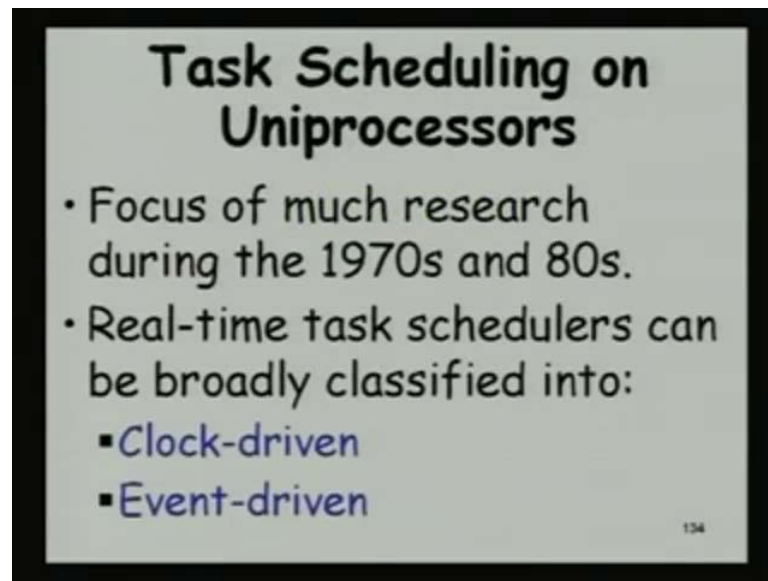
That is not really a big constraint.

New, **new** task is arriving, that is difficult to scheduler earlier.

But that is of all clock driven schedulers. The clock driven schedulers are all static. So, new tasks very difficult to handle, that is true, but we are looking at another clock driven scheduler today - the cyclic scheduler, but the table driven scheduler, what were the difficulties, other than this?

See, one is that setting timers. Each time a task completes, you have to set a timer, **right**. So, let us see how this can be eliminated by cyclic schedulers, which are very popular for this reason; table driven schedulers are rarely used, but cyclic schedulers used extensively. So, let us look at this today.

(Refer Slide Time: 01:49)



So, let us just recapture what we have been discussing last class. We said that see, there are a large number of schedulers that have been in use for real-time systems, and these are basically clock-driven and event-driven; this can be classified into clock-driven and event-driven.

The clock-driven ones are the ones where the scheduling points are marked by clock intervals; is it not? That is what we had said. The event-driven ones are those where the scheduler wakes up when certain events occur; and these events are the starting of a task, I mean a task has arrived or a task has completed. Right.

But in one sense, you can say that see, the task arrival also might depend on a clock, like periodic tasks. So, that way a clock might be involved, but a otherwise this is purely event-driven other than the clock events, right, where as this clock driven ones are solely based on clock events.

Let us proceed, this we had seen last time. And then, we had seen that in a clock-driven scheduling, we need to store a pre computed schedule, and then this schedule is repeated forever, and then to use this scheduler, the designer has to develop a schedule right, but what is the period for which you can needs to design it? We discussed it last time; anybody can remember? See, if we have say n tasks of period p_1, p_2 to p_n .

(Refer Slide Time: 03:06)

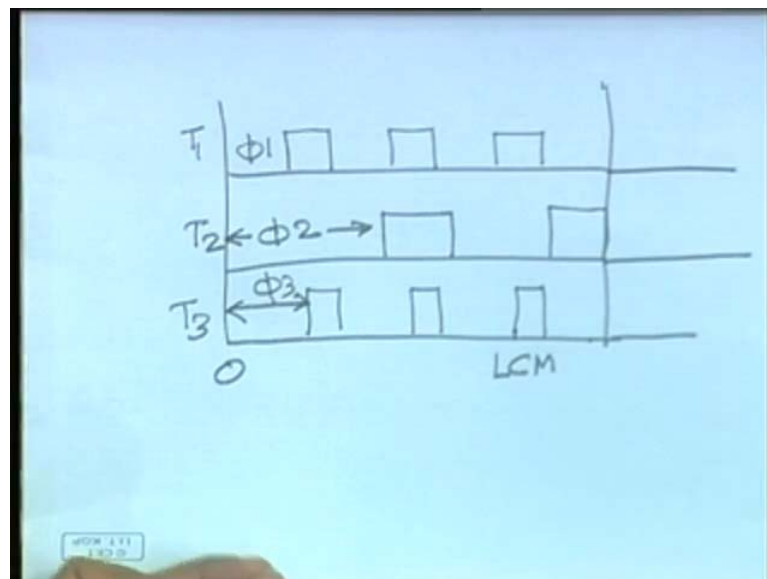
Clock-Driven Scheduling

- For scheduling n periodic tasks:
 - The schedule is stored in a table.
 - Repeated forever.
 - The designer needs to develop a schedule for what period?

135

LCM. So, why LCM? (Conversation not audible Refer Time 0:45) So, we are assuming that the tasks start in phase and for again that similar condition to occur, **right**, we have to look at a multiple of them, that is the common multiple, and we look at that the lowest common multiple, but what if the task do not start in phase? We said that, see when they start in phase, different tasks are there. Let me just draw; so different tasks.

(Refer Slide Time: 04:14)

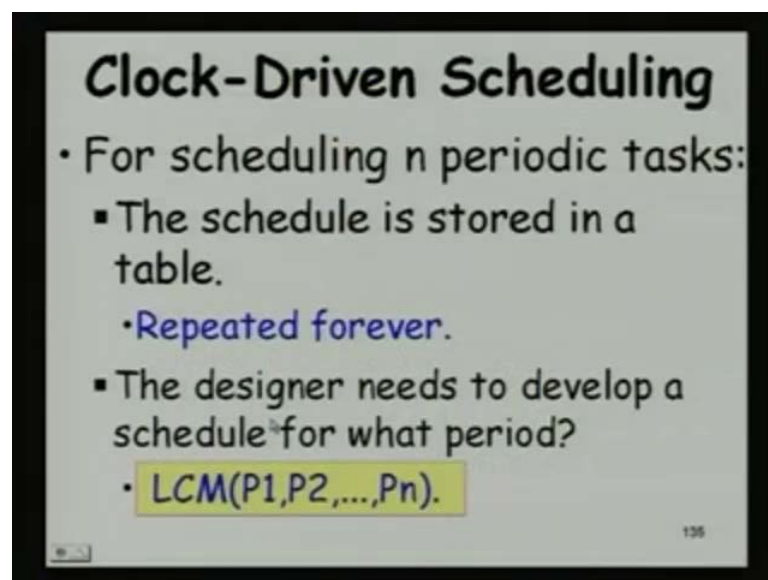


So, if they are starting in phase from 0, then at after LCM p 1, p 2, p 3 they **are all...** again in similar condition will occur, and they will again repeat their occurrence here,

because these are periodic tasks. But, what about there is a phase here. Each one has a different phase. Phi 1, let us say after phi 1, let us say the task T 1 starts and repeats, after phi 2 task T 2 starts and repeats, and the task T 3 starts after phi 3 and repeats. So, will the result continue to hold - the LCM result?

The answer is **yes** and one of your predecessor batches were given this as an exercise; so bonus problem, and some of them had done this actually, they showed it very easily, and the proof that they had given is there in the book that was worked out by them. So, very simple thing; argument was very simple; just have a look. We will not really go into how to prove that, but please look at it. It is very intuitive. They came up with various proofs actually, but the one that was most convincing and the simplest that we have included.

(Refer Slide Time: 06:05)



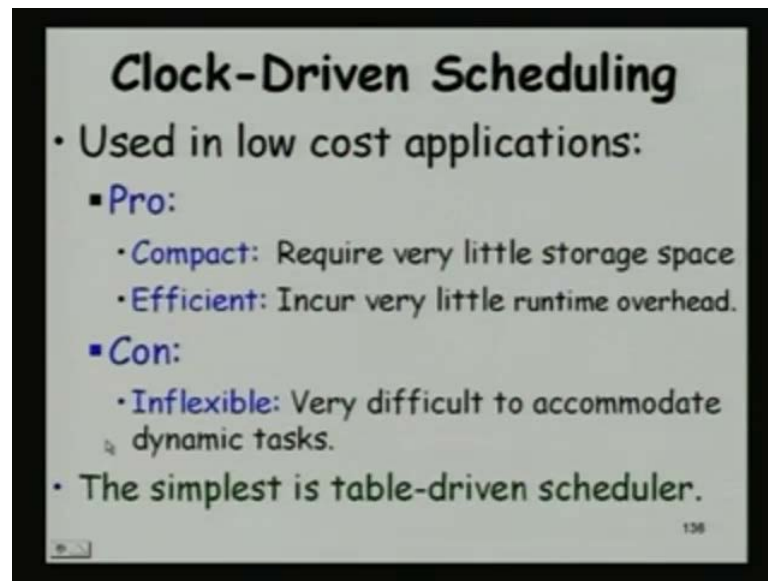
Clock-Driven Scheduling

- For scheduling n periodic tasks:
 - The schedule is stored in a table.
 - Repeated forever.
 - The designer needs to develop a schedule for what period?
 - $LCM(P_1, P_2, \dots, P_n)$.

135

So, the schedule needs to be stored for LCM p_1, p_2, p_n irrespective of whether the tasks start in phase or they are out of phase.

(Refer Slide Time: 06:13)



Clock-Driven Scheduling

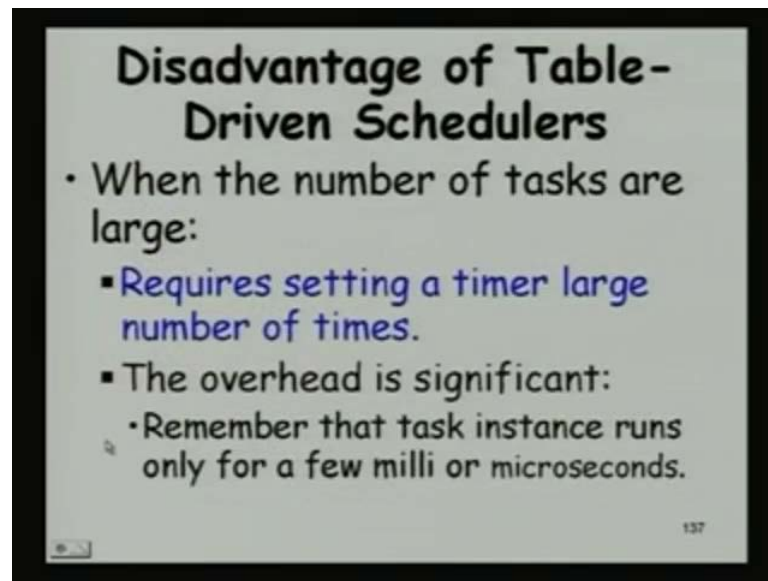
- Used in low cost applications:
 - **Pro:**
 - **Compact:** Require very little storage space
 - **Efficient:** Incur very little runtime overhead.
 - **Con:**
 - **Inflexible:** Very difficult to accommodate dynamic tasks.
- The simplest is table-driven scheduler.

136

We had seen that these are used in low cost applications; and the advantages are that these are very compact. The algorithm is extremely simple, nothing much actually, based on that clock interrupt. Look at the table, select the next task. These are efficient, very little computation needs to be done, each time the scheduler runs.

But if you have to talk about disadvantage, I think some of you had already mentioned that today, that these are cannot handle tasks that are **run** dynamically. And we had seen that the simplest the table-driven scheduler, and as all of you mentioned that see, this table-driven scheduler has the disadvantage that require setting a timer a large number of times, and if a task runs for only a few milliseconds or microseconds.

(Refer Slide Time: 07:00)



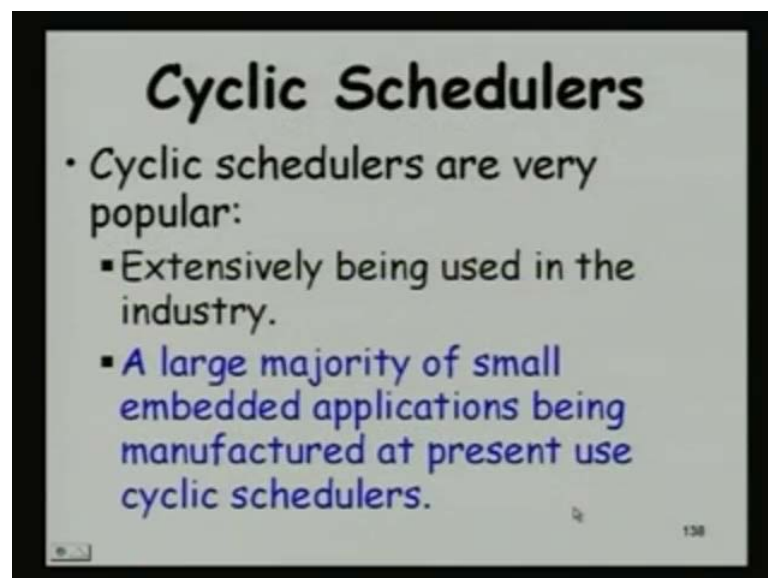
Disadvantage of Table-Driven Schedulers

- When the number of tasks are large:
 - Requires setting a timer large number of times.
 - The overhead is significant:
 - Remember that task instance runs only for a few milli or microseconds.

137

That is the one actually in a real time environment task does not run for minutes or hours; they run for few milliseconds or **advised some micro** few microseconds or milliseconds. And there this setting a timer each time is a significant over head; prevents their use in some applications.

(Refer Slide Time: 07:41)



Cyclic Schedulers

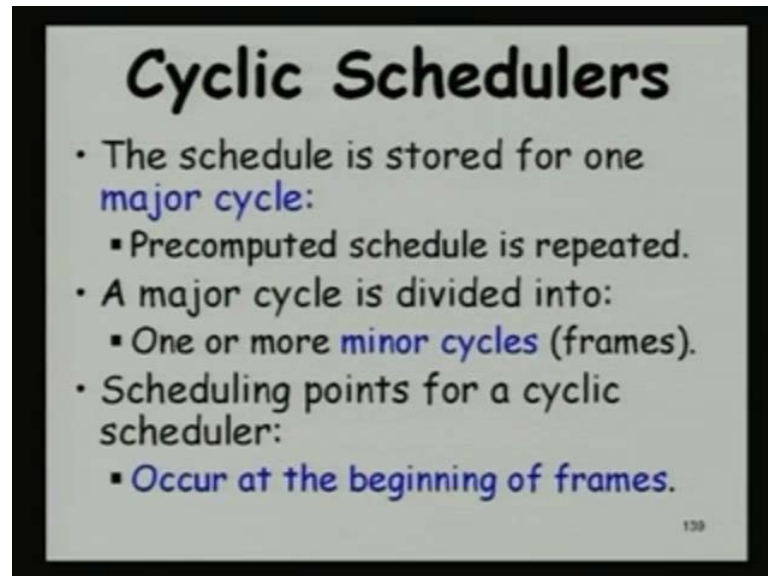
- Cyclic schedulers are very popular:
 - Extensively being used in the industry.
 - A large majority of small embedded applications being manufactured at present use cyclic schedulers.

138

Now, let us look at the cyclic schedulers which are on improvisation of the basic table-driven scheduling. We had that **shaft** coming of a setting a timer large number of times. You know, each time a task starts look up the table, find out when the next one will run,

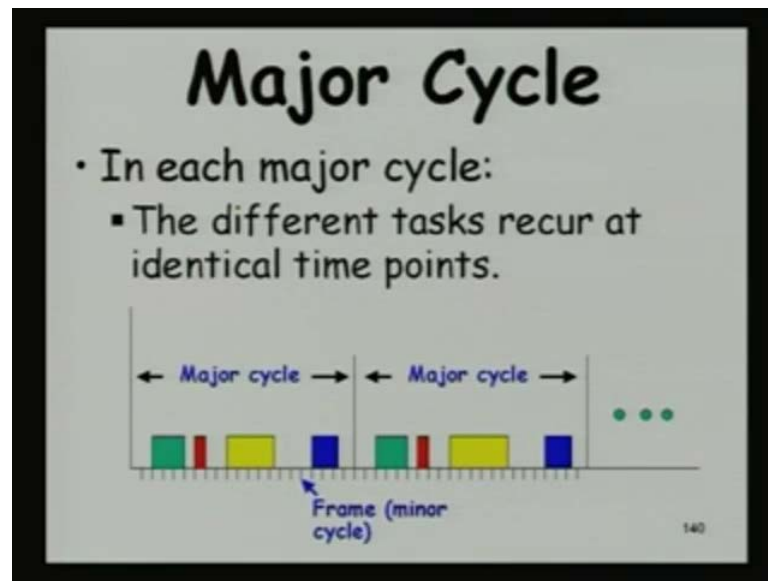
set the timer for that value - is not required. So, let us see how that is achieved and no wonder that these are very popular. If you look at an embedded industry, you will find that large number of their products uses this, and here we have their major cycle, which repeats, and the schedule is required to be computed for the major cycle, and this schedule is repeated.

(Refer Slide Time: 08:20)



But a major cycle is divided into minor cycles or frames. We are not saying right now that whether there will be an integral number of minor cycle, whether the minor cycle will squarely divide the major cycle, we are not really mentioned that, just said that it is divided into some minor cycles. And here the tasks cannot start running at any point in the clock time; whereas, in a table driven scheduling, the tasks can start any time based on when the timer interrupt comes. But here, just see here, that the tasks can run only at the start of a frame or a minor cycle.

(Refer Slide Time: 09:26)



So, the schedule that is computed is repeated perpetually. So, this is the schedule which the designer has statically computed. And see here, when the tasks are all starting at the start of a frame. These small lines that you are seeing, these are the minor cycles or the frames, and just see here that, **that** this task - the green one - is starting at the start of a frame, and also nearing ending at the end of a frame; similar is this one.

But look at the yellow one, starting at a start of a frame, but ending somewhere in the half of a frame. So, it is not necessary that a task has to run for the entire duration and those times will be idle, basically, right. If a task completes early, cannot use a full frame, that much time is idle. But we will see that later, I mean when we look into more details of this; we will see that if it completes early, then we can schedule some of the tasks for which static scheduler is not developed. For example, some are periodic tasks etcetera can be accommodated here, if part of a frame is...

Start only if start of a frame.

That is for the table. The table **stores...** at the start of a frame, we look at a task and start it, but for a periodic task the scheduler what as you will see, it will, if you have to accommodate that. If you will have to accommodate a periodic tasks, it will utilize parts of a frame even, and also all those empty unused frames that are there will be used. That

is an extension of a cyclic scheduler you can say; it is not a basic cyclic scheduler where a periodic and sporadic tasks run.

But this where the are they is similar to such a clock something something...

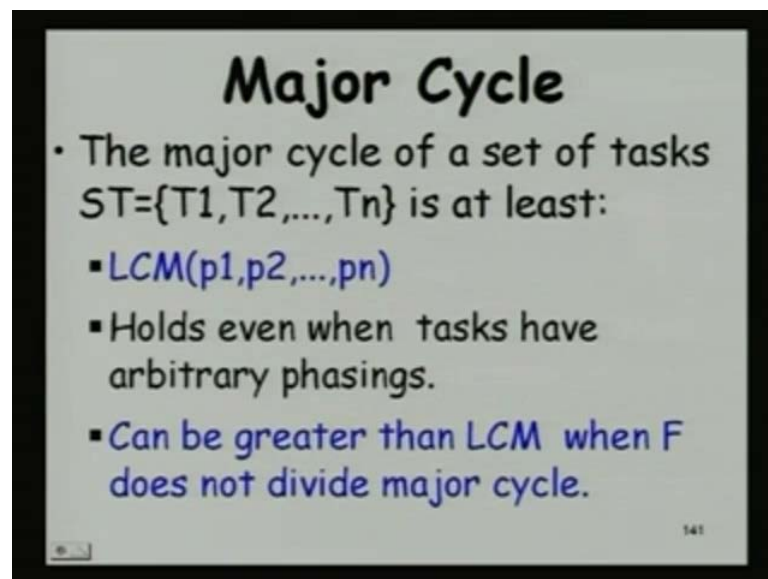
Similar to clock pulses, but these are not really clock pulses; these are periodic, may be 10 clock pulses is equal to 1 minor cycle.

So, one periodic timer is set once; they start here at time 0, a periodic timer is set, let us say for 15 milliseconds or 15 milliseconds is too much, may be let us say some 50 microseconds or something, depending on the application, and then it keeps on giving this interrupts every 50 microseconds. Is that ok?

Smaller ones are that...

Smaller ones are the frames, which are obtained through interrupts from a periodic timer.

(Refer Slide Time: 12:20)



Major Cycle

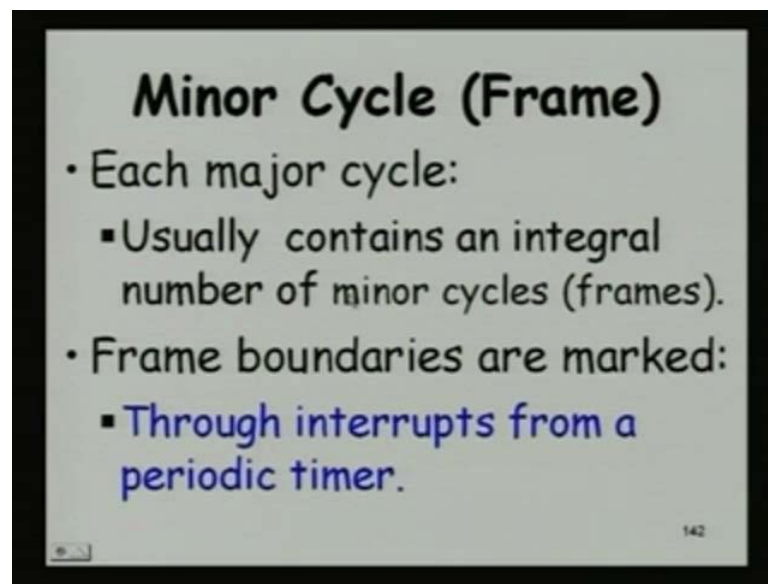
- The major cycle of a set of tasks $ST=\{T_1, T_2, \dots, T_n\}$ is at least:
 - $LCM(p_1, p_2, \dots, p_n)$
 - Holds even when tasks have arbitrary phasings.
 - Can be greater than LCM when F does not divide major cycle.

141

So, the major cycle, let us see what is the length we need to store; let us say we have n tasks, constituting the tasks set, then again we need to store for p_1, p_2 up to p_n and as we were discussing, this holds even when the tasks have arbitrary phrasings. But, the thing is that, if the frame there is not an integral number of frames in a major cycle; that is, F does not squarely divide the major cycle, then we will see that this $LCM p_1, p_2, p$

n is not enough; we need to store for longer duration. So, maybe I can give you that as an exercise, that what if, the frames do not end, **I mean** they do not integrally divide. So, what will be the length of the schedule that needs to be stored? As a bonus problem, **right**; it is not **it is not** there in any book or anywhere reported. So, please try out, what if the frame size does not squarely divide the major cycle. What is the duration of the schedule that needs to be stored?

(Refer Slide Time: 13:42)



As we will see that to reduce the schedule length, we will impose that there is an integral number of frames in a major cycle; and the frames are marked by interrupts, from a periodic timer.

So, bonus marks we will just keep on giving as we proceed, the ones there, where you have to think and are not really there in the book or any reported result, you would not quickly find through a Google search or something. So, this, those who are able to do, please submit it, and we will, as I announced earlier, you will have special marks **right**, but it is not mandatory that all have to submit etcetera; no, if you are able to think and submit please submit, but it has to be your original **right**.

(Refer Slide Time: 14:47)

A Typical Schedule

Frame	Task
F1	T2
F2	T3
F3	T2

So let us proceed; this looks like a schedule. This is a typical schedule, where these are the frames F1 F2 F3 etcetera. And a task T2 runs in F1, T3 runs, and it is not necessary that all frames need to be utilized by some tasks, there are usually many frames will be idle.

(Refer Slide Time: 15:11)

- ### Minor Cycle (Frame)
- Each task is assigned to run in one or more frames.
 - Frame size (F) is an important design parameter while using cyclic scheduler.
 - A selected frame size has to satisfy a few constraints.

And a task can run in one or more frames, and as we will see in the subsequent discussion today, that the frame size is one of the important parameters that a programmer has to design; and then, based on the frame size found, then he will have to

set to the periodic timer, which will give the... determine the frames. Now, the frame size has to satisfy a few constraints.

Let us look at the constraints.

Sir, is it necessary that our task should complete within a major cycle?

So, the question is that - whether the task will complete in a within one major cycle. So, what about anybody has answer to his question. Let us hear your answer; then I will give my answer.

Yes.

Why, why is that? See you are saying that it needs to complete within a...

The task maybe comprises of some minor frame and all the minor frames will make one major frame. So, any task has to...

No, you are saying that a task is having runs on multiple frames and...

And so, and that has to make within one major frame.

Not really very convincing.

Because it is the LCM of all the...

Exactly, here is the answer. That see, the major frame, sorry, the major cycle is defined as the LCM of the task periods, right. So, anyway it will be no much larger it is a multiple of that.

Sir, one is of 2 and the other is of 4; that the n period of 1 p 1 and 2 n, that n period of T2 is 4 the LCM is 4. Then, the main cycle 4, but they both are adding to 6.

No, that is ok. I mean, we are not saying that whether they are schedulable or not. We are saying see, it is possible that we find a set of tasks, which are not, they cannot run, they are not schedulable, but whatever saying is that, if they run, the period for which the schedule needs to be stored is LCM of their periods.

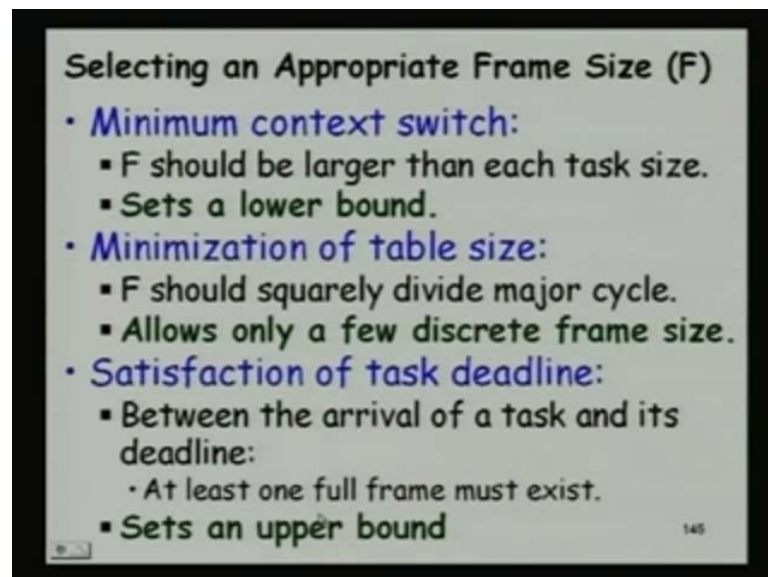
There is no question of preemption there.

Nothing, it is just these are... See preemption comes when you talk of an event-driven one **right**; here we have stored the schedule, but it might so happen that for schedulability, a designer might say that see, a task T1 runs, and as soon as T2 comes it completes, and then again T1 is taken up.

So, T1 is broken up into two frames possibly. But notion of a preemption comes in an event-driven scheduler, where certain event comes, **which has** running task is preempted. So that a concept is not here; it is the pre-computed schedule which will be kept on running. Let us proceed.

Let us see, how a frame size will be determined, what are the constraints in the frame size. One is we have to minimize the context switch. So, to minimize the context switch we need to make the task run in 1 frame, as far as possible, because if it is scheduled in different frames, then obviously, the context switch overhead will come.

(Refer Slide Time: 18:48)

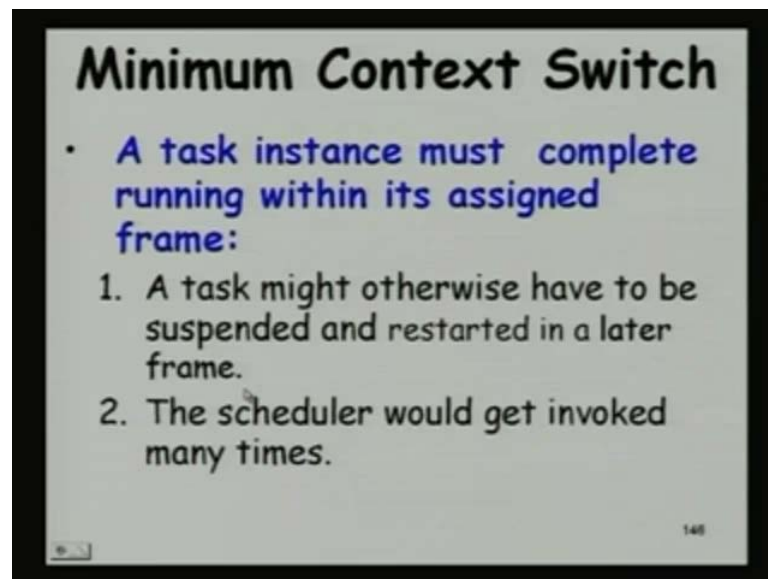


So, to satisfy this constraint of a minimum context switch, the frame size will be larger than the execution time of a task; is it not? Because if the frame size is less than execution time of a task, we will have to run it in multiple frames, and this will set a lower bound, and the size of the frame.

The other constraint is to minimize the table size and this constraint requires that the frame size should squarely divide the major cycle; and this constraint, the implication is that we can use or try out only a few discrete frame sizes. For example, if you have fixed the major cycle to be let us say 20, then we can try out 2, 4, 5, 10, etcetera. So, there has to be a multiple.

The other constraint that needs to be satisfied concerns satisfaction of task deadline. When a task arrives, it has supplanted line, and then, the task must complete by that deadline. And, the constraint is that at least one full frame must exist between the tasks arrival, and its completion **right, sorry**, its deadline. The tasks arrival **and** deadline there must exist a full frame, because our constraint is that the task has to start exactly at the frame boundary. So, if there is a not a full frame, you cannot really run that task; is it not? So, this will set an upper bound and the frame size. If the frame size is more than something, then this constraint - a full frame to exist between a task arrival and its deadline - may not be satisfied.

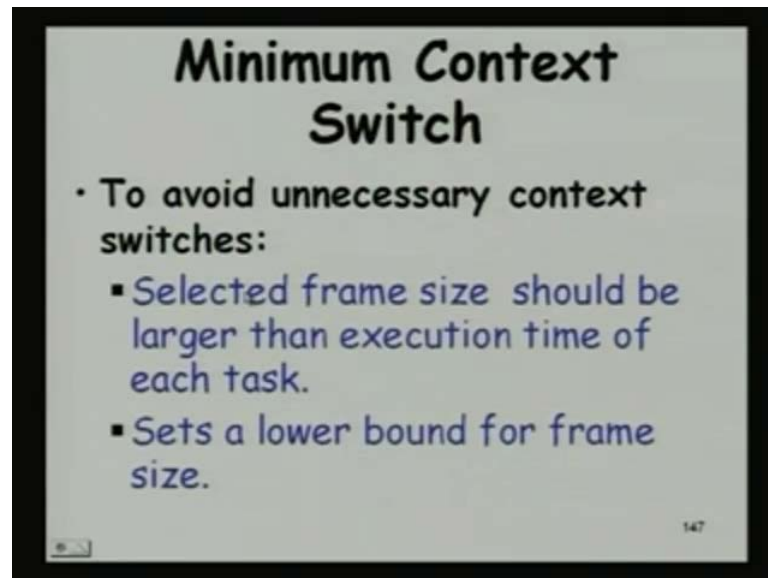
(Refer Slide Time: 20:48)



So, let us look at these three constraints that we identified in more detail. The first one we discussed is minimum context switch, where task instance would preferably complete within its assigned frame. Otherwise, we will have unnecessary run time overhead, which we do not need. So, a task would otherwise, if it does not complete, it might have

to restart at a later frame, and also, not only that, the scheduler will get invoked many times.

(Refer Slide Time: 21:27)



To avoid unnecessary context switches, the frame size will be larger than the execution time of each task **right**. And this will also minimize the number of times the scheduler runs; is it not? If the frame size is too small, let us say one clock cycle is one frame. So, every clock cycle, the scheduler comes up and runs. It is not really very practicable situation. So, let us look at the other constraints. **So...**

Sir, it should be larger is the level of each task.

I mean all tasks together; let us say we have e_1 , e_2 , e_3 or let us say 2, 5 and 10 are the execution time, in microseconds Let us say; 2, 5 and 10, three tasks, and then, we can say that the frame size had to be at least 10. If it is less than 10, then see, one task will have to be split into 2 frames.

Sir, but in the previous another constraint ... if a new task arrives.

Yes, it is dead line yes, **yes, yes, yes**.

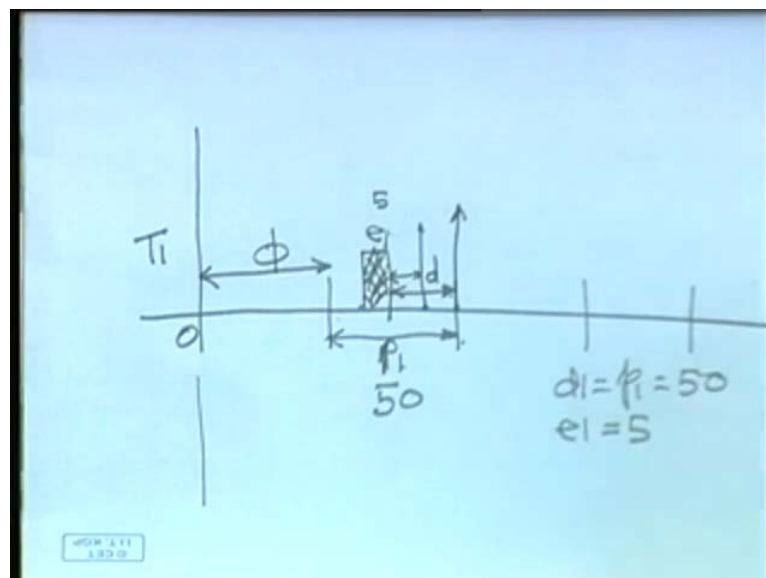
So, there should be an optimum level of...

Exactly, so that is what it is the designer needs to do - find the optimum level. See, we have a lower bound and also we have upper bound. We have to find somewhere the correct frame size. We will try out.

Sir, that is not, I mean larger than the execution time of each task.

Yes, it will be larger than execution time of each task. See a task, once it arrives, it is taken up for running after sometime, is it not? May not be immediately, and then, it has a deadline by which it needs to be finished; we will see the typical periodic tasks, that we will see.

(Refer Slide Time: 23:25)



See, a task T 1 may be a handling a temperature sensor event. So, the T 1 task starts after some phase, and after that it arrives, keeps on arriving periodically. And each time it arrives, it is taken up for running after sometime, usually we will see. And then, it has to complete by a deadline and the deadline typically is the start of the next task.

It may be different, it may be that it has to complete within, let us say 10 milliseconds, and the period is let us say at 50 milliseconds, the task is repeating with 50 milliseconds frame **sorry** cycle, but the deadline is 10 milliseconds from here; it is possible.

But typically, we will see that all tasks that we normally come across, we will have the deadline in same as the start of the next task. So, this is the deadline we are talking of.

This is the execution time and this is the period p_1 ; task T_1 has period p_1 , e_1 is the execution time and d is the deadline. For example, p_1 might be 50, the execution time might be 5 and the deadline is starting from this point is 50. So, d_1 equal to p_1 is equal to 50 and e_1 is equal to 5. Does that appear ok? So, we will consider all tasks like this.

(Question not audible. Refer Time: 25:20)

No, this is just this task characteristic. There will be many tasks like this T_1 , T_2 and so on. Now, our job is to run that using a cyclic scheduler and the cyclic scheduler will have to design the frame, the major cycle, all those we will have to design right. Let us, as we proceed, we will see how to design the major cycle and frames. So, we will let us continue.

So, the first constraint that we are discussing is to set the frame size by considering the minimization of context switch.

Sir...

Yes, please.

Sir, sir here we are (()) to avoid context switches will this frame size should be larger than the existing task time of each task.

Each task exactly.

The set of task that we have...

Yes.

There is a... So, there most of the tasks are very less time consuming.

Yes.

Only one task is there which is it takes lot of time.

Yes. That is an interesting question. He says that see, every task runs within, let us say some 10 millisecond. There is one task which takes let us say 100 milliseconds. So, do we run really design the frame to be 100 milliseconds?

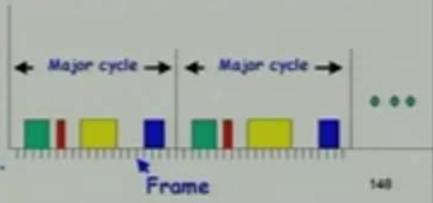
In that case, now all other tasks are wasting that much time, where you know, each frame only one task can run, and all of them are taking only 10 or so, 5 or so and 90 is wasted for every task. Yes, that is a question we will address in the course of our discussion; very important question. And we will see how such cases can be handled. We have one task which runs, takes too much time to run, and then the other task, if we fix according to this rule - that the execution time has to be **sorry** the frame size has to be at least as big as the largest execution time - then we will end up in lot of wastage of the processor time and task set may not be schedulable. So, we will **we will** handle that. We will **we will** discuss about it. Let us proceed.

So, the minimization of context switch will set a lower bound of the frame size, and if we make the frame size any smaller, then the context switches will increase, and also, the scheduler will run many more times, making the execution inefficient.

(Refer Slide Time: 27:58)

Minimization of Table Size

- Unless the minor cycle squarely divides the major cycle:
 - Storing schedule for one major cycle would not be sufficient.
 - Schedules in the major cycle would not repeat:
 - This would make the size of the table large.



Yes, please.

Sir, when does the context switch occur **within apparent** a big frame or at the end of every task?

No, let us listen to the question; you are saying that when does the context switch occur.

Yes sir.

See a context switch, as in a basic operating system, will occur. Let me just ask this question - what is a context? Then we will talk about context switch; can anybody tell what is the context of a task?

(Conversation not audible. Refer Time: 28:27)

What do you mean by?

Yes, exactly. The registers, the cache, the memory it uses; memory etcetera, they are not really context; the context is basically the registers.

The state of the processor.

Right, the state of the processor defines the context. So, when a task is running, and we need to start a new task, the context of the new task has to be loaded, and if the current task has not completed, its context must be saved **right**, when it will be loaded, when it is restarted earlier **sorry** restarted at a later time. Is that ok?

Sir, if the frame size is even lesser than the execution time of the smallest task.

Yes.

Then it should not cause the context...

No, you see if it continues to run in the adjacent frame, then it need not be a context switch. But what if, it runs in multiple frames and in between other tasks around, then there will be context switch.

Yes, not preemption other tasks are scheduled to run. See T 1 runs, then T 2 runs and T 1 contexts we have saved and T 2 runs after sometime **right**, there will be a context switch. Is that ok? Let us proceed.

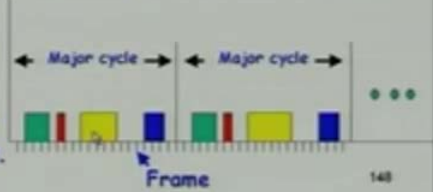
So, the second constraint is minimization of table size. And had we would said that see, unless the frame size squarely divides the major cycle, then we cannot really store that schedule for LCM $p_1 p_2 p_n$, it will be much larger. Just look at this, that see here, between, here the frames are squarely dividing **right**. The frame has started here, the first frame and the last frame has ended here. But in case, let us say, half of the frame is here

right. So in that case, the green one cannot start exactly at this position right. It will be shifted, and all these will get shifted, and in subsequent major cycle, they will keep on shifting until we have lot of free cycles here right. So, it will not, it cannot really repeat the schedule.

(Refer Slide Time: 30:54)

Minimization of Table Size

- Unless the minor cycle squarely divides the major cycle:
 - Storing schedule for one major cycle would not be sufficient.
 - Schedules in the major cycle would not repeat:
 - This would make the size of the table large.



148

Sir,

Yes.

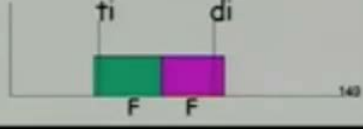
In this slide, you have said that each frame should be... we have a size that is a minimum of all task size...So, we cannot see any...

So, the first constraint that we mentioned - that the frame size should be at least as big as the task execution size; here we have not drawn it like that, excepting I think the red one which is running here. So, this not a realistic figures, to just to explain this concept.

(Refer Slide Time: 31:23)

Satisfaction of Task Deadline

- Between the arrival of a task and its deadline:
 - At least one full frame must exist.
- If there is not even a single frame:
 - The task would miss its deadline,
 - By the time it could be taken up for scheduling, the deadline could be imminent.

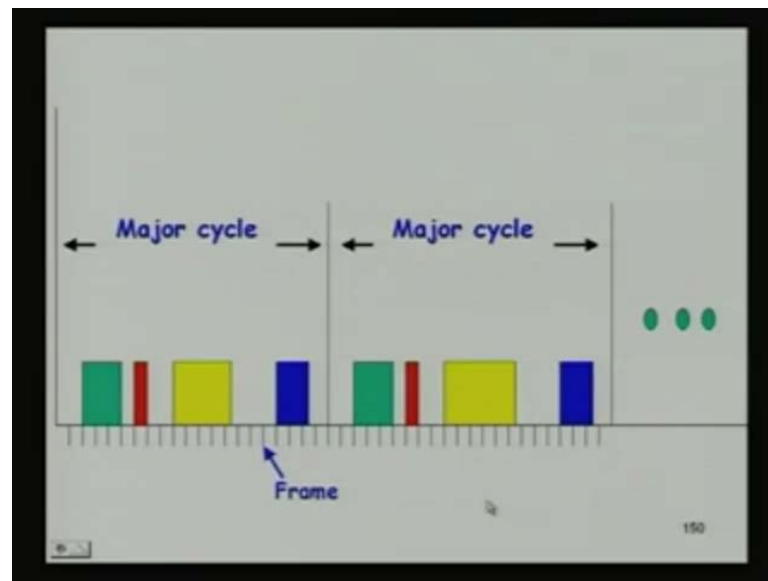


149

Now, let us look at the third constraint; that is, satisfaction of task deadline. So, let us say this t_i arrives that **this...** t_i is let us say a task name and it arrives at this point. Now, the frame has already started here. The green one is one frame, and the frame has already started, it cannot run here, and if you want to run it here, the task t_i , its deadline cannot be met **right**.

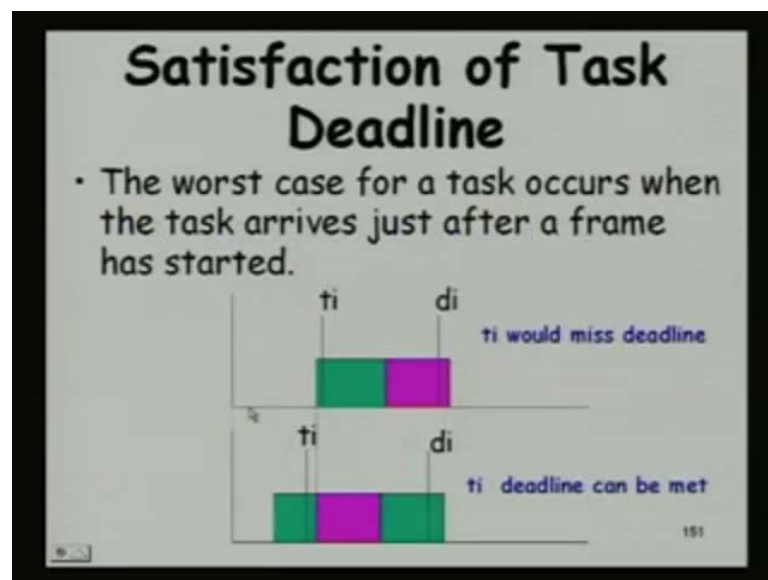
So, this is the problem we will face, if we do not even have a full frame between the tasks starting and tasks and the deadline **right**. In this case, it will surely miss its deadline; you cannot do nothing about it.

(Refer Slide Time: 32:33)



So, this will set an upper bound on the frame size, you cannot have large frames. It will have to have frames, which are... I think this source that.. if a... if a frame is size is too large, then we cannot really meet the deadline; let us not spend time on that.

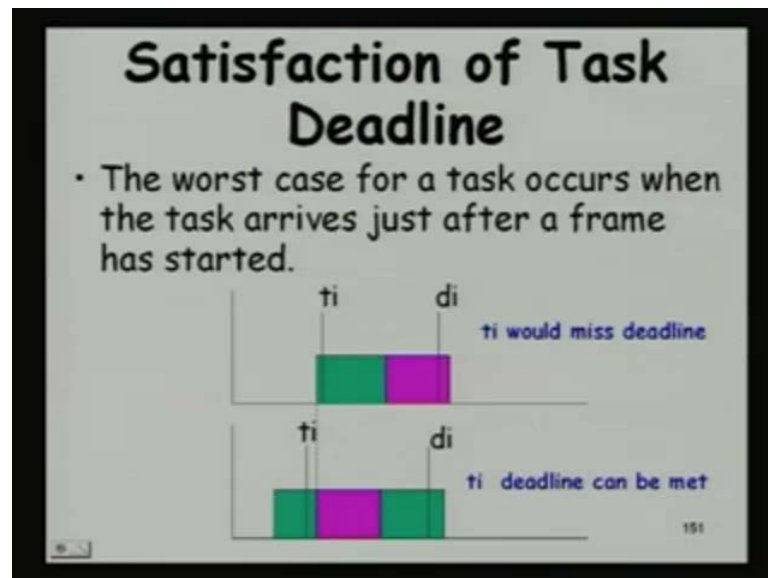
(Refer Slide Time: 32:49)



Here, just look at this diagram. See here, that the frame it is it.. The task arrived just after the frame started and it could not run. Now, we have reduced the frame size right, and the task arrived just before the frame has started, and see here, there is a full frame available, we can run the task here, in this. Here, there is no frame, I mean this will be

the frame which we can run, but by that time the deadline will be over. So, this is the third constraint - that the frame size can have a maximum certain size and this deadline this constraint cannot be violated. See the other constraints you are saying minimization etcetera, those are desirable **right**; minimization of the context, which the frames size is integral multiple of the major cycle etcetera.

(Refer Slide Time: 32:49)




So, those are not really **very...** if we violate that, I mean still it might run, but this we cannot, otherwise the tasks will miss the deadline.

Now, let us see, what is the implication of this? The tasks are periodic; that is what we had said, we are considering only periodic tasks and their start time keeps on shifting from their successive occurrences from a frame size.

(Refer Slide Time: 34:08)

Satisfaction of Task Deadline

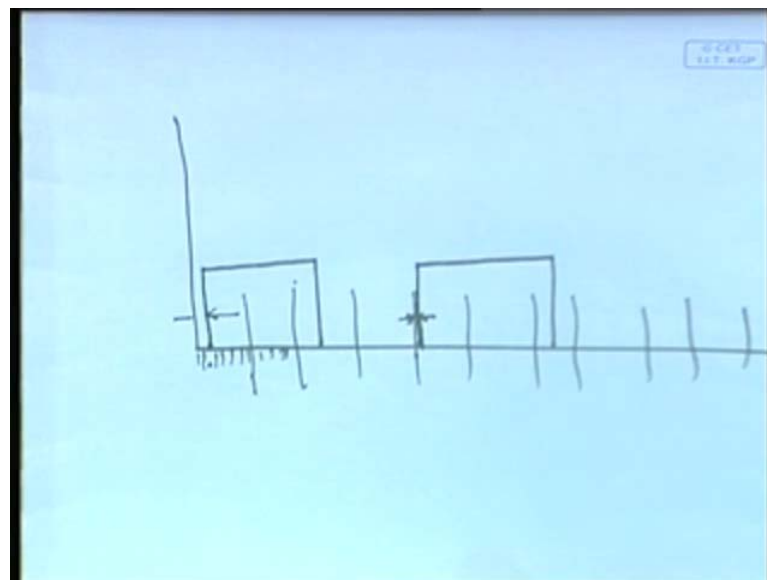
- The minimum separation of an arrival time for t_i from a frame start:
 - $GCD(F, p_i)$
- Thus, for all t_i
 - $2F - GCD(F, p_i) \leq d_i$ must be satisfied



152

Just to take an example, let us say we have a task T 1. Let us say we have a frame like this marked. These are the frames.

(Refer Slide Time: 34:44)

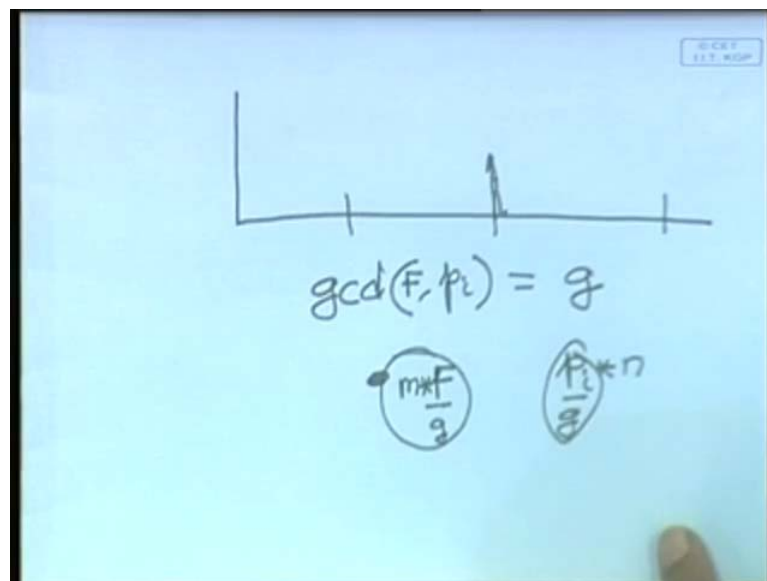


Now let us say we have a task, which arrived here, and in the next cycle, let us say it started at this point, right, because see the clock cycles are here, and what is marked the larger lines are actually the frames. So that starting of a frame to the tasks arrival will change from each task occurrence. Now, the question is that - what is the minimum time, minimum separation between a frame starting and the tasks arrival? This keeps on

changing; for every periodic task, this keeps on changing. Unless, the task arrival time is an exact multiple of a frame size, which may not be the case; it is a multiple of clock size, **right**? The tasks arrive at multiple of clock size; they do not arrive at multiple of frame size **right**.

So, what is the minimum difference that can be observed for a task from the starting of a frame? So, the result is GCD of the frame and p_i **sorry** p_i . So, if we are considering p_i is the task, and F is the frame size, then the minimum separation that can be observed for this task is $\text{GCD } F, p_i$. Now, this again was proved by one of your predecessors, who did a course earlier, not one, many of them actually came up, some of them... one person came with a very nice result, which is also there in the book. So, his argument was very simple. Let me just tell the argument that he had.

(Refer slide Time: 36:58)



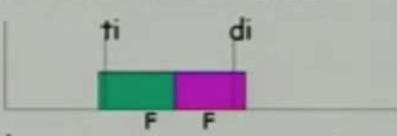
So, he says that see, let us say, we have this repetition of the frames, and then the task, and since... let us assume that the constraint is violated. So, we will prove by negation of an assumption; by contradiction we will prove. So, let's the GCD of F, p_i ; let us assume that this is not true. So, $\text{GCD } F, p_i$ is a multiple of both F and p_i , **the sorry** the F both F and p_i are multiples of the GCD. The GCD is the highest common factor **right**. So, this is the factor, the GCD, let me just write g ; the g is a factor of both F and p_i .

Now, let us assume that there is an occurrence of the task, at which the difference between the start of a frame and the task is less than g , **right**. So, we will have the situation in that case, that let us say, let us assume that they exactly coincide; let us say its 0, **right**. So, if we have a multiple of F **let us say** m into F and p_i into let us say n . Some m and n , let us say these are multiples where they actually have a 0, they arrive at the same time, **right**. So, since both of them are divisible by g . So, we can say that m and n , this is an integer, **right**. So, we can... So, by contradiction that, this they can, **they can** that can be only 0, **right**; it cannot be more than g , **sorry** less than g . You please look at the proof on the book, it is given; very simple argument or may be if you are not convinced, I will just get it onto the slide and we will discuss.

(Refer Slide Time: 39:55)

Satisfaction of Task Deadline

- The minimum separation of an arrival time for t_i from a frame start:
 - $\text{GCD}(F, p_i)$
- Thus, for all t_i
 - $2F - \text{GCD}(F, p_i) \leq d_i$ must be satisfied



So, for every task, the minimum duration from the frame to the task's arrival is $\text{GCD } F, p_i$, and therefore, for all t_i we must have $2F$ minus $\text{GCD } F, p_i$ is less than d_i satisfied. Why is that? See look at this, this is $2F$. Is it not? Now, let us say this is GCD **right**, this is g , the one that you see here, the task has arrived here is the minimal separation within the task and the frame, and so, $2F$ minus GCD should be less than d_i **right**. So, we are saying that between this and this, d_i should be. So, d_i can be later than this one, but it cannot be less than $2F$ minus GCD . So, this is a case which cannot be scheduled, **right**. So, the d_i can occur only later.

(Refer Slide Time: 41:53)

Selection of a Suitable Frame Size

- Several frame sizes may satisfy the constraints:
 - Plausible frames.
- A plausible frame size has been found:
 - Does not mean that the task set is schedulable.
- The smallest plausible frame size needs to be chosen:
 - The chances of successful scheduling is higher.

153

Now, several frames might satisfy the constraint; as we will see, we will take some examples and those are called as the plausible frames. Those are the frames which we can actually make use of. But as somebody of you, some of you were just mentioning, that even if the frames are satisfied, it does not mean that the task set is schedulable, **right**. You are saying that see, there are 4 and 2, and their frame size is 5, how will it run? **right**.

So, we have found a plausible frame size, does not mean that the task set is schedulable. And to increase the schedulability, we need to choose the smallest frame size. And we will see that the chances of successful scheduling can be higher, once we choose this smaller frame size. We will take examples and show this. Let us look at one example here.

(Refer Slide Time: 42:15)

Example 1

- Compute a suitable frame size for the following task set:
 - $e_1 = 1, p_1=4, d_1=4$
 - $e_2 = 1, p_2=5, d_2=5$
 - $e_3 = 1.5, p_3=20, d_3=20$

154

So let us consider, three tasks e_1 sorry t_1, t_2, t_3 which have execution time of 1 let us say millisecond or something, and the period is 4 milliseconds, and the deadline is 4. See all of them, the period and deadline are the same; this is typical of all real time tasks; and each of them take one clock cycle to run; this takes 1.5; 1.5 clock cycles to run and these has repeats after every 4 milliseconds, this repeats after 5 milliseconds, and this repeats after 20 milliseconds.

(Refer Slide Time: 43:12)

① 2 or greater.

② $LCM(4, 5, 20) = 20$ Maj Gcd
3, 4, 5, 10

③ $gcd(F, t_c) \mid 2F - gcd \leq d$
 $gcd(2, 4) = 2$
 $2 \times 2 - 2 = 2 \leq 4$ ✓
 $gcd(2, 5) = 1$ ✓
 $2 \times 2 - 1 = 3 \leq 5$ ✓
 $gcd(2, 20) = 2$

So, how do you do it? Let us, let me just do it on the paper. So, the first constraint is that the frame size to be chosen must be greater than all execution times. Is it not? That is the minimization of context switching. So, we can choose 2 or greater, 2 milliseconds or greater can be a frame size. We cannot have 1 etcetera, because some task will not run in one frame. Is it not?

Now, let us look at the second constraint. The second constraint is that, the frame size should divide the LCM of the periods **right**. Now, what is the LCM of the periods? LCM of 4, 5, and 20, is 20. Is it not? Now, this allows us to select frame sizes, valid frame sizes at 2, 4, 5, and 10. Is it not? These are the allowable frame sizes, considering, that the second constraint, that the frame size squarely divides the major cycle; this is the major cycle.

Now, let us look at the third constraint, that there must exist one full cycle between this arrival of the task and the deadline. And we have to consider the worst case scenario in the task arrivals. So that worst case scenario is given by GCD of F, p_i ; is not it? Now, let us try 2. See, we should try from the smaller one, because the smallest one is preferable; if 2 works, then we should use that, we need not look for 4, but if 2 does not work, we will have to look for 4. So, let us see GCD of 2... the first task runs were 4, **right**, it repeats after 4. So, the GCD is 2. Is it not? So, $2F$; the constant is $2F$ minus GCD is less than equal to d_i that were the constant if you remember. Is it not?

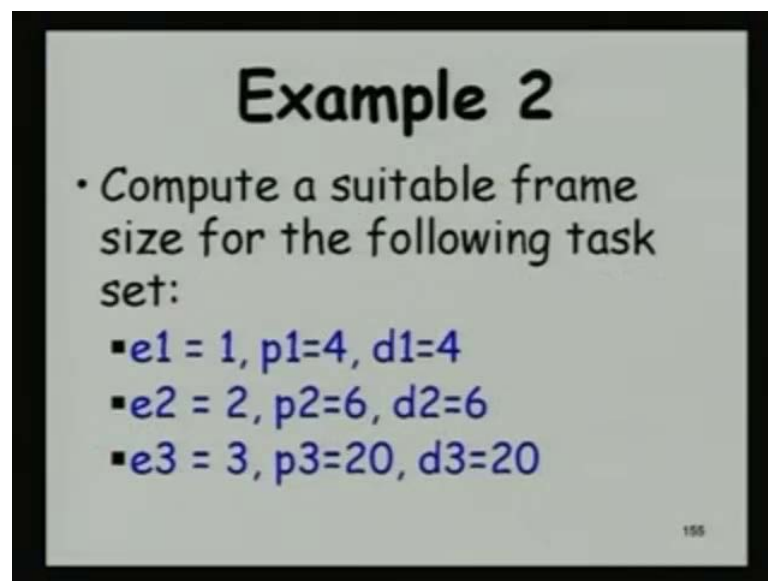
So, $2F$ will be 2 into 2 minus GCD 2 is less than the deadline is 4, satisfied for the first task, there will exist at least 1 frame. Now, what about the second task? The second task GCD of 2, 5 is equal to 1 **right**. So, 2 into F minus GCD will be 2 into 2 minus 1 is equal to 3 and 3 is less than 5 **right**. So, even for the second task 2 is satisfied. Now, what about the third task whose GCD is equal to **sorry** GCD this is 2, 20 is equal to 2 **right**.

So, 2 into 2 minus 2 is equal to 2 which is less than 20. So, all the three tasks can use 2 as the frame size, and 2 is the smallest, and we can use that. But one point that we need to consider, is that a with 2 they are all schedulable, they will run, but if you really want to minimize the number of times the scheduler needs to run, we should choose the maximum frame size. Is it not?

See with 2 it will run, but if we, let say try out 4 and we find that even for 4 that runs, so, in that case we will minimize the context switches. Sorry not context switches, the number of times the tasks the scheduler needs to run, if we choose 4 as the frame size; that is the implication. Is that ok? Fine.

So, let us look at another example.

(Refer Slide Time: 48:00)



The slide is titled "Example 2" and contains a bulleted list of instructions and task parameters. The text is as follows:

- Compute a suitable frame size for the following task set:
 - $e_1 = 1, p_1=4, d_1=4$
 - $e_2 = 2, p_2=6, d_2=6$
 - $e_3 = 3, p_3=20, d_3=20$

A small number "155" is visible in the bottom right corner of the slide.

So, since you know how to do it, maybe you can also try. So, that we are looking at three tasks. The tasks are... the execution time is 1 4 1 and the period is 4, deadline is also 4. So, all of them the period and deadline are the same. First task is 1.

Sir...

Yes please.

In previous example, you have said that we have to take 4 as the period frame size.

4, yes, if you take 4, then the number of times the scheduler runs will be...

Why only 4? Why not 5 also? 5, 5 is a more than 4. So, even what is the outline if we certainly choose the...

No; see you can try 5 also, because 5 is also a factor of 20, but if 5 runs, for 5 also the task can run, then you can choose 5 that will also minimize it, the number of times it runs. So, maybe we can try out what is the maximum frame size for which it needs to **it** **can** run and that will minimize the number of times the scheduler runs.

The maximum most number...

The maximum frame size for which the tasks are schedulable. Is that ok?

So, we need not which we are some say 4, some say 5.

No, **no**, if 5 it runs, see you can try 2, 2 we find that it runs; we will try, you can similarly try 4, if we try 4 it runs. Now 5, it may not run, then we choose 4, but if 5 runs, we can choose 5.

Sir, instead of starting from 2, we can start from 5, it runs from 5, then we can choose that itself.

Does not **does not** matter. Yes, possible. So, please try this one.

Choosing the smallest will be the best.

No **no**, what you said is for the smallest frame size, the tasks are lightly to run, because no there is likely to be 1 frame size between the deadline and the tasks arrival. A larger frame size it may not run, where as a smaller one it may run.

So, the chances of...

By taking the smallest one we are trying to find out, whether this frame size the task can run.

Yes. Schedulable.

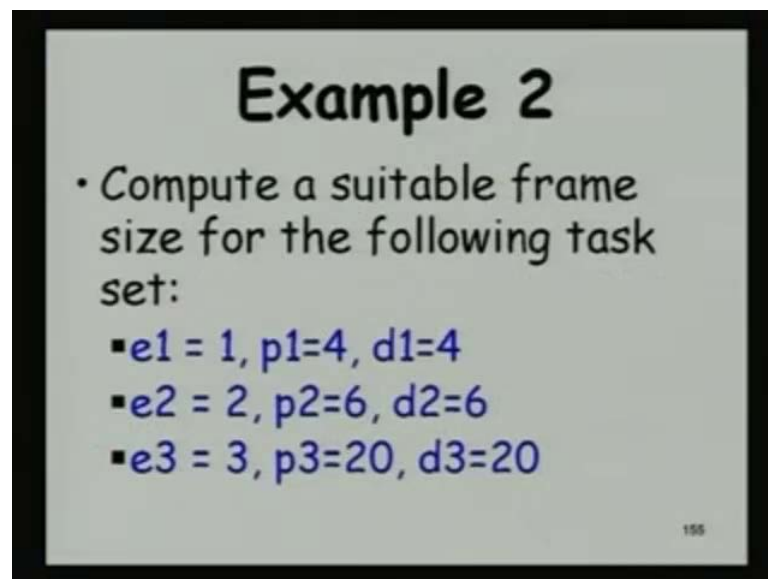
Schedulable basically.

Actually they are not that...we are stopping it as and when...

No, all those are fine; they are not each task is running in one frame size etcetera; all that are fine. What we are saying is that the number of times the scheduler wakes up and has to look at the schedule is minimized if the frame size is more. Otherwise, it knows wakes up and every frame size, and then, it just does not do much.

So, please try this one. Three tasks: t_1 has an execution 1 period 4; task t_2 , execution 2 periods 6; and task 3, execution 3 period 20.

(Refer Slide Time: 51:19)



Example 2

- Compute a suitable frame size for the following task set:
 - $e_1 = 1, p_1 = 4, d_1 = 4$
 - $e_2 = 2, p_2 = 6, d_2 = 6$
 - $e_3 = 3, p_3 = 20, d_3 = 20$

155

So, first tell me that what the maximum frame size is, **sorry** minimum frame size; the first one sets the minimum constraint. So, what is the minimum frame size? 3 milliseconds. 3 milliseconds or higher. It should be greater than or equal to 3; that is the first constraint.

Now, what about the major cycle for this? Major cycle is 60. M is equal to 60. Now, what are the possible frame sizes? Yes. What is the possible frame size for this? Anybody would like to answer this question?

3, 4, 5, 6, 10, 12 and so on.

So, 3, 4, 5, 6, 10, 12 and so on. These are many frame sizes, 15 etcetera.

Now, let us look at the third constraint. So, let us look at 3 first. So, we need to run on 3. So please try out. The first task, what is the GCD of 3, 4? For the first task is GCD of 3, 4 is 1.

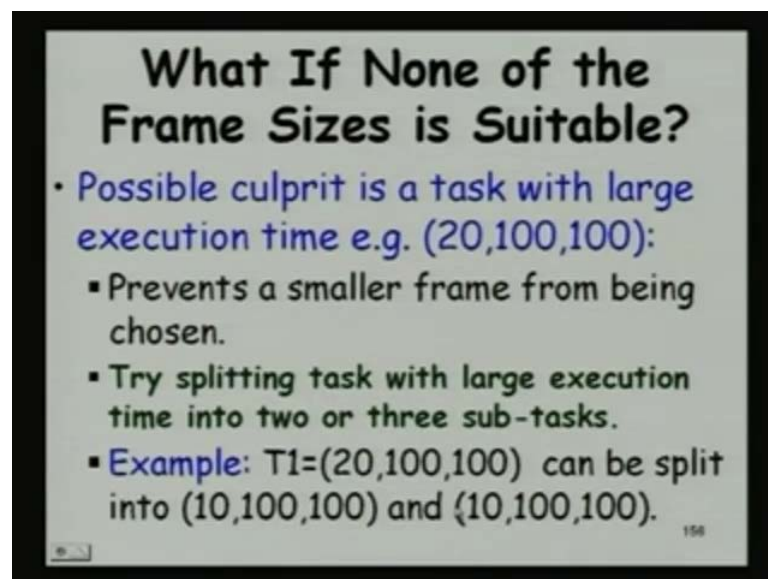
So, $2 \text{ into } f \text{ is } 2 \text{ into } 3 \text{ minus } 1 \text{ is less than equal to } 4$. It does not satisfy. Now, let us look at the next frame size 4. So, for 4, GCD 4, 4 is 4. Now, $4 \text{ into } 3 \text{ minus } 4 \text{ is equal to } 8$ sorry $2 \text{ into } 4 \text{ into } 3 \text{ minus } 4 \text{ is equal } 8$, sorry 2, I am sorry, $2 \text{ into } 4 \text{ minus } 4 \text{ is equal to } 4$ less than equal to 4, satisfied. Is it not? Satisfied for... is that ok? $2 \text{ into } 4 \text{ minus } 4 \text{ is equal to } 4$, which is less than equal to 4.

Now, for the second task if you try out, the GCD 6, sorry 4, 6 is 2. Is it not? So, $2 \text{ into } 4 \text{ minus } 2 \text{ is less than equal to } 6$. Is it not? It is equal to 6. So, is less than equal to 6 is satisfied for second task also all right.

What about the third task? The third task GCD 4, 20 is equal to 4 right. Now, $2 \text{ into } 4 \text{ minus } 4 \text{ is less than equal to } 20$; satisfied.

So, all the three tasks can run on frame size 4, but not on 3. So, it is not really necessary that they will run on the smallest frame size; it is not necessary. So, let us proceed further.

(Refer Slide Time: 55:42)



What If None of the Frame Sizes is Suitable?

- Possible culprit is a task with large execution time e.g. (20,100,100):
 - Prevents a smaller frame from being chosen.
 - Try splitting task with large execution time into two or three sub-tasks.
 - Example: $T1=(20,100,100)$ can be split into (10,100,100) and (10,100,100).

158

Now, as some of you were saying that - what if a task takes too much time and some other task takes too little time?

So, that can make it un-schedulable. So, a task with a large execution time like 20 and others are 2 etcetera, they may can, they can prevent the smaller task from being, frame being chosen, and in this case, we will have to split the task with a large execution time into two or three sub-tasks. For example, 20, 100, 100. So, 20 is the execution time, and 100 is the period, and 100 are the deadline can be split into 10 10. 10 and 10 two sub-tasks we can consider this to be.

So, since this hours time is up, we will just stop here; we will come, we will proceed in the next class. Thank you.