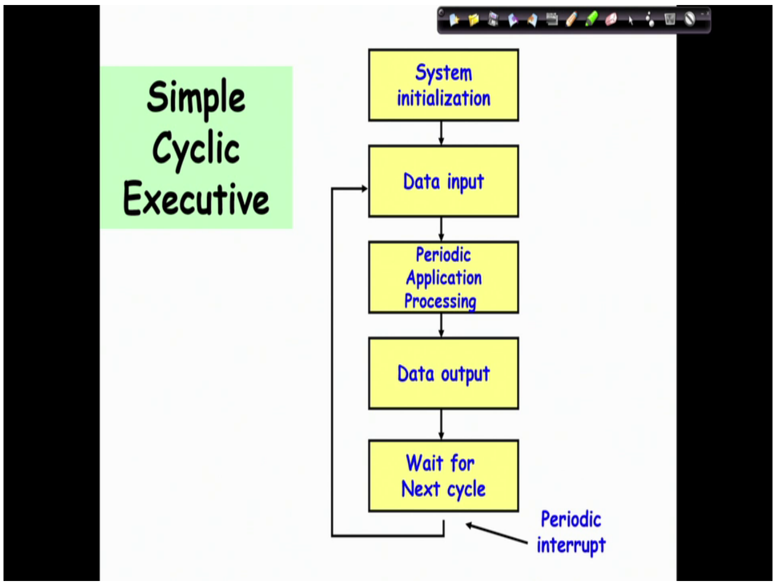
Real Time Operating System

Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 03
Cyclic Executives

Welcome everybody. So, far we had looked at some very basic issues in real time operating systems and we had seen that tasks scheduling is one of the major issues with real time operating system because the application deadlines are made with a help of a suitable tasks scheduler. So, now, we will start looking at some of the schedulers that are being used we will start with a simplest scheduler. It goes by the name cyclic executives.

(Refer Slide Time: 00:59)



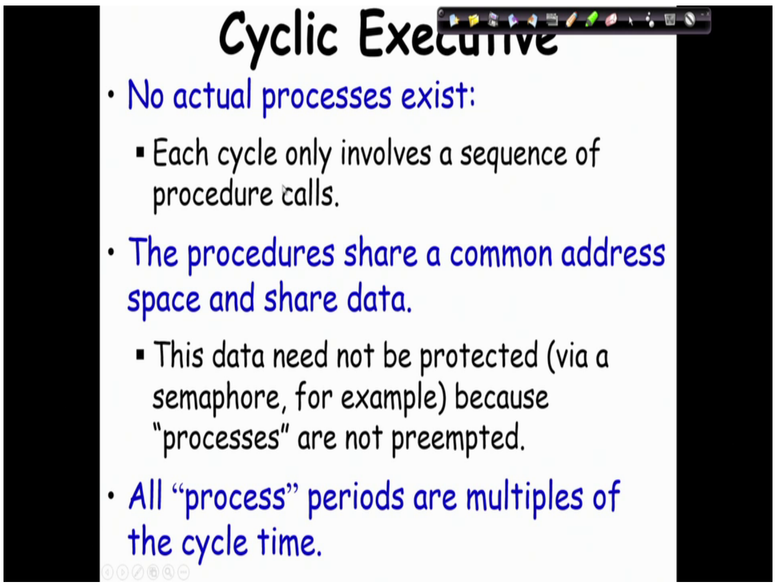
Simple Cyclic Executive

The cyclic executives are the simplest real time operating systems these are run on very simple embedded applications where there is a severe constraint and the processor capabilities may be a 4 bit or 8 bit processor and the memory is very very less. So, in this situation a full pledged operating system is difficult to use, the tasks are simple here and periodic in nature and this real time cyclic executives are basically very small programs. Let us look at the basic structure of a cyclic executive.

So, here initially to start with the system is started with a system initialization where various parameters are set and then there is a periodic timer which gives timer interrupt and it is a cyclic executive which starts a call to a program or executes a program where initially there is a data input, application processing and then data output and then it keeps on waiting until the next period comes. So, this is the simplest real time operating system it is basically a small program may be just a few dozens of line. And here as you can see that just to take an example of how it works is the data input let us say that we have a temperature controller.

In the temperature controller initially let us say the temperature has to be read from the sensor. So, that is the data input and then we need to check that is it below a threshold, higher than a threshold, is it abnormal etcetera and then based on that we possibly want to show the current temperature reading on a display. So, that is the data output. And then we wait for the next interrupt and the cyclic executive continues looping here infinitely each time waiting for the interrupt doing some simple processing. So, this is the simplest real time operating system run on the most elementary embedded systems.

(Refer Slide Time: 03:50)

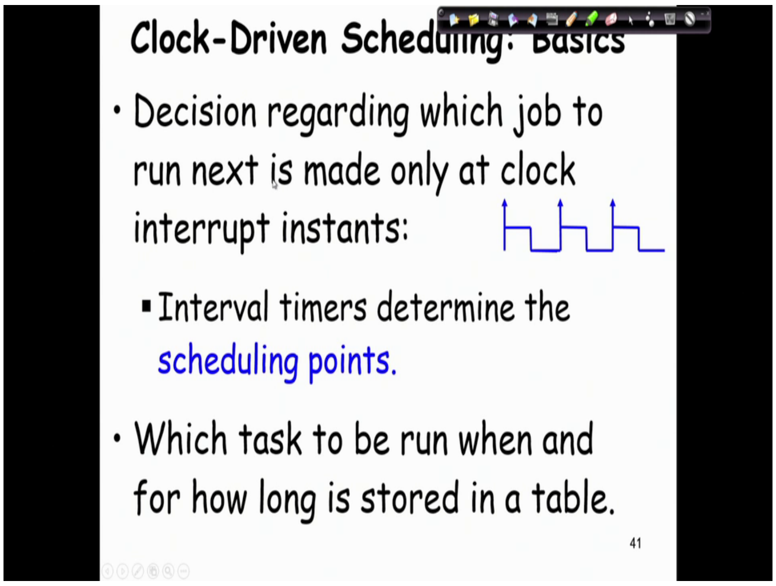


Cyclic Executive

- No actual processes exist:
  - Each cycle only involves a sequence of procedure calls.
- The procedures share a common address space and share data.
  - This data need not be protected (via a semaphore, for example) because "processes" are not preempted.
- All "process" periods are multiples of the cycle time.

One thing to note here in that, in this cyclic executives there are no processors it basically involves running program. A program initially samples some sensors then does some processing and then calls some display and here this program is executed number of times and each time there is a interrupt this program is run.

(Refer Slide Time: 04:31)



Clock-Driven Scheduling: Basics

- Decision regarding which job to run next is made only at clock interrupt instants:
  - Interval timers determine the scheduling points.
- Which task to be run when and for how long is stored in a table.

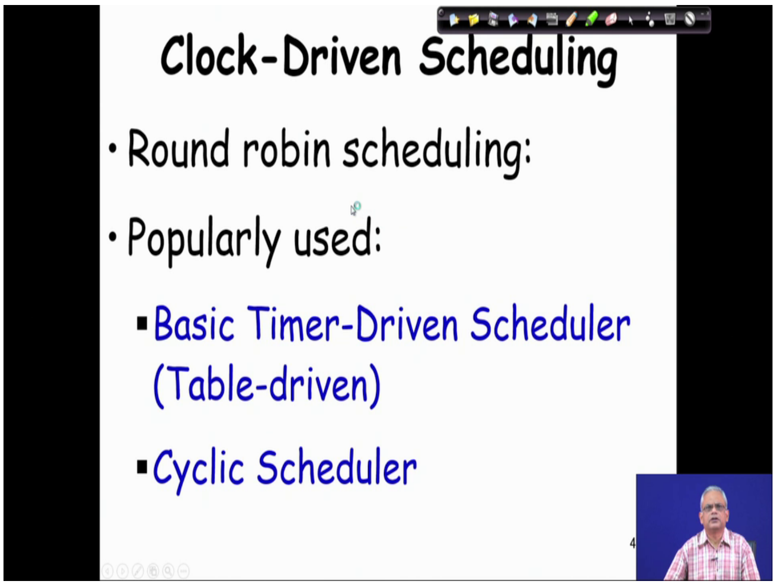
41

So, now let us look at more examples of operating systems which are sophisticated compared to the simple cyclic executive. So, all these schedulers that we are looking at are at the simplest end of the real time operating systems and these are called the clock driven schedulers. The ones that are much more sophisticated are the event driven schedulers.

So, we heard in the last discussion seen that there are two broad categories of real time task schedulers one is the clock driven schedulers the other one is the event driven schedulers. So, now, we are just mentioning that the simplest end of the real time operating systems we have the clock driven schedulers starting with simple cyclic executives. Now let us look at slightly more sophisticated clock driven schedulers.

So, all these clock driven schedulers the clock interrupt comes at regular intervals on the rising edge of the clock, the clock interrupts come and as soon as the clock interrupt comes the scheduler becomes active. So, these points at which the clock interrupts come these are called as the scheduling points. And in our last discussion we had mentioned that the instance at which the scheduler becomes active and checks what to do next are called as scheduling points and here the scheduling points are defined by a clock.

(Refer Slide Time: 06:27)



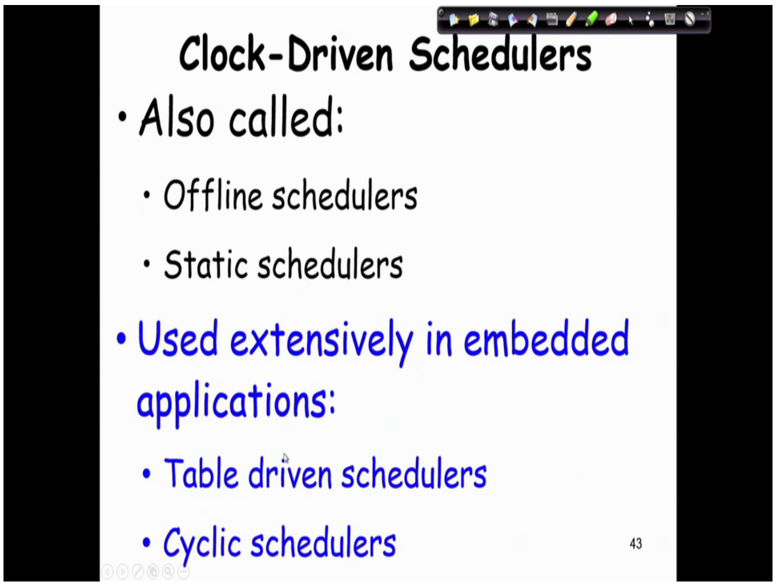
Clock-Driven Scheduling

- Round robin scheduling:
- Popularly used:
  - Basic Timer-Driven Scheduler (Table-driven)
  - Cyclic Scheduler

So, some of the important clock driven schedulers are the basic time driven schedulers which is called basic table driven scheduler which is also called as the timer driven scheduler and other is a cyclic scheduler. And even to think of it a simple round robin scheduling where based on clock interrupt the different tasks are run in a round robin fashion is also example of a clock driven scheduler, but we are not going to discuss about that that is used in traditional operating systems.

But as far as the real time operating systems are concerned we are now looking at the simple end of the operating systems the simplest operating systems used in small embedded applications looked at the cyclic executives now let us look at more sophisticated clock driven scheduler, first we will look at the table driven scheduler and then we will look at the cyclic scheduler.

(Refer Slide Time: 07:37)



Clock-Driven Schedulers

- Also called:
  - Offline schedulers
  - Static schedulers
- Used extensively in embedded applications:
  - Table driven schedulers
  - Cyclic schedulers

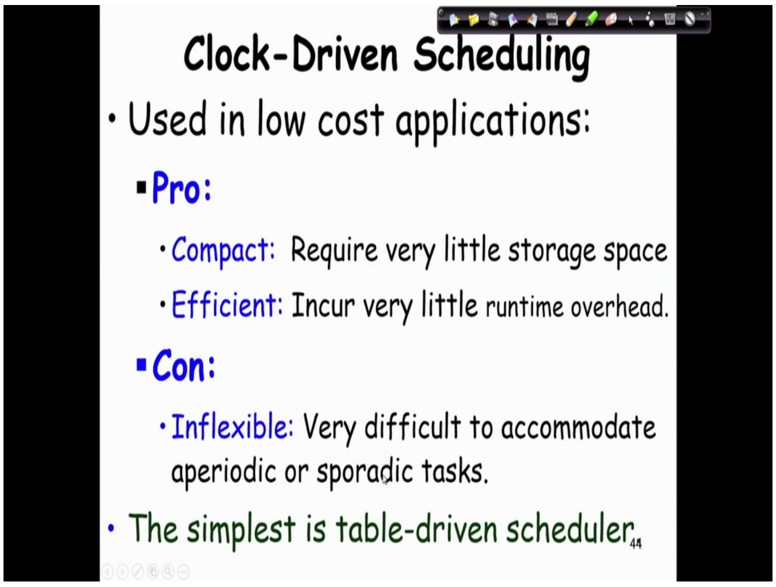
43

These clock driven schedulers are also called as offline schedulers or static schedulers because the tasks set to be handled is known beforehand, it cannot handle sporadic aperiodic tasks only the periodic tasks which are known beforehand whose period is known these are handled execution time and periods of course, these might start at different phases we had seen the concept of phase of a task.

So, these tasks may start with different phases, but then that is handled in by the clock driven scheduler the different phases But periodic and aperiodic tasks are difficult to handle and even the aperiodic tasks can be handled which do not have deadline and after completing the real time task the lax time that is available can be used for handling aperiodic tasks, but sporadic tasks cannot be handled by the clock driven schedulers.

So, first let us look at the table driven scheduler and then we will look at the cyclic scheduler these are used extensively in embedded applications.

(Refer Slide Time: 09:03)



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 aperiodic or sporadic tasks.
- The simplest is table-driven scheduler.

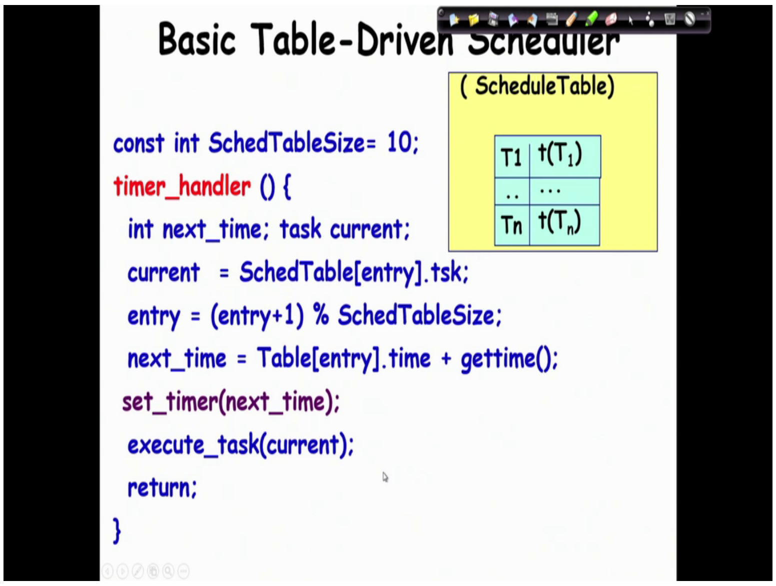
44

As we mentioned that, these are used in low cost embedded applications having very less processing power and very small memory which cannot host full blown, full pledged operating systems.

So, the advantage of these are, these take very small code space very little storage the code size is small and even the temporary storage requirement is very small. And these are very small programs run for a few microseconds just few lines of code and incur very less runtime overhead. When the processing power is small if you want to run a sophisticated operating system then the overhead of the operating system will be so much that there will hardly any time left for running the real time tasks.

But then you will these have the advantage of running efficiently and with less space on a embedded application, but then these have their set comings which we must be aware of. The important set comings of these are that these are difficult to accommodate aperiodic and sporadic task. And now let us look at the table driven scheduler.

(Refer Slide Time: 10:29)



Basic Table-Driven Scheduler

(ScheduleTable)

T1	$t(T_1)$
..	...
Tn	$t(T_n)$

```
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;
}
```

In the basic table driven scheduler we have the operating system stores a table in the memory this table basically contains only two parameters in the simplest form that what are the tasks and what are the time for which it will run. The tasks may not be real tasks actually whenever you are calling tasks it may be just invoking or running certain programs because handling tasks require sophistication and the part operating system.

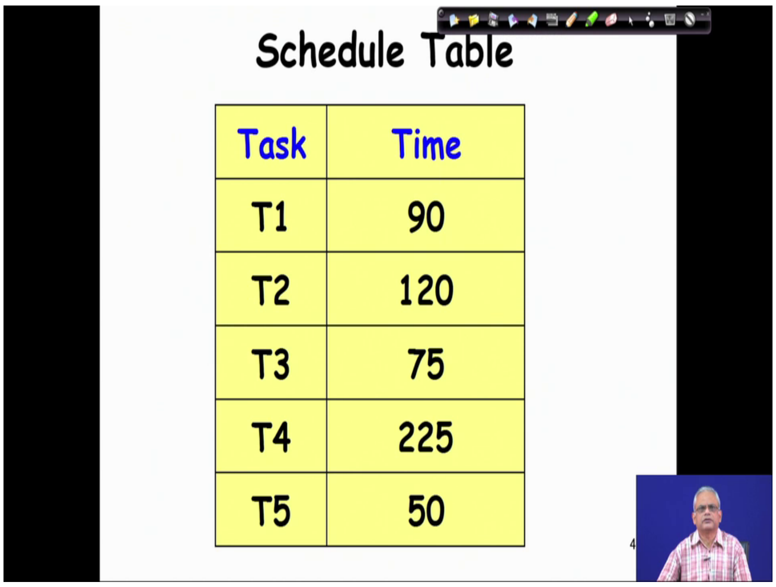
The task table creating task a tasks state and so on as we know from our knowledge from a basic operating system task handling requires lot of sophistication on the part of a operating system and we are now discussing the simplest real time operating system and here even though we use the term tasks, but then these may be just running a program is may not be task in the true sense of a task in a operating system. So, here in the basic table driven scheduler we have the different tasks and their maximum runtime and then we have a periodic interrupt from a timer and as the timer interrupt comes the timer handler routine is invoked.

The timer handler routine keeps track of which is the task that is running now and that is given by the entry. So, it just looks into the schedule table and finds out which task to run and just finds out the task some parameters of the task which tell how to execute name of the program and so on name of the executable which is to be executed. And then the entry is augmented to point to the next entry in the table and it is mode of the schedule table size because it just keeps on executing that.

So, the next time is given by the table entry time that get time when the current time plus the time that is required by the task. So, here the timer is set for the next time and that is the time that the task takes. So, from the current time the task is started executing. So, this is the one to start execution of the current task and the timer is set for the maximum duration of the task. And then as the timer expires the timer handler is invoked and the next task is taken out from the table and the pointer to the table is incremented, the entry that is the next task to be taken is incremented and then the time is set for the timer and the task is taken up for execution and so on.

So, here as you can see in a basic table driven scheduler we have a simple data structure that is stored in the memory and their code is small small and it just keeps on executing this schedule. So, this is also not a sophisticated operating system used for simple applications, but slightly more sophisticated than the simplest cyclic executives.

(Refer Slide Time: 14:59)

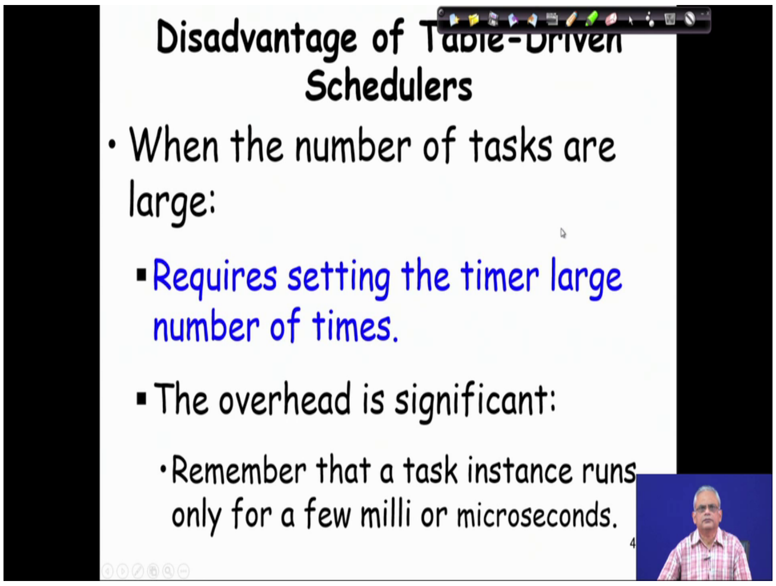


Schedule Table

Task	Time
T1	90
T2	120
T3	75
T4	225
T5	50

So, here is an example of a schedule table. These are the tasks or the programs to run and maximum execution time is mentioned in milliseconds and the timer is set, and just observe here that the tasks are typically the time is in milliseconds. So, the scheduler overhead must be much smaller than the task execution time if the task execution time is ninety milliseconds and the scheduler itself takes 100 millisecond then it is not a good idea. So, here the cyclic schedulers are very efficient run in just a millisecond or so, and then they just set the timer access the table and then they start the execution.

(Refer Slide Time: 16:04)



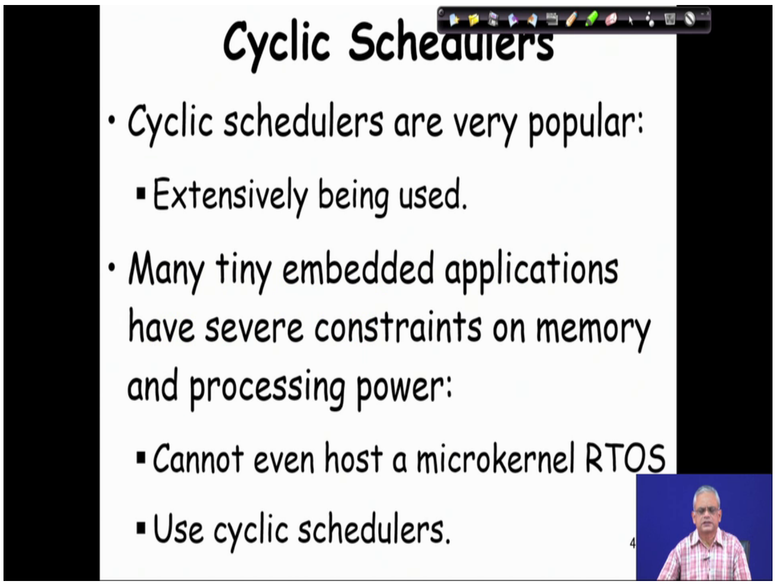
Disadvantage of Table-Driven Schedulers

- 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.

But one thing that we must notice here is that in a table driven scheduler we need to set the timer a large number of times and setting a timer is actually is rather expensive task compared to a time it takes for running a task and also the scheduler overhead, a major component of that in setting the timer the other things like incrementing the entry count pointer etcetera these do not take much time large amount of time is taken in stetting the timer. So, this overhead is significant.

And the next scheduler that we are going to look at is bit more sophisticated than a simple table driven scheduler. But one important advantage of the cyclic scheduler is that you do not have to set a timer again and again timer is set only once and the overhead due to setting timer is largely overcome in a cyclic scheduler let us look at the cyclic scheduler.

(Refer Slide Time: 17:27)

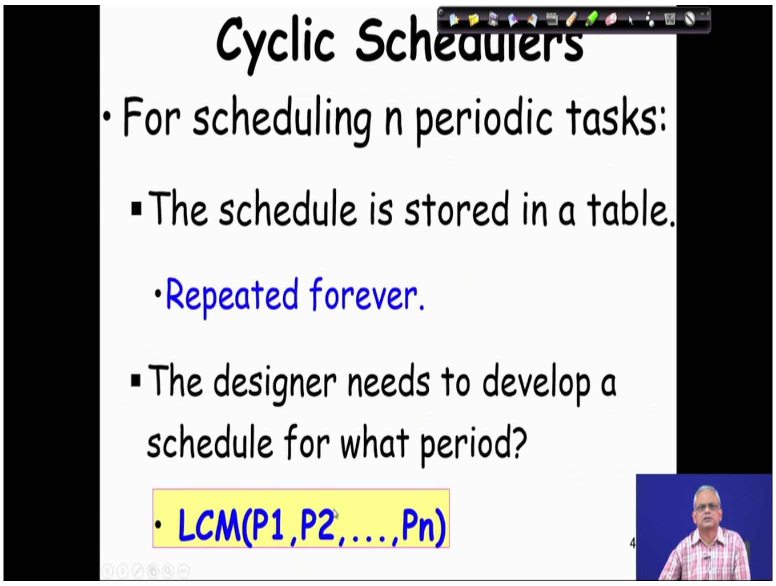


Cyclic Schedulers

- 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.

The cyclic schedulers are very popular used in embedded applications. These are also for meant for simple applications where there is not enough memory and processing power to run a full blown event driven. Real time operating system for example, a micro kernel real time operating system which will see later they take several even a micro kernel operating system may take several kilobytes of memory and requires running several thousands of lines of code each time a scheduling point occurs. But here the cyclic scheduler is very simple it requires running only few lines of code each time a new task is to be taken up for scheduling.

(Refer Slide Time: 18:32)



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(P1,P2,...,Pn)

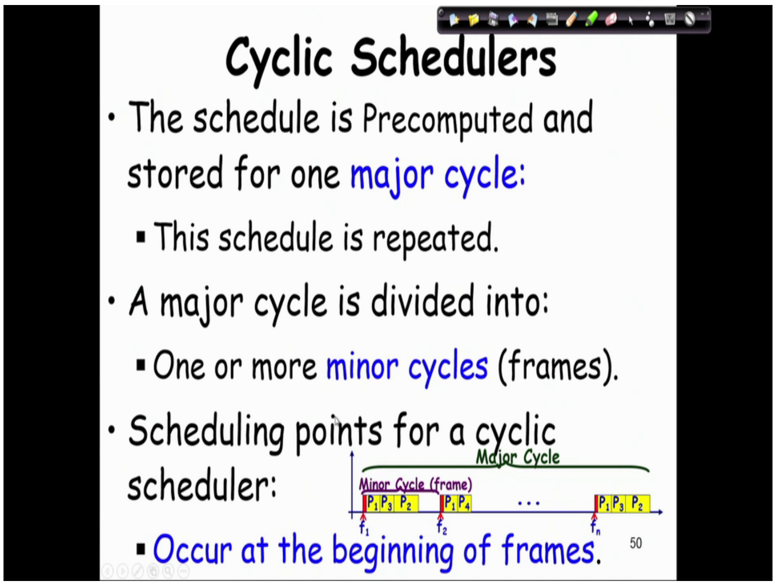
So, let us look at the cyclic scheduler. So, here again let me just mention that even though we are saying tasks, but these may not be real tasks because these are used in simple real time operating system. Here we do not have capability to manage tasks these are actually simple programs that are stored and the scheduler just starts executing these programs these are not tasks in the actual sense of the task as used in the operating system literature, but that for uniformity of treatment occurs different schedulers we are using the term task, but we must remember these are not tasks in the real sense.

So, here if we have n periodic tasks and each task requires running at different period. So, it seems that in the simple cyclic executive there was only one period, but here there are multiple periods. So, which are you called as multi rate operating system. In multi rate we have multiple tasks which require running at different periods some may be requiring every 100 millisecond some may be 500 millisecond some may be 2000 millisecond etcetera. So, the schedule here is stored in a table as in the case of other cyclic scheduler that the table driven scheduler which is repeated forever, but we will see that it overcomes setting a timer again and again.

So, we have n periodic tasks then let me ask this question that to repeat the schedule what should be the period of repetition. Let me rephrase that if we want to develop a schedule for this n periodic tasks each task requiring running at different time interval some may be 100 some may be 250 some may be 300 millisecond etcetera. So, what is the maximum length for which you need to create the schedule and store it in a table and then we keep on repeating it forever.

So, the answer to that is LCM. So, we take all the periods and then compute the least common multiple. So, if we have 100 300 and let us say 400 then we need to store the period the schedule for a period of 1200 millisecond. So, 100, 200 and, sorry 100, 300 and 400 the LCM is 1200 and therefore, we need to maintain the schedule for 1200 millisecond and then keep on repeating that.

(Refer Slide Time: 22:00)



Cyclic Schedulers

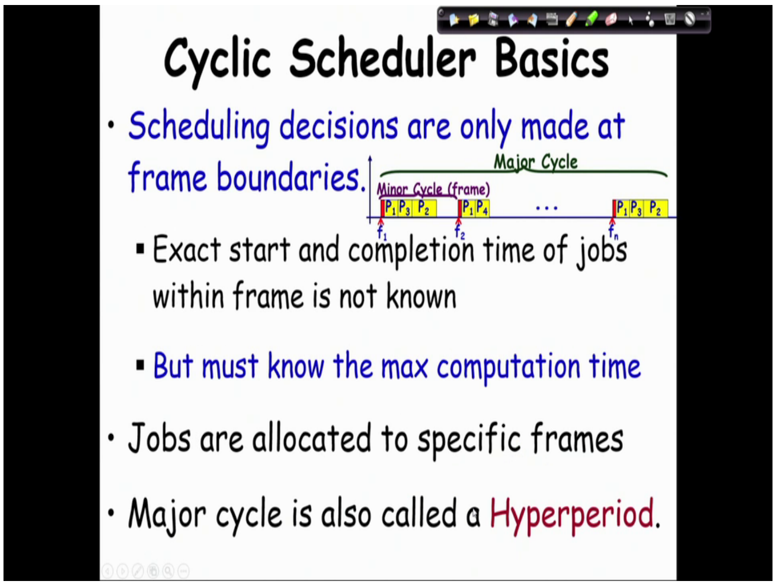
- The schedule is Precomputed and stored for one major cycle:
  - This schedule is repeated.
- A major cycle is divided into:
  - One or more minor cycles (frames).
- Scheduling points for a cyclic scheduler:
  - Occur at the beginning of frames.

So, this 1200 millisecond or the maximum period for which we need to maintain the schedule is called as a major cycle. So, after every major cycle we repeat the schedule. But if the major cycle is divided into many minor cycles the minor cycles are also called as frames.

So, as you can see these lines here, these are basically clock interrupts and these define the frames this is the periodic timer. So, it have needs to be set only once and then keeps on giving interrupts that these interval defining the frames. And to each frame a task is assigned and that is stored in a table and these points at which the frames start the frame boundaries, these are the scheduling points, the cyclic scheduler becomes active at this scheduling points and then decides which task to run and then the table is stored for one major cycle and then the task execution is repeated.

So, the scheduling points define the frames the beginning of the frames and these are set by a periodic timer which keeps on giving interrupts at regular intervals.

(Refer Slide Time: 24:03)



Cyclic Scheduler Basics

- Scheduling decisions are only made at frame boundaries.
  - Exact start and completion time of jobs within frame is not known
  - But must know the max computation time
- Jobs are allocated to specific frames
- Major cycle is also called a Hyperperiod.

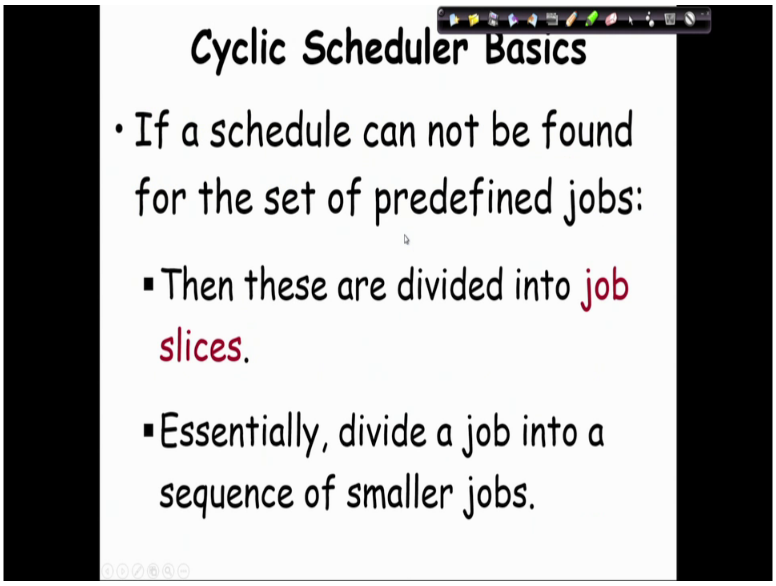
Major Cycle
Minor Cycle (frame)
P_1 P_3 P_2
P_1 P_4
...
P_1 P_3 P_2
f_1
f_2
f_n

And each time there is interrupt in the periodic timer, the cyclic scheduler is activated, it starts running and then consults schedule table finds the next task to run and then after it completes it may idol for some time until the next interrupt from the periodic timer comes.

So, one important thing to notice is that compared to a table driven scheduler where the timer was set again and again requiring much overhead here that is overcome. The periodic timer is start only once during the initialization of the running of the system. Now, the important thing that we must understand is that how is the frame time set for a given task set and how are task assigned to frames. So, the task instances are jobs are assigned to specific frames and these tasks have different periods. So, may be that in one major cycle or a hyper period major cycle is also called as hyper period may be in one hyper period some task instances that is jobs there are multiple times some jobs run only once etcetera.

So, as far as the cyclic schedulers are concerned one important question to answer is that how to fix the frame duration and how to assign tasks to the frames.

(Refer Slide Time: 26:03)



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 as we take examples and explain the methodology we will find that it may be possible that with a given set of tasks and task periods we may not be able to find a frame size and assign tasks to frames. So, then we need to divide some tasks into smaller jobs or job slices and then retry the methodology. So, compared to the simple cyclic executive and the table driven scheduler here for the system programmer who is trying to run a set of real time tasks on a cyclic scheduler the development of the schedule is non-trivial.

We will examine what is involved in setting up a schedule in fixing the frame time what are the factors to be considered and given a task set how does one go about doing it will do a number of examples. And we will also request you to try out few examples on your own so that you understand the integrity of developing a schedule in a simple cyclic scheduler, so that we will continue it next lecture, now we will stop.

Thank you.