

Real Time Systems
Professor. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Lecture 11
Cyclic Scheduler

Welcome to this lecture, over the last few lectures, we have been looking at the Real Time schedulers. Real time schedulers are a very important component of every real time operating system and there are several types of schedulers and the simplest scheduler is the clock driven schedulers and we have been discussing over last one or two lectures about the clock driven schedulers. We had looked at the table driven scheduler, which is a very simple infinite loop in which we set a one set timer based on a scheduled table, consult the schedule table, see a task runs for how much time set the one set timer.

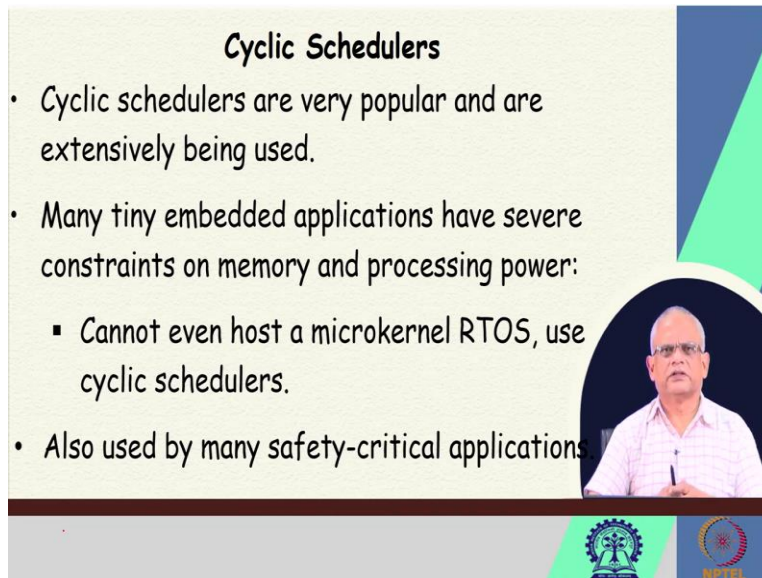
And then the scheduler starts execution of this task and it idles if it completes early until the one set timer fires and then the scheduler gets control, that is the scheduling point and the scheduler consults the schedule table again and finds the next task to run, sets the one set timer depending on how much time the task is supposed to take.

And one important question at this point is that how does the designer know, how much time a task is going to take? Because you have to write that in the schedule table. The answer to this question is about the worst case execution time or WCET, the worst case execution time, if we consider a program or a piece of code, we can see that there are various paths through this code, there may be loops and various paths in this code.

So, the worst case execution time is the longest time it can take, need to experiment with various data and see the situation on which the longest execution time is executed and that for is the worst case execution time.

The cyclic schedulers are an improvement over the table driven scheduler, in the sense that, do not have to each time set a one set timer, just have a one periodic timer which is set once and it keeps on giving interrupts at every fixed time, which we call as the minor cycle. We had looked at some basic aspects of the cyclic scheduler and these are used widely in small embedded devices and safety critical applications. Now, let us proceed from that point onwards.

(Refer Slide Time: 03:42)



Cyclic Schedulers

- Cyclic schedulers are very popular and are extensively being used.
- Many tiny embedded applications have severe constraints on memory and processing power:
 - Cannot even host a microkernel RTOS, use cyclic schedulers.
- Also used by many safety-critical applications.

The slide features a video inset of a man in a white shirt and glasses speaking. The background is light green with a blue and green geometric design on the right. Logos for IIT Bombay and NPTEL are visible at the bottom.

The cyclic schedulers are very popular are extensively being used, these are also called as cyclic executives, because the operating system is almost synonymous with the cyclic schedulers, very small operating system and the major component is the scheduler and possibly it has some error handling capabilities, like let us say what happens when a frame overrun occurs like a code segment or a task is taking more time than the frame it does not stop the frame.

What does the cyclic scheduler do? Or the cycle executive? How does it take care of this? And also how does it handle the periodic tasks? So, those are some of the things that the cyclic executive can do, but let us look at the basic cyclic scheduler, because as long as you are developing a small embedded device or a safety critical device very likely to use the cyclic scheduler. Many small embedded applications, which have constraints on the processing power and memory, they do use the cyclic schedulers.

The embedded devices that we are talking of, the cannot even host simple microkernel based real time operating system, out of all the real time operating systems, other than the executives, the microkernel real time operating system can be configured to take the least space, but even this small embedded devices, they cannot even host a microkernel real time operating system and then they will have to use a cyclic executive.

Not only tiny embedded applications, but also several safety critical applications do use the cyclic executive, because we had seen the reason the last class that the cyclic executive code is

small and it can be provably guaranteed, the behaviour of this can be guaranteed that under no circumstance it will take long response time or gets into an infinite loop or hangs and so on, all those aspects can be proven because the code is so small.

But in the sophisticated real time operating systems, it becomes difficult to give that guarantee, because there are millions and billions of paths and do not know if there are other paths which exist to it may be tested for million paths, maybe there are more paths I do not know, but here there is a small well understood number of paths is can be easily identified and you can test the executive for all the paths.

(Refer Slide Time: 07:02)

Cyclic Schedulers

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

Task	Timer
T1	50
T3	75
T2	30
T4	85
T3	70

Schedule table

The slide also features a video inset of a man speaking and logos for IIT Bombay and NPTEL at the bottom.

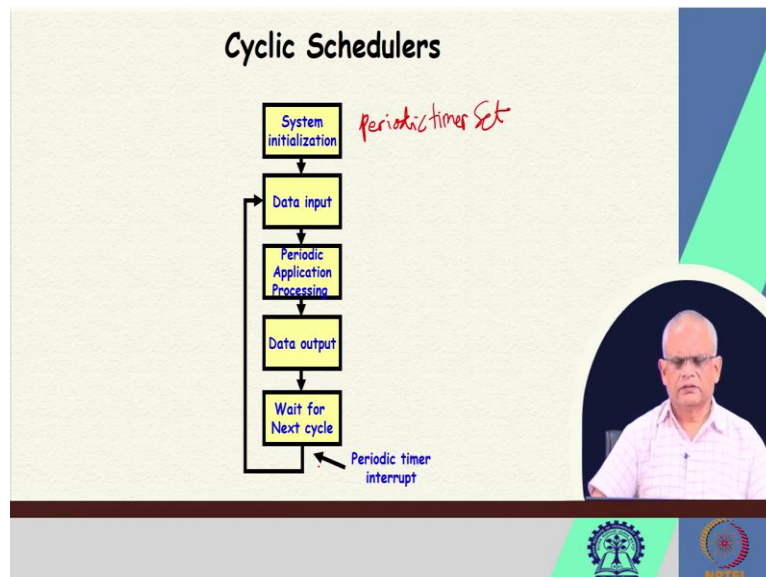
As we had mentioned in the last lecture, at the heart of the cyclic scheduler is the schedule table, the schedule table, where the designer computes beforehand which task needs to run for how long and the long is in terms of, number of, it can be given in millisecond or number of interrupts and so on. Typically, these are used for simple applications having maybe less than a dozen tasks and remember we have been saying that the task here is not the task in the operating system terminology, but here task is a segment of code, maybe few procedure calls.

And all these procedures they share common data and these are not really tasks, these are not normally preemptible only when we discuss about a periodic tasks, we will look at preemption but periodic tasks will not be preemptible. So, we have this schedule worked out before the program executes which task is going to take how long, we set a one set timer, ok in a cyclic

scheduler it is a periodic timer and it is assigned to one of the time slots and it runs and this is repeated forever.

So, that is the main idea here, the central is the schedule table, but the question is that, if there are n tasks, T_1, T_2, T_n , with these are periodic tasks n periodic tasks with periods P_1, P_2, P_n , so what is the length of the schedule table? For how long we need to store the schedule like this the T_1 runs for 50 etc. We had discussed this last time, let me just repeat that. This is the LCM of P_1, P_2, P_n easy to argue, why LCM, because after this and this is the, at LCM all the periods all the tasks are the same point of time at which they had started, so they are identical conditions it repeats.

(Refer Slide Time: 10:07)

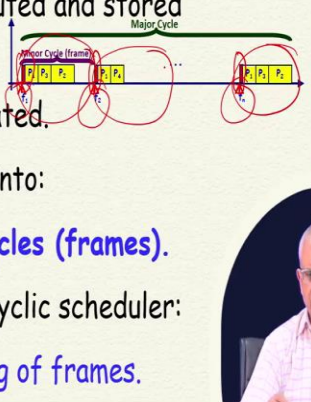


The cyclic scheduler, if we represent it will be like this that initially the system initialization and here we also set the periodic timer and then the code for the task is executed, the code for the task is typically read some input variables, do some processing and possibly produce some output and then it waits for the interrupt to occur from the periodic timer and as soon as the periodic timer interrupt occurs, the scheduler gets invoked, it consults the schedule table finds out the next task to run and it runs it that is it, keeps on doing that infinitely. So, this infinite loop that is the basic idea of the cyclic scheduler.

(Refer Slide Time: 11:25)

Cyclic Schedulers

- The schedule is Precomputed and stored for one **major cycle**:
 - This schedule is repeated.
- A major cycle is divided into:
 - One or more **minor cycles (frames)**.
- Scheduling points for a cyclic scheduler:
 - Occur at the beginning of frames.



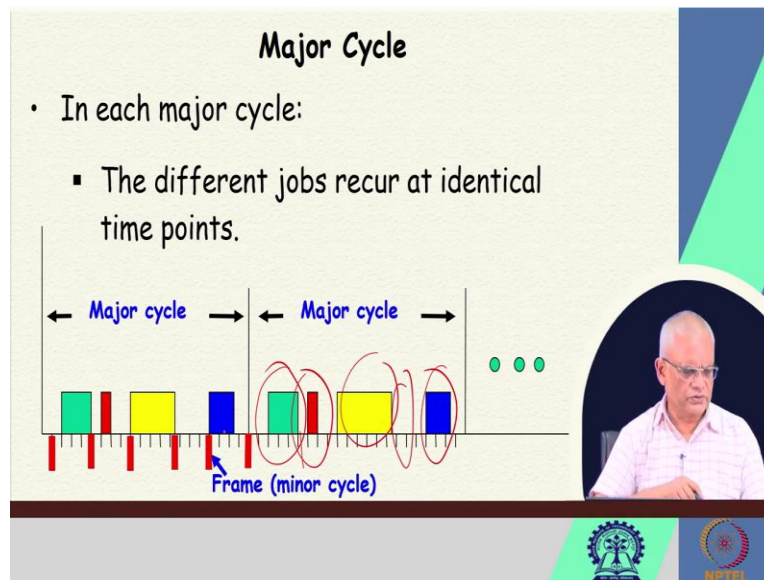
The diagram illustrates a cyclic scheduler. A horizontal timeline shows a sequence of frames. Each frame is a box containing tasks P1, P2, and P3. Red arrows point to the start of each frame, indicating scheduling points. A green bracket above the timeline spans the duration of one major cycle, which consists of multiple frames. A video inset in the bottom right corner shows a man in a white shirt speaking.

And the two important concepts here are the major cycle and the minor cycle. The major cycle is the LCM of all task periods and the schedule needs to be stored for the major cycle and from that it just keeps on repeating infinitely and the major cycle is divided into one or more minor cycles, the minor cycles are also called as frames.

Now, each task is assigned to one frame, so we have a set of jobs that occur in a major cycle, we assign those jobs to the minor cycles and we can club few of the jobs into one cycle and for doing that assignment of the jobs to the frames, we can use a network flow maximization algorithm, but we will not really go into that, we will just assume that the jobs are defined, the jobs like we have written here P1, P2, P3 etc., those have been already assigned to the frames, that means the jobs are defined.

And each job is assigned to one frame and just look at here f_1, f_2, f_3 these are the clock interrupts, the periodic timer interrupt and these mark the beginning of the frame and as the interrupt from the periodic timer comes, the scheduler gets activated and that runs for some time consulting the schedule table finding out which job to run and running it and that takes a little bit of time and that we have shown here as the initial red time on the time axis, there is a small time for which the scheduler runs and then the job assigned to the frames run. So, this the first frame, this is the second frame and this the n^{th} frame and so on.

(Refer Slide Time: 14:18)



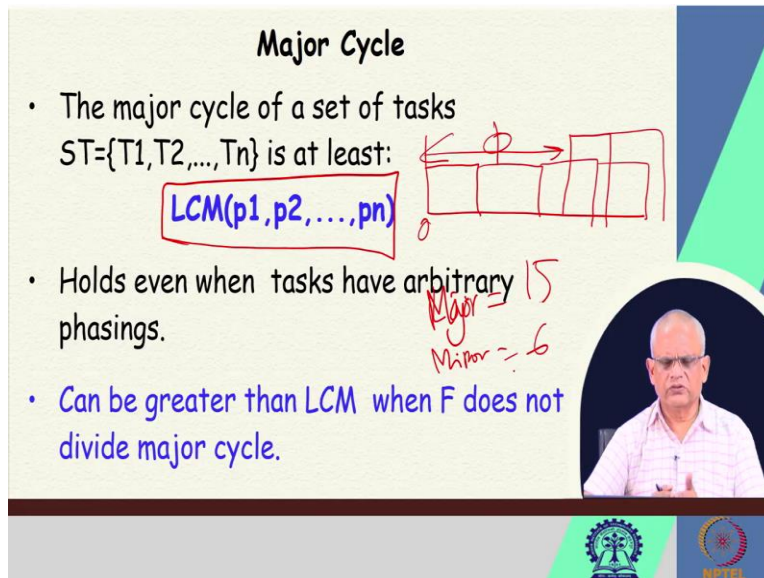
Now, the major cycle is crucial parameter and each major cycle consists of an integral number of frames and to each frame one task is assigned. So, this is assigned to the first frame, second frame, third frame and this is the fifth frame and in each major cycle, the same thing repeats. So, again in the first frame, the green job executes, in the second frame the red job, third frame the yellow job, the fourth frame is idle, and the fifth frame the green job and so on.

So, in each major cycle the jobs recur or they are executed exactly in the same frame and each frame runs one job and as I already mentioned that for our discussion, we will consider the job to be just one integral core component, but then a job might consist of several smaller jobs, which are put together, but that we will not really focus on this course.

(Refer Slide Time: 15:59)

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.



Now, if we have n tasks, then the major cycle is the LCM of the periods of those n tasks, these are n periodic tasks and the major cycle is given as the LCM p_1, p_2, p_n . Now, this holds even if the tasks are arbitrary phasing. For example, one task may just start from 0 and keep on repeating, another may start with a phase and then it keeps on repeating, so this is the phase for this.



The third job might have a different phase, but even when such a situation occurs that tasks have different phases, still we use the same formula to find the major cycle $LCM p_1, p_2, p_n$ and then we divide the major cycle into an integral number of minor cycles or frames. But the question is that, what if the frame duration does not properly divide or squarely divide the LCM, like LCM is let us say 15, LCM is the major cycle is, the major cycle is 15 and the minor cycle is let us say 6, is it possible?

Ok If we really do this, then we need to have the schedule table much longer, we will have to the schedule table we need to store at least for two major cycles here, that is 30, which divides 6. So, if the minor cycle does not divide the major cycle, the schedule table size becomes long, requires more memory and more processing and nobody would like to do that and therefore, the rule is that as far as we use a cyclic scheduler, we have to select a frame size which squarely divides the major cycle.

(Refer Slide Time: 18:40)

Minor Cycle (Frame)

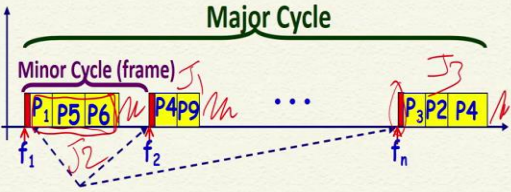
- Each major cycle:
 - Usually contains an integral number of minor cycles (frames).
- Frame boundaries are marked:
 - Through interrupts generated from a periodic timer.





So, each major cycle contains an integral number of frames and the frame boundaries are marked by the interrupts generated by the periodic timer.

(Refer Slide Time: 18:56)

Major and Minor Cycle (Frame)



- Cyclic scheduler runs in response to a clock tick event
- Red bar shows time to execute scheduler



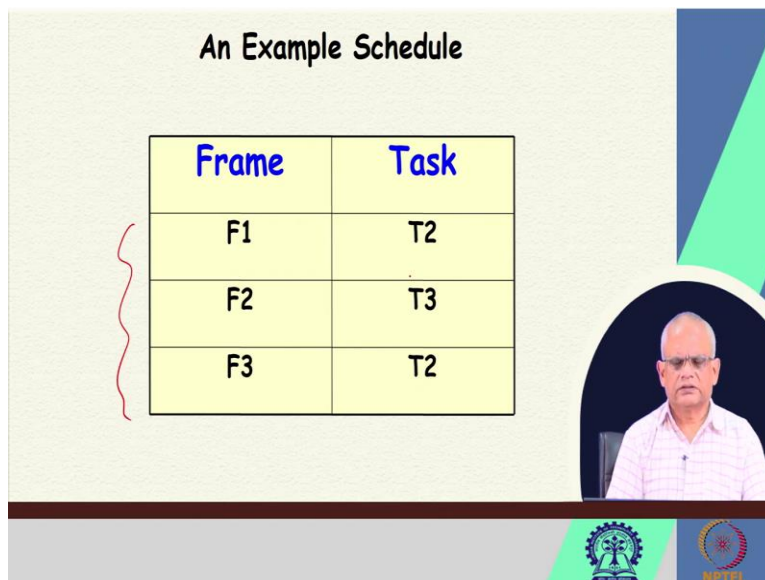
Now, this diagram just captures that, as I was saying that when defining the jobs, we might have put several small jobs together, to fit into the frame size and we said that we will not really delve into that, we will just assume that jobs will just consider this itself as the job, one job. So, maybe J_2 , this is the J_1 , J_3 , we will not consider the small jobs which constitute these jobs.

And each of these jobs are assigned to exactly one frame and there can be some idle times at the end of the frame and as we have been saying, the scheduler gets invoked on the periodic timer alarm and the scheduler wakes up when there is a timer alarm and then it runs for small while and then it runs the corresponding job J_3 here.

(Refer Slide Time: 20:23)

An Example Schedule

Frame	Task
F1	T2
F2	T3
F3	T2



This is the example of a schedule table which stores an assignment of frames to the tasks, so if we our major cycle contains three frames, we can assign different tasks, some of the frames can be idle, but we cannot assign two tasks to one frame, I mean, if we had to do that, we would have already combined them and that would constitute our task here. But at this level, we will just consider that each task, a single task is assigned to one frame, the earlier combining different small jobs into larger jobs that are done, earlier.

(Refer Slide Time: 21:17)

Constructing a Schedule

- Construct static schedule for a Major Cycle
- Cyclic Executive repeats this schedule
- There may be resulting idle intervals

Cyclic Schedule

f_1	T_1
...	...
f_5	T_5

Task = (r, e)

T_1	=	(4,1)
T_2	=	(5,1.5)
T_3	=	(20,1)
T_4	=	(20,2)
T_5	=	(25,3)

Now, let us see how to construct a schedule. So, basically we need to develop the schedule table for one major cycle and then this repeats, so our schedule table will look like this the different frames, we have to find out how many frames are there in the major cycle, we have to define the frame size, if there are five frames in our major cycle, the schedule table has five entries and then we also write which job is assigned to which frame.

The major cycle is also called as a hyper period and if we, this is our schedule table, then at time equal to 0 the frame 1 and then if P1 is to be run at f_1 , then the scheduler just runs it and then and the clock interrupt, let us say 4 is the frame size, then at 4 the clock interrupt will occur and then it will run the next job and so on.

(Refer Slide Time: 22:37)

Partitioning A Major Cycle into Frames

Major Cycle (Hyperperiod)

- Design steps:
 1. choose frame size,
 2. partition jobs into slices (if needed),
 3. place jobs/slices into frames.
- At Frames boundaries :
 - Cyclic executive performs scheduling
- There is no preemption within frame

So, one important design task here is to choose the frame size. The major cycle is easily obtained by just considering the LCM of the periods of the given jobs, given tasks. But then we have to choose the frame size because several frame sizes may be possible which integrally divide the major cycle, we have to choose one of them, the best one and some of the frame size may be infeasible, we will choose the one feasible frame size, we will see the details of that.

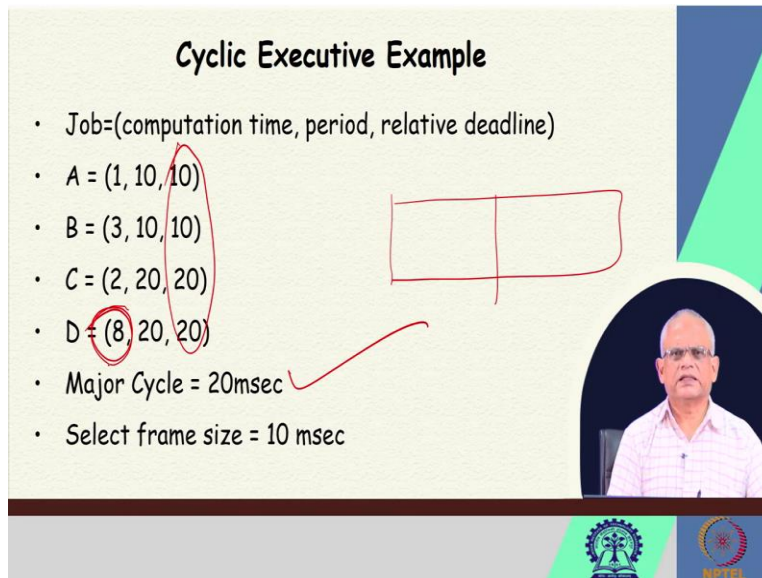
And then we will see that if we cannot really have enough frames to run the different tasks, we might have to partition some jobs into smaller parts and see if it becomes possible to assign the task to a unique frame and then we assign, do this assignment of these jobs into the frames and that is the, in nutshell that is our design state.

And at the frame boundary, the scheduler we will consult the schedule table and find which tasks to run in the frame and it just runs it and just remember that there is no pre-emption within a frame, there is no other interrupts which are recognized by the scheduler, it just recognize as at the end of the frame interrupt. So, there is no preemption within a frame. One task is assigned to a frame and the tasks may consist of several small jobs that we had mentioned, so that the frames are filled as much as possible.

(Refer Slide Time: 24:44)

Cyclic Executive Example

- Job=(computation time, period, relative deadline)
- A = (1, 10, 10)
- B = (3, 10, 10)
- C = (2, 20, 20)
- D = (8, 20, 20)
- Major Cycle = 20msec
- Select frame size = 10 msec



So, let us look at a small example, that we have two tasks. If we use this notation, that a job is defined in terms of its computation time period and relative deadline, we have two in our simple application, where the computation time is 1 and for the second one computation time is 3 and the period and the relative deadline are both same 10.

And we have another one whose computation time is 2 and the period and relative deadline is 20 and the third one is, fourth one is computation time is 8 and we have the period as 20 and here we can see the LCM of the periods is 20, so finding the major cycle is no problem, LCM of all the task periods and that is 20 millisecond and these are all in millisecond.

And we need to select a frame size, can we select 10 millisecond? Yes, 10 millisecond squarely divides 20, it is a feasible frame size. But if we select 10, then a major cycle consists of only 2 frames and here we have 4 tasks to assign, we cannot use 10 actually, because then we do not have enough frames to assign the tasks, we need to choose a smaller frame size maybe 4 or 5, because they also squarely divide 20.

But then one task here is taking 8, we cannot really make it 5 or 4, because one is there already taking 8. So, we need to split this into smaller tasks, two smaller tasks and then we can try to compute the schedule. So, that is the kind of design step involved, but the frame size other than that it squarely divides the major cycle and it is larger than all the task computation times we

have few other constraints, we will look at the constraints in the next class, so that for a realistic problem, we can design the schedule table.

That is a crucial step in designing a real time system, we are at the end of the lecture now and we will look at how to select a proper frame size, what are the constraints on the frame size and given a task set, how do we prepare the schedule, so that we will look at in the next lecture, we will stop here. Thank you.