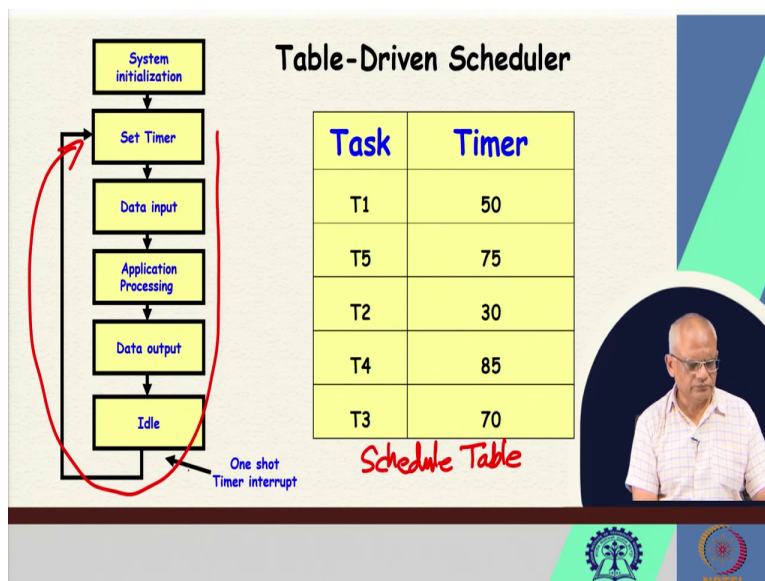


Real Time Systems
Professor Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 10
Basics of Cyclic Schedulers

Welcome to this lecture. In the last lecture, we were looking at some basics of clock driven scheduling and we said that the popular types of clock driven scheduler are round robin, table driven and cyclic schedulers. The round robin scheduler is not really a real time task scheduler, because it does not do anything special to let the tasks meet their deadline and that is why we did not discuss much.

Now let us see discuss about the table-driven scheduler and the cyclic scheduler, which is a refined version of the table-driven scheduler, it takes care of several shortcomings of the table driven, basic table-driven scheduler. Now first, let us look at the table-driven scheduler.

(Refer Slide Time: 1:12)



The main thing here is the schedule table. This is the schedule table, this is the main thing in a table-driven scheduler, this is pre designed by the developer of the real time system, and he would have two columns here. Actually, one column, but then just for our understanding, I have

written two columns that T1, T5, T2, T4, T3 etc. which tasks to run and another one is how much time to run?

Now, the scheduler by itself is like a infinite loop. Initially, the system initialization is basically test and set some parameters and so on. Now, in this loop, so, this is the loop, in this loop, each time the scheduler becomes active starts here. Okay, the scheduler is actually active all the time here. It just sets the timer and if there is any data to be read, it reads that processes that data produces some output or maybe some actuator commands and so on.

And then it waits here idles until there is a timer interrupt and it is a one set timer. So basically, each time the table-driven scheduler looks at the schedule table and finds out how much time the code will run and it sets the timer to that value. For example, initially, it will set it to 50 and for T1 it knows what are the code to run, and then it will run that code for reading data maybe these are two or three procedures, P1, P3, P5 and this is P2, P7, P9 etc. So, there are some procedures it is called a sequence of procedures.

So, just as starts that code here and then once the code completes, it just keeps on waiting, because 50 was the time for which the one-shot timer was set. And when the timer interrupt comes it consults the schedule table and then sets the timer to 75 and then the code that is required in T5 that is run here which may involve again data input, application processing, output.


And again, it waits for the timer to give interrupt and once the interrupt comes again it starts for the next task, and it just keeps on repeating at the end of the last task, it again starts from the first task. So, that is the basic idea here. It is kind of infinite loop but then it just keeps on idling at the end until the timer interrupt occurs.

(Refer Slide Time: 5:26)

Basic Table-Driven Scheduler

```
const int SchedTableSize= 10;
timer_handler () {
    int next_time; task current;
    current = SchedTable[entry].tsk;
    entry = (entry+1) % SchedTableSize;
    next_time = Table[entry].time + gettime();
    set_timer(next_time);
    execute_task(current);
    return;
}
```

Task	Timer
T1	50
T5	75
T2	30
T4	85
T3	70



If we have to represent the scheduler, that is the cyclic executive in the form of a pseudo code. This will be the one that after every timer interrupt; the control is handled to the timer handler. And the timer handler here it just gets what is the next, the current tasks to run, which code to execute. And then it increases the next entry level that is which is the next task to run, next it ready for running the next task and it just gets what is that timer that is set here on the table.

So, if it is 50, it will get 50 plus the current time. And that is what it will set under set timer here. So, this is the one-shot timer and it sets it to initially to 50 and then the task, current task it runs and then it waits until the next timer interrupt occurs. So, in pseudocode for this is the representation. But finally, this will be an assembly or a machine code implementation. But the main idea is that we have this schedule table. And we just have which code to run, how much time are that these are all mentioned. And it just keeps on doing that and at the end here, since it is mod (%) operator with respect to the table size, the schedule table size and therefore it comes back to the first task.

(Refer Slide Time: 7:54)

Table-Driven Schedule: Example

- Consider a system of four tasks, $T_1 = (4,1)$, $T_2 = (5,1)$, $T_3 = (20,1)$, $T_4 = (20,2)$.
- Static schedule: *Period = 4 clock Execution time*

$LCM(p_i) = 20$

But then, one thing we must understand here is that it is the designer who decides which tasks to run next, we are loosely using the term task but remember that it is just one program there are no tasks actually, the task is basically a sequence of procedure calls. And let us say we have four tasks and we want to do a table-driven schedule for this. So, task T_1 it needs to run with a period of 4 clocks, 4 clocks. And for 1 clock so it runs 4 clocks that is the period and this is the execution time is 1, execution time.

Similarly, T_2 every 5 clock, it needs to run for 1 clock, T_3 every 20 clock needs to run for 1 clock and T_4 every 20 needs to run for 2 clocks. How do we prepare the schedule table here? We need to prepare manually before the system starts. We need to store a schedule table and the scheduler will consult the schedule table. One way is that okay, the first task it is the shortest task. Every 4 it needs 1. So, what we will do is we will just keep 4 here and we will put T_1 and then the T_2 it needs to run for 1 clock every 5 so we will put and T_3 , T_3 every 20 it needs 1.

So, we can develop a schedule. So once T_1 run, we can run T_2 or T_3 and so on we will do through a manual trial and error, we can develop a schedule table, but what is the length of time for which we to need to prepare the schedule table, we need to prepare the schedule table for LCM

of the task periods. So, LCM P_1, P_i is the period of the tasks. So here it is 4, 5 and 20. For the four tasks, the period is 4, 5 and 20 and 20. And the LCM works out to be 20.

So, we need to prepare the schedule table for 20 clock interrupts, now by trial and error we can prepare the schedule for 20 that at some slot, some tasks are assigned some slots, nothing runs, it is idle. But then the question is that, is the schedule table unique? No, we can have various schedule tables which still meet all the requirements we can have a greedy heuristic where we first assign T_1 and then T_2 and so on.


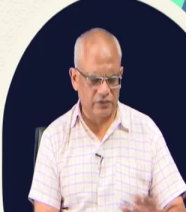
And we can prepare a schedule table, but the greedy heuristic may not always produce a successful schedule even though a schedule may be possible. But then we have genetic optimization and so on, where we can easily write a program which will produce a scheduled table for a set of tasks. If at all that set of tasks can be run, it will be able to produce the schedule table.

So, here one of the skills of the designer is to produce the schedule table for very simple system manual trial and error working will produce the schedule table otherwise, you can use a genetic algorithm or something. If we are not able to do it manually, you can try a genetic algorithm or something which will produce the schedule table and we know the length of the time for which the schedule needs to be prepared and very simple scheduler. But let us examine that is it good scheduler, are there any disadvantages of using this scheduler?

(Refer Slide Time: 13:36)

A Disadvantage of Table-Driven Schedulers

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





The main problem here is not about preparing the schedule table. The schedule table is done only once by the designer offline and as I said can use various optimization techniques like genetic algorithm or something and can produce or maybe a bin packing, a type of bin packing algorithm and so on and produce a schedule, manually working out maybe sometimes for complex systems, it may be difficult to work out the schedule table, but using a simple heuristic we can work out the schedule table.

But then the main problem actually is that the one-shot timer needs to be set again and again for every task, we need to set the timer and setting timer requires computation time. So, the overhead of this scheduler is rather large because after all the tasks that we are talking of here are very small tasks, maybe the sample the temperature, the temperature is large switch off etc. So, which may be just requiring milliseconds or microseconds to run the tasks and if our scheduler on every scheduling event needs several milliseconds or microseconds of processing the overhead becomes very large. So, that is the major disadvantage of the table-driven scheduler, which is overcome by the cyclic scheduler.

(Refer Slide Time: 15:42)

Cyclic Schedulers

- Cyclic schedulers are very popular:
 - 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.



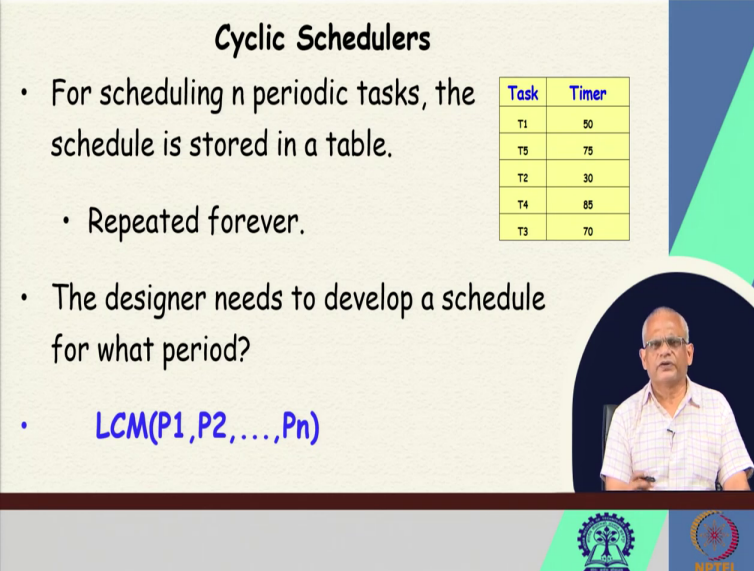
The cyclic schedulers overcome the problems of the table-driven scheduler and are also not that complicated. So, almost everybody in real time embedded application would like to use cyclic scheduler, if the application is small enough, simple and static tasks, no sporadic tasks and so on, would rather use a cyclic scheduler rather than event driven scheduler, which are much more sophisticated, bulky and so on, this is just the executive, just as scheduler as the operating system is very popular in small embedded systems, extensively being used.

Many tiny embedded applications, which have severe constraints on memory and processing power, the cyclic schedulers are popular, we will see that in the event-based scheduler will have modularized operating system called as a microkernel real time operating system. Here the main idea in a microkernel real time operating system is that depending on the size of the application, that is the system that we are developing, we can tailor the real time operating system, we can leave out some components; add some components and so on.

So, they are tailorable, the microkernel real time operating systems are tailorable there is a very small kernel and then the other parts can be added or may not be used. But even the microkernel real time operating system requires megabytes of storage for hosting the operating system, the basic microkernel and therefore, for many embedded applications where such constraints exists, the need to use the cyclic schedulers. And also, as you were mentioning that many small safety

critical applications, they prefer to use the cyclic scheduler, because this well proven the reliability is established and can be used in safety critical applications.

(Refer Slide Time: 18:22)



Cyclic Schedulers

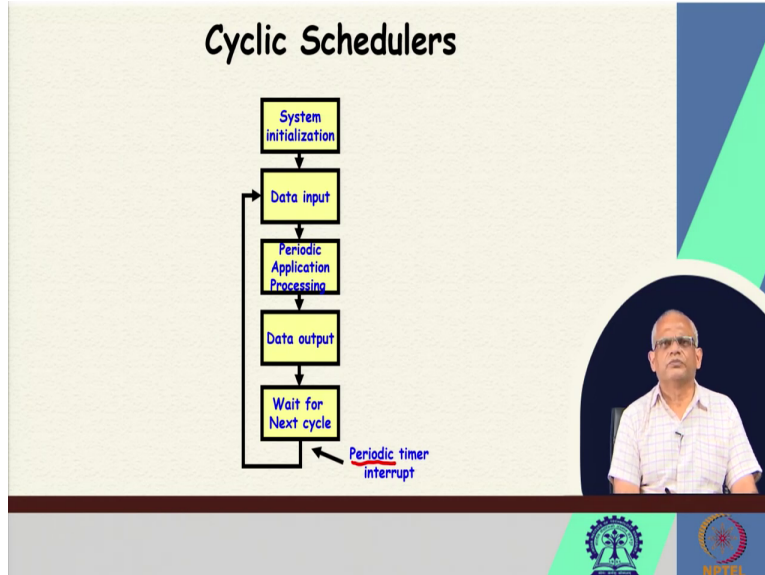
- For scheduling n periodic tasks, the schedule is stored in a table.
 - Repeated forever.
- The designer needs to develop a schedule for what period?
- $LCM(P_1, P_2, \dots, P_n)$

Task	Timer
T1	50
T5	75
T2	30
T4	65
T3	70

The slide features a video inset of a speaker in the bottom right corner. At the bottom of the slide, there are logos for IIT Bombay and NITW.

Now, let us look at the cyclic schedulers. Here again, we have a scheduled table, the schedule has to be restored, it has to be statically worked out the schedule and stored and the schedule is repeated forever something like a table-driven scheduler. And just like the table-driven scheduler, we need to develop the schedule for LCM of the periods of the tasks. But then you will ask that what is the difference between a table-driven scheduler and a cyclic scheduler. Now, let us look further into the cyclic scheduler.

(Refer Slide Time: 19:12)




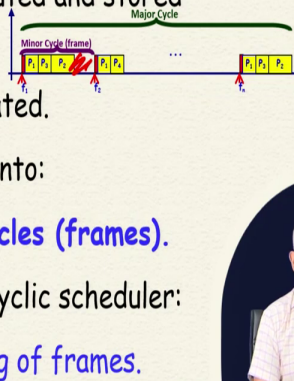
Here even the working of the scheduler looks almost like a table-driven scheduler, system initialization then we have this timer handler where it just data input and periodic application processing, data output, wait for next cycle etc. But one thing you can notice here that the set timer is missing here. We have set the timer during system initialization only once because we use a periodic timer here which needs to be set once and therefore a lot of overhead is saved here.

In cyclic schedulers, unlike the table-driven schedulers, we use a periodic timer, which needs to be set once, rather than the one-shot timer used in the table-driven scheduler need to be set again and again, significant overhead that is saved. But how exactly it uses the periodic timer. Let us look further, we have just seen the difference between the table-driven scheduler and the cyclic scheduler very briefly, but let us see more details of cyclic scheduler.

(Refer Slide Time: 20:35)

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.



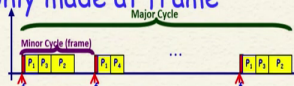
Here in cyclic scheduler, there is a concept of a major cycle and a minor cycle. The minor cycle is also called as a frame, in the major cycle, that is the repeatable the schedule table need to store the schedule for one major cycle and that schedule is repeated, but the major cycle is divided into an integral number of minor cycle, the minor cycle are also called as frames and the major cycle is the integral multiple of the minor cycle or the frame and every frame, we assign one task to run.

The task may consist of several procedures. So, the first task let us say the first frame f_1 we have written here in first frame f_1 we have this P_1, P_3, P_2 procedure is run that is task 1 and there is some idle instant here and then the timer interrupt, the periodic timer interrupt comes and we have the second frame the second frame, the next task is run and so on. So, every task is assigned to a thread and what are the scheduling points here, the scheduling points is the starting of the frames, at that time the timer interrupt comes and that is the scheduling point.



(Refer Slide Time: 22:30)

Cyclic Scheduler Basics

- Scheduling decisions are only made at frame boundaries.
- Exact start and completion time of jobs within frame is not known
- Max computation time cannot exceed frame size
- Jobs are allocated to specific frames
- Major cycle is also called a **Hyperperiod**.



The diagram illustrates a cyclic scheduler. It shows a horizontal timeline with a vertical axis on the left. A 'Minor Cycle (frame)' is indicated by a bracket above the first frame, which contains three jobs labeled P_1 , P_2 , and P_3 . A 'Major Cycle' is indicated by a longer bracket above the entire sequence of frames. The timeline shows a repeating pattern of frames, with the first frame containing P_1 , P_2 , and P_3 , and the second frame containing P_1 , P_3 , and P_2 . Ellipses indicate that the pattern repeats. The jobs are represented by small rectangles within the frames, and their start and completion times are marked with red arrows on the timeline.

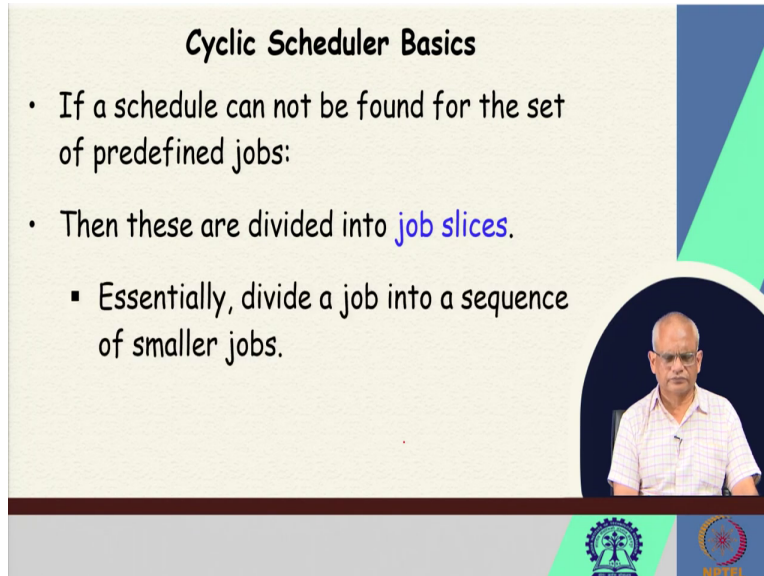


The scheduling decisions are made at the frame boundaries. And basically, the scheduler consults the schedule table and decides which tasks to run. That is about it, no timer setting and so on. And of course, none of the task computation time can exceed the frame size because within one frame, every task will be assigned one frame and the task has to complete with that, within that frame. And the jobs are allocated to specific frames, each job to one frame and the major cycle is also called as the hyper period because it is the multiple of, it is the lowest common multiple, the LCM of all the periods, it is the hyper period.

(Refer Slide Time: 23:51)

Cyclic Scheduler Basics

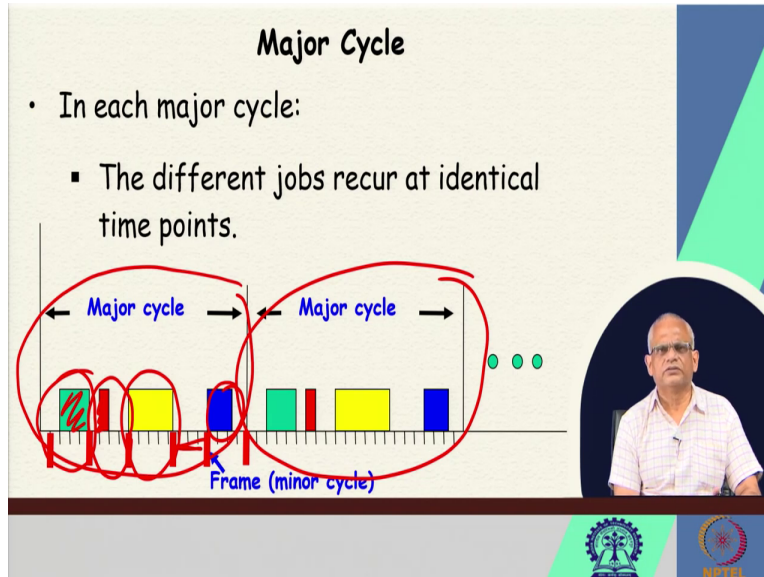
- If a schedule can not be found for the set of predefined jobs:
- Then these are divided into **job slices**.
 - Essentially, divide a job into a sequence of smaller jobs.



But it may so happen that we may not find a suitable schedule, we cannot find a frame to which task to assign, the system becomes un-schedulable and, in those cases, what we can do is the large jobs because see the constraint here is that every task must be assigned to one frame. We cannot assign a task to two frames. Because the scheduler here every frame, interrupted just looks at the schedule table and runs the next door.

And therefore, if some tasks are really large, the frame size becomes large and a lot of time is wasted because the smaller tasks they will also run on one frame and a lot of time will become wasted. The system will remain idle. In those situations, we divide the large jobs into job slices, and the system becomes schedulable. We will take some examples, and we will see how it is done.

(Refer Slide Time: 25:18)



And the major cycle is the schedule table size and we keep on repeating the same major cycle, just look at this diagram. This is the same major cycle is repeated. And in the first major cycle, see here, the frames occur here, the frames are shown as the red lines here. This is the interrupt from a periodic timer and then it takes some time for the scheduler to run its own by the width of this red. That is for the scheduler to decide which job to run, and then it runs.

So, the first one runs here, the first task, the second task. Sorry. So yeah, this is the frame. So, in the first frame, in the first major cycle, the green task runs, in the second frame the red task runs, in the third frame the yellow task runs and see the yellow task is almost taking the entire frame, I think that is the one based on which the frame size was decided. Because the frame size must be greater than all the task sizes and the yellow one, that is the largest task and the frame size is decided based on that. And then we have this green one and for one frame, we did not run any task.

And the same thing is repeated here. So, we just need to develop the schedule for one major cycle and how to develop the schedule for the major cycle that is a topic for discussion now. But unfortunately, we are end up this lecture, we will stop here and in the next lecture, we will see how to design the minor cycle, how to assign the tasks to frames. And if the schedule becomes

infeasible, that we cannot really schedule the set of tasks then we will have to do slicing up the jobs. Those issues we will discuss in the next lecture, we will stop here. Thank you.