Real - Time Systems

Prof. Dr. Rajib Mall

Department of Computer Science and Engineering

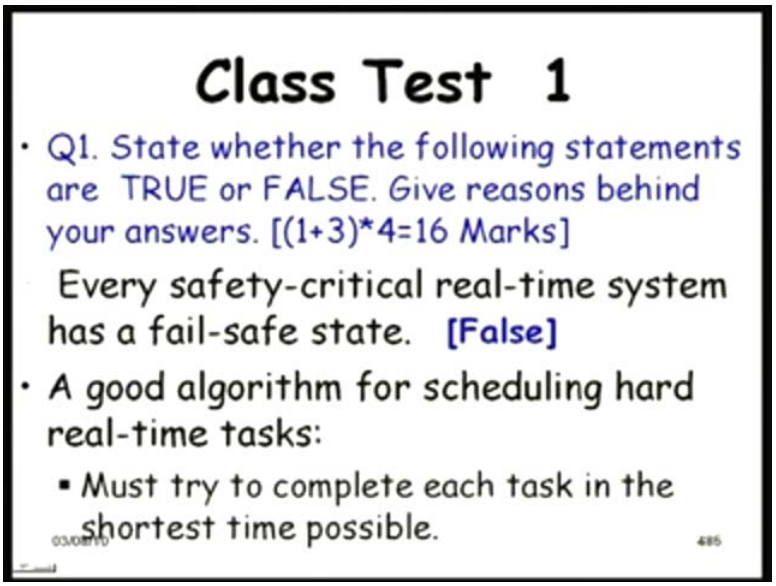
Indian Institute of Technology, Kharagpur

Lecture No. # 22

Tutorial – I

Good morning. So, let us get started. So, today, we will just discuss the class test and the mid sem; the answers that were expected. So, the answer scripts have been distributed. So, let us see the questions and the answers, and then, we will proceed from what we are doing last time.

(Refer Slide Time: 00:50)



So, let us look at the questions that were there in the class test. So, displayed here; check your answers with respect to the answers that we discuss and arrive at, and then, see if these are marked alright. So, I hope everybody has got their answer scripts for both the class test and the mid sem. So, the first question in the class test was a TRUE and FALSE question. There were four parts, each one having 4 marks.

So, the first, so, there are basically some statements and you will have to identify whether this statements are true or false and then give a reasoning. So, what about the

first one? Every safety-critical real-time system has a fail-safe state. What do you think? Do you agree with this or disagree? Is it TRUE or FALSE?

False

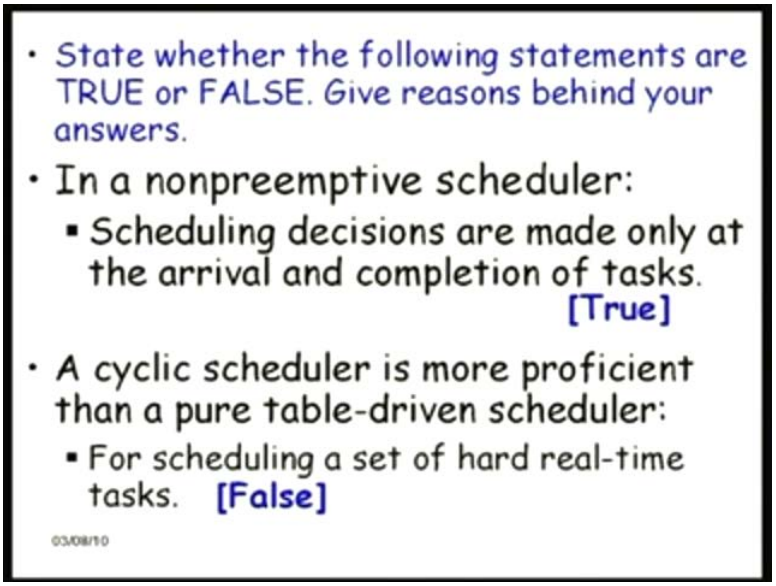
Yeah, false, because even if it is safety critical, there is no guarantee that there will be a fail-safe state. For example, if it is a aircraft is safety critical, live's of people are dependent on it, do not have a fail-safe state. Anybody has any difficulties with this answer? So, the answer is false; check your answers.

The second statement was that a good algorithm, a good scheduling algorithm for hard real-time tasks: Must try to complete each task in the shortest time possible. What do you think? Is it true or false?

False

It is false, because there is no advantage in finishing the task, the earliest possible time. It is good enough if all the tasks meet their deadline, and there is no special reward if you complete the tasks in the shortest time, and other hand, for a traditional scheduling algorithm, the task throughput is a important criterion. They are finishing it at the shortest time as a revert; here, there is no revert. So, the answer is false.

(Refer Slide Time: 03:18)



Now, let us look at the next part. In a non-preemptive scheduler: the scheduling points or the scheduling decisions are made only at the arrival and completion of tasks.

What do you think? True or false?

True

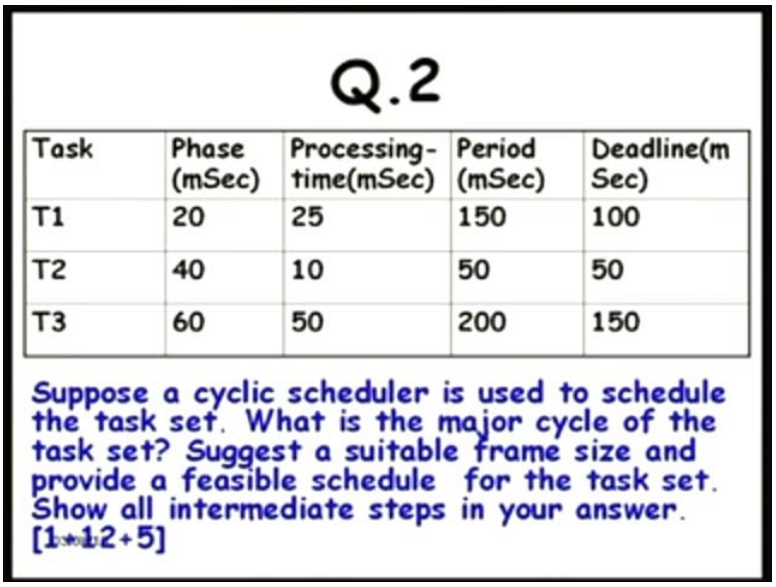
Yeah, it is true, because when a task completes, need to decide which is a next task that needs to run, and even when a task arrives, there are no task running. Vven if it is non preemptive to, if there is no tasks running, you will have to decide whether to take this out. So, this is a true answer. Then, let us look at the next part. A cyclic scheduler is more proficient than a pure table-driven scheduler for scheduling hard real-time tasks.

What do you think? Is it true or false? A cyclic scheduler is more proficient for hard real-time tasks. What is your answer? What do you think? True or false? Yes, what do you think? Is it cyclic is more proficient or table-driven scheduler is more proficient? I hope you remember what is (( )) proficient.

(( ))

Exactly table driven is more proficient, because it does not waste any time, whereas in a cyclic scheduler, part of every frame almost every frame gets wasted. So, something which may not be possible to schedule using a cycle driven, a cyclic scheduler may be possible to find a schedule in the table-driven scheduler. So, the answer is false. Check your answers whether they have been marked properly. If some answers are not marked properly at the end of this class, we will just correct the mistakes if there are any.

(Refer Slide Time: 05:21)



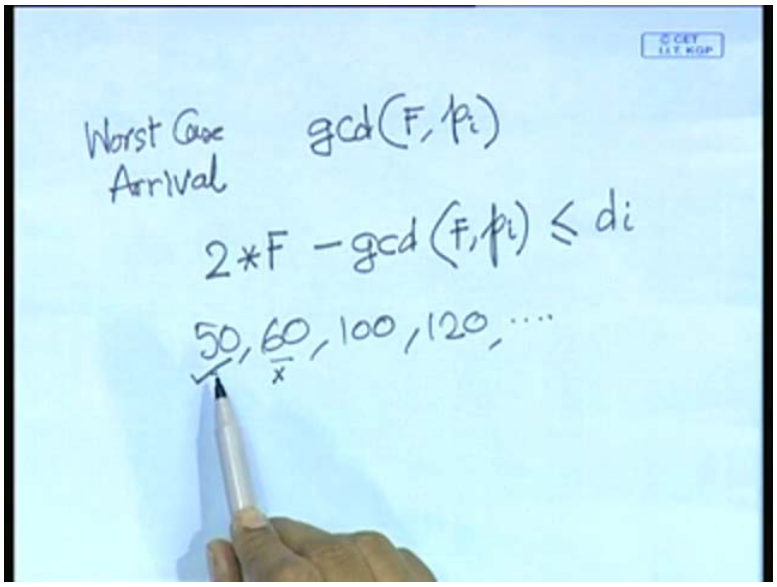
Now, let us look at the question two which had a 1 plus 12 plus 5 is 18 marks for this question. Three tasks were given, three task characteristics, three periodic tasks. Phases were given for the three tasks. Processing time required for the three tasks were given. The period for the three tasks were given and the deadlines were given and a cyclic scheduler is used to schedule the task set.

The first part of the question is what is the major cycle of the task set? So, how do you do the major cycle? You look at the periods and find the l c m for the periods, is not it. So, what will be the L C M? It will be 600 secure answer if it is 600 must get this one mark. Suggest a suitable frame size and provide a feasible schedule for the task set.

So, the first, the second part is suggest a suitable frame size and we know that a frame size in a cyclic scheduler must divide the L C M must be integral number of frames in the major cycle and also it must be, the frame size must be larger than the largest execution time of a task. So, we get that it has to be greater than 50 greater than equal to 50 and the possibilities are 50, 60, 100, 120 etcetera.

Then, you need to try out whether a task will meet its deadline; all the tasks will meet their deadlines in the worst situation and the worst situation we had derived a formula for that, that is 2 into f minus g c d of f comma p i, something like that should be than d i is not it?

(Refer Slide Time: 07:51)



So, we had derived a formula that the worst case occurs for a task. The worst case g c d F, P i is the worst case arrival for a task, and then, for the task to meet its dead line, we must have 2 into f minus g c d F, P i should be less than equal to d I, this is the one expression I had derived.

Now, you have to try out the frame sizes, and the largest frame size is suitable, because why is largest frame size suitable? What is the harm in choosing the smallest frame size? Because see, we said that the options are 50, 60, 100, 120 etcetera. So, why is the largest frame size?

(( )).

Exactly, that is correct. We have to select the largest frame size because that will have minimum overhead.

So, you can try out here from 50, 60 etcetera. So, I think as I can calculate that 60 is not possible. So, the largest is 50; just check your answer. So, once you have got the frame size finding the schedule according to the phase, the first arrival is not difficult. Do you agree with that or so we do that?

Why, we can take fifty only know, why you want to take the highest?

Because if you have a higher frame size, see the, let me repeat the question. The question is that once we find that 50 is a feasible, it can, is a feasible frame size, why do we need to check for 60 before we take 50? Answer is that we should check for a higher frame size in case that is also feasible. We should choose the higher one, the higher one, because you know that in a cyclic scheduler, we have a clock that the cyclic a periodic interrupt from a timer arrives and that time it decides. So, if we have a larger frame size, then the timer gives less frequent interrupts.

What to do similarly are you choosing the frame size bigger than all other individual lines.

Yes.

So, it does not do not need a higher time (( )) number of times they switch across.

(( )) address this question the the question is that. See as long as the task set runs, the hard real-time task set runs, why do you worry whether it is a higher or lower frame size? Is that your question?

No sir. Actually, if we are choosing the 50, which is the higher than the (( )) yes.

Individual lines of each of the tasks.

So, the number of switches will be same even if we take a higher frame size.

No, no, no, see, let us just, see he says that number of switches will be same. See, the answer the question, the answer that i am giving please listen to it, is that depending on the frame size chosen whether it is 50 or 60, a timer will be set, periodic timer will be set. So, if you set the periodic timer at 50, you will get more frequent timer interrupts. If you set at 50, you will get less frequent interrupts, is not it? And you need less frequent interrupts.

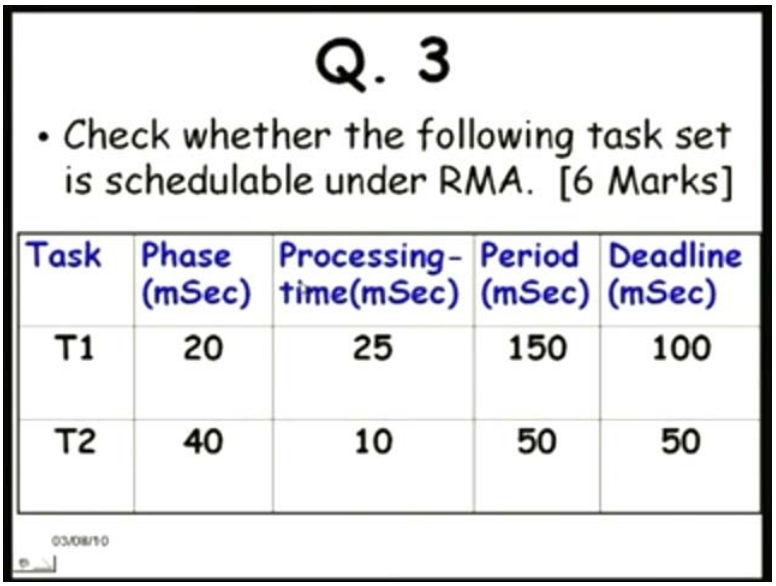
But in within a frame, the space is wasted.

It is not wasted. See, if we, if you look at the extensions we had done, see, this is for running the hard tasks; there will also be the soft real-time tasks. So, even if these appear

to have been wasted, it is not really so, because they will be taken up by the soft real-time tasks. So, just revise the things that we did.

So, the higher frame size is preferable, but here 50 turns out to be the highest frame size, that is feasible; it can be feasibly be used. So, let us proceed. You check your corrections with respect to this answer.

(Refer Slide Time: 12:33)

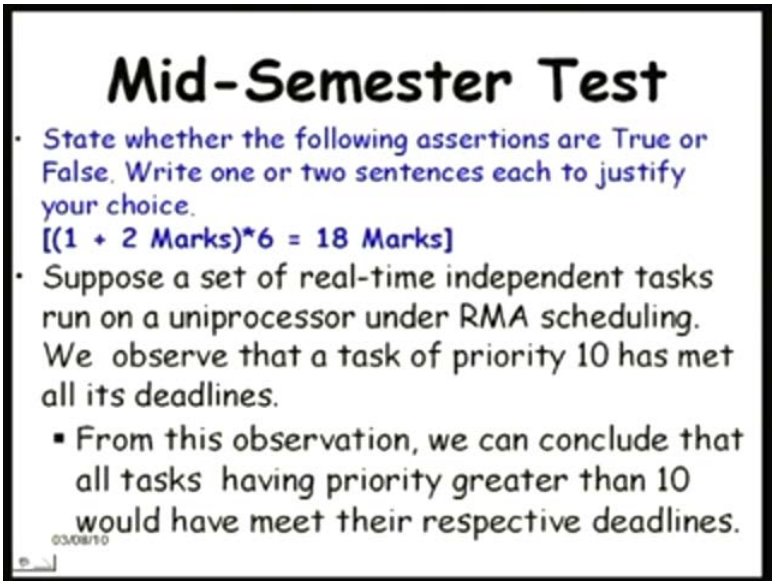


Now, let us look at the next question. See, there are two tasks given. So, this is for the class test one, the third question having 6 marks. Two task characteristics are given - T1, T2 phase of 20, 40 and there are some execution time - 25 and 10, and the periods are 150 and 50 deadline is 100 and 50. Now, you have to check whether the task set is schedulable under RMA, straight forward question. Basically we will have to check whether the utilization due to tasks is less than one and also whether it satisfies the liu layland criterion which is the pessimistic criterion. If it satisfies, then you do not have to check further.

So, if you compute the utilization, I have not worked out. If anybody wants, you can work out, but it is straight forward. The utilization if you check, it will satisfy, and the phase has nothing will not be used here. For a liu layland criteria, the phase is only there and in we need to ignore phase while using the liu layland criteria, because that itself is a pessimistic result; phase is inconsequential, and even for liu lehoczky's, we do not use

the phase, set it to be 0. So, that also checks for a worst case situation and actual 20, 40 will be if you check the phase, it will be very difficult to derive a formula. Closed for formula which will check for a given phasing whether the task set will run? Is there any difficulty with this question?

(Refer Slide Time: 14:32)



So, let us move to the Mid-Semester questions. The first question was again giving some statements, some statements are given and you have to identify whether these are true or false and then each statement 3 marks and there were 6 parts here, 6 statements were given total of 18 marks. The first statement was something like this that suppose a set of real-time independent tasks run on a uniprocessor under RMA scheduling, and then, we observe that a task of priority 10 has met all its deadlines. From this observation, can we conclude that all tasks having priority greater than 10 would have met their respective deadlines?

What do you think? Yes or no? True or false? Given that the task priority 10 has met all his deadline will all tasks having larger priority or higher priority 11, 12, 15 etcetera, they would have also met their deadline.

Is it then deadlines are different if they are having different phases?

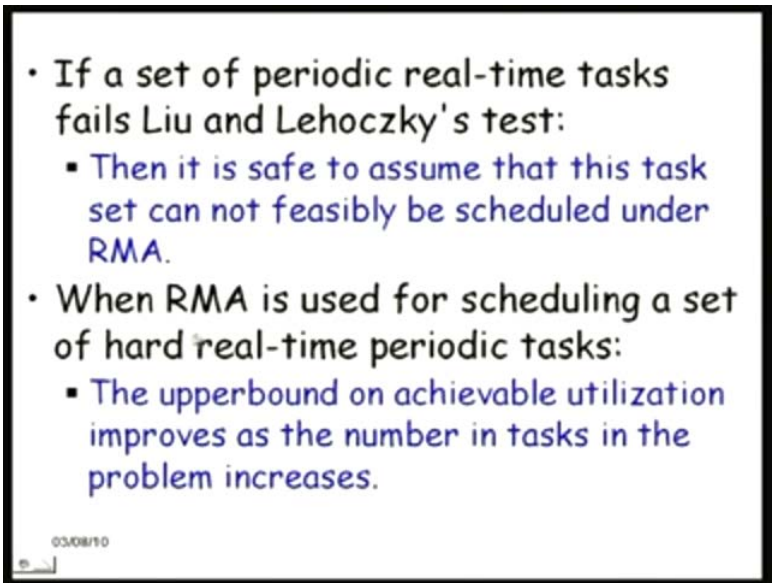
No, do not complicate the answer. See, the answer is very straight forward. See, look at the RMA here.

(( )).

The RMA here it says that see, as long as task priority is higher, it will preempt the current running task. So, it is never possible that a higher priority task will keep on waiting the lower priority task runs. So, it would not be possible at all.

If a low priority task has met, met, it all its deadline, we can safely conclude that higher priority tasks will run will meet their deadlines, respective deadlines, but possibly the only ambiguity that was there in this question is that we did not mention here higher priority, priority, greater than ten means higher priority, because in some operating system, a larger priority value indicates a lower priority. So, possibly that is the ambiguity in this question, I forgotten at that part while setting the question. Anybody has written about that or written that in that connotation will get mark.

(Refer Slide Time: 17:10)



If a set of this is the next statement that was given, if a set of periodic real-time tasks fails the Liu and Lehoczky's test: Then it would be safe to assume that this task set cannot be feasibly scheduled under RMA. So, basically the task set has failed the Liu Lehoczky's test. Then the claim is that we cannot schedule it under RMA.

False

It is false, because Liu Lehoczky's test and assumes zero phasing of the tasks and zero phasing is the worst case arrival of the task. All tasks arrive together, that is the worst case situation. So, answer will be false.

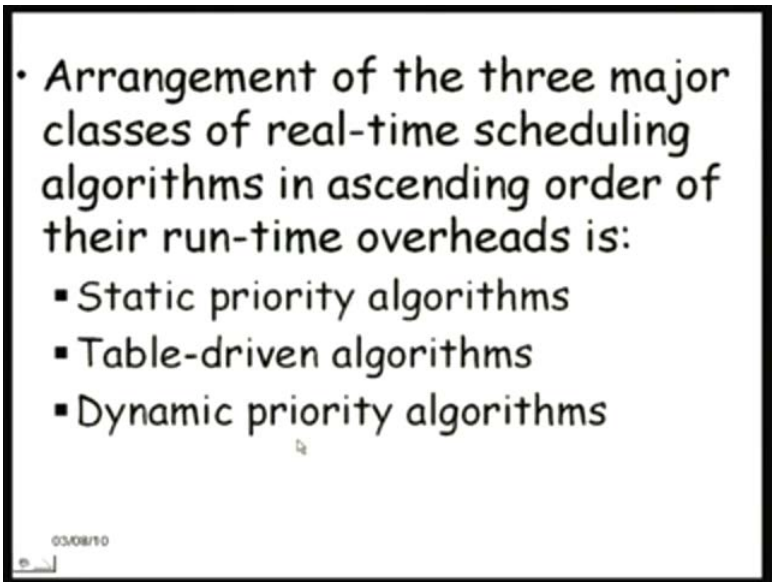
Now, next statement: when RMA is used for scheduling a set of hard real-time periodic tasks: the upper bound on achievable utilization improves as the number of tasks in the problem increases.

False.

What about you? Do you agree it is false?

Yeah, it is false, because the utilization actually decreases with the number of tasks and then it remains constant, is not it? 0.69 as the number of task approaches infinity.

(Refer Slide Time: 18:38)



Now, the next part of the question was an arrangement of the three major classes of real-time scheduling algorithms in ascending order of the run time overhead is: Static priority algorithms, Table-driven algorithms, and Dynamic priority algorithms. So, do you think it is a correct ordering in terms of increasing overhead run time overhead?

What do you think? Is it a correct ordering or not?

(( )).

Yeah. So, run time overhead if you look at it, the run time overhead is basically the decision and the frequency of decision, the time taken to arrive at a decision and a frequency of decision making that is all is a run time overhead. And in table driven algorithm, the decision is not there; it just run the task, that is there in the table; decision time is almost its constant its.

So, we have (( )) does it not count.

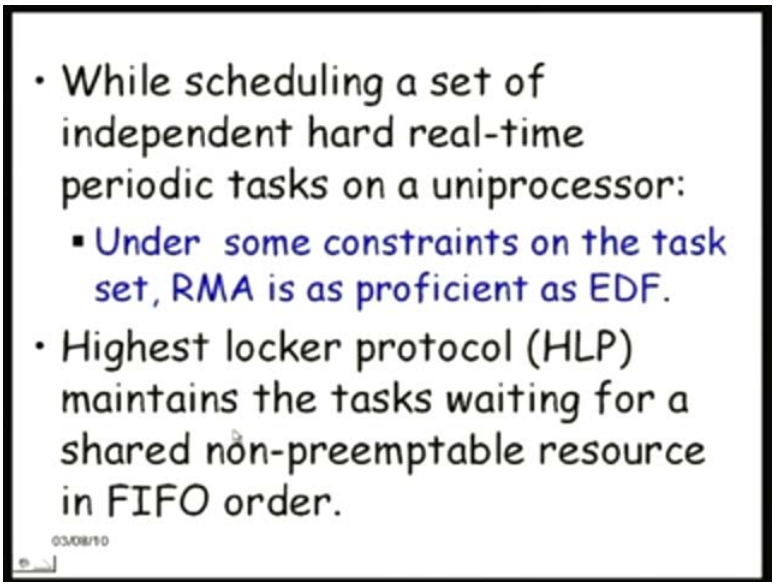
See, setting a clock in a table driven, yes, it is a, each time we will have to set the clock, that is also constant. It does not depend on the, where as in a static priority algorithm, each time we will have to checkout which, which, is the highest priority task in that time and also it number of context switches will be higher in a static priority algorithm. More number of times, it will run; each time you need to preempt and so on.

So, static priority has more overhead than a table driven algorithm and…

Same we have come from table driven just because of the overhead of setting a clock (( )).

Not really, see, if you please revise your this thing, the first lectures, we had said that the table-driven algorithm is good for small applications, its efficient and good, but the only difficulty with this is that it cannot really handle dynamically arriving tasks and the dynamic priority algorithms off course these are proficient, but they have large run time overhead.

(Refer Slide Time: 21:11)



Now, let us look at the next bit of the question. While scheduling a set of independent hard real-time periodic tasks on a uniprocessor, so, we have set of hard real-time tasks to be run on a uniprocessor, under some constraints on the task set, RMA is as proficient as EDF.

True.

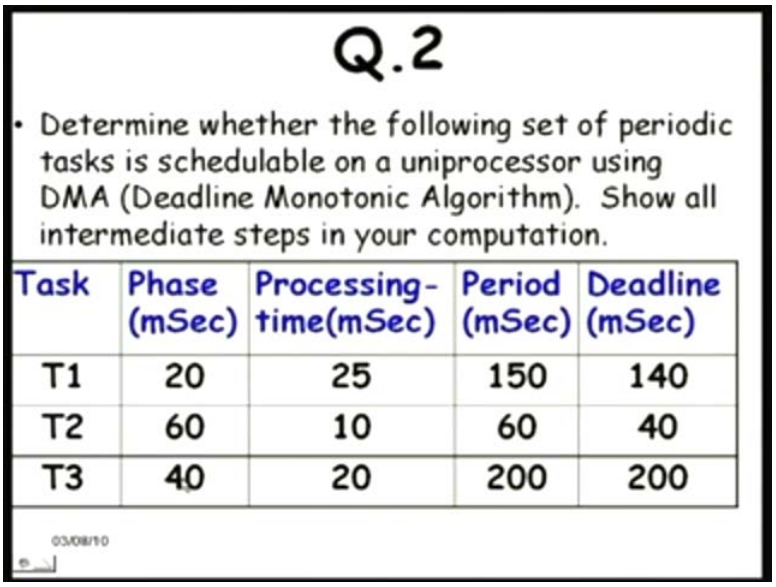
Yes, it is true as some of you are saying that if the…

Harmonically related

Yeah, if the periods of the tasks are harmonically related under that constraint on the task set, both of them will achieve a equalization of one. So, both will be equally proficient. Now, the next statement was a highest. The highest locker protocol for resource sharing real-time tasks would maintain the tasks waiting for a shared non-preemptable resource in a FIFO order.

What, what, do you think about this statement? It is false. It does not require how the tasks keep on waiting. The highest locker protocol just deals with how to set the priority computing the priority ceiling and setting the based on the resources and then the inheritance scheme, but how the waiting tasks that depends on the implementation? This is a possible implementation, but not necessarily the only implementation.

(Refer Slide Time: 22:51)



Now, let us look at the next question. So, this is a, to determine whether the following set of periodic tasks is schedulable on a uniprocessor using the Deadline Monotonic Algorithm, and then, so, the intermediate steps, three tasks are given - T1, T2, T3; phases are given; processing time is given and the periods and deadlines are given.

So, how do we proceed with this problem?

The second priority and…

Exactly. So, as he says that here the in a deadline monotonic algorithm, the priority will be assigned based on the deadline unlike the rate monotonic algorithm. The first one is to assign the priorities to tasks. T2 will be the highest priority. T1 is the second highest priority. T3 is the lowest priority task here, and based on that, what do it do?

After assigning the priority, what do it do?

Check whether (( )).

How, how do we check?

(( ))

Can we use the Liu Layland criterion? Yes. Can we use the Liu Layland criterion? Not really, cannot use really the Liu Layland criterion, because the Liu Layland criterion is based on the period if you check. So, so you can check for each of the tasks. One is you can consider the lower of the deadline and period and compute the bound on that or you can use the Liu Lehoczky's criterion where the first task will meet its deadline, because 40 is less than 60. The second priority task is 140 is, the period is 150. Within that, T2 will arrive 3 times.

So, all the 3 times this needs to run. The T2 has to run all the three times, because that is higher priority according to DMA. So, 3 into 40 is 120 is eaten up by, no, sorry, 3 into 10 is 30 is taken up by T2, and then, it will get 25. So, 30 plus 25 is 55 is less than 150. It will also run. The third one T3 within 200, T2 will arrive 3 times; T2, T1 will arrive 2 times. So, 3 into 10 is 30 plus 2 into 25 is 50, 50 plus 30 is 80 plus 20 is 100 less than 200. This will also run.

Is that or should I work it out?

Sir.

Yes please.

Sir you said in one time you said that if a, if a task set is schedulable on RMA.

Yes.

Then it should be schedulable on DMA also

No, we did not say. Please revise your, I mean the notes. What we said is that if the deadline is less than the period, then DMA is more proficient than RMA. That we had discussed when we discussed in the context of DMA, please revise your notes.

So, RMA might fail for when the deadline and period are not same. See, when the deadline and period are same, RMA is the same as DMA, but if they are different, DMA is more proficient algorithm than RMA. So, please look back your notes. Let us go to the next problem.

Sir one more question.

Yeah please.

When we start sir, T2 has the same period as, as, its phase. So, while actually scheduling, if there is a…

No. Let us look at the one. So, what you are saying task T T 2

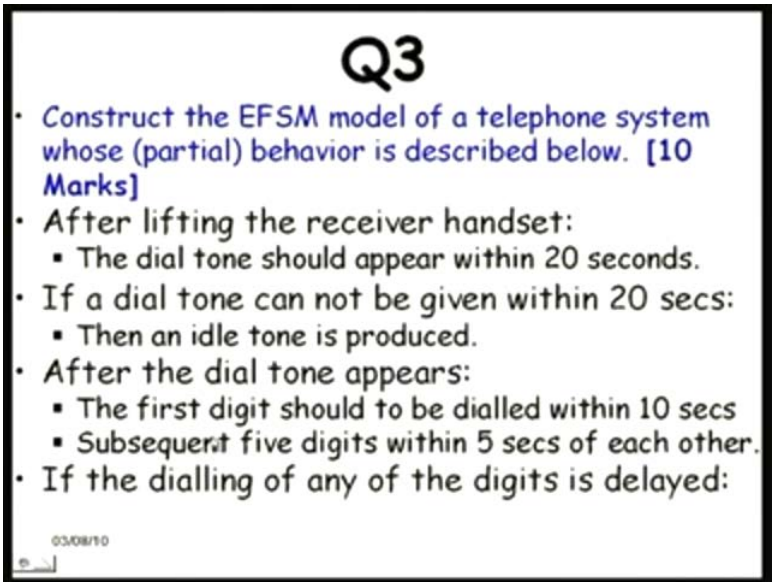
Phase and the period are the same.

See, we are considering the worst situation when the phases are all zero. So, it does not matter what is the phase value whether 60 or 130, it does not matter.

When actually scheduling, we want to draw a diagram. Then the phase is 60, and after that, it should start, but by that time, the period is over.

No, no, no, see the period is from the point it starts. So, if it arrives at 60, its deadline will be at 100; these are relative values.

(Refer Slide Time: 27:51)

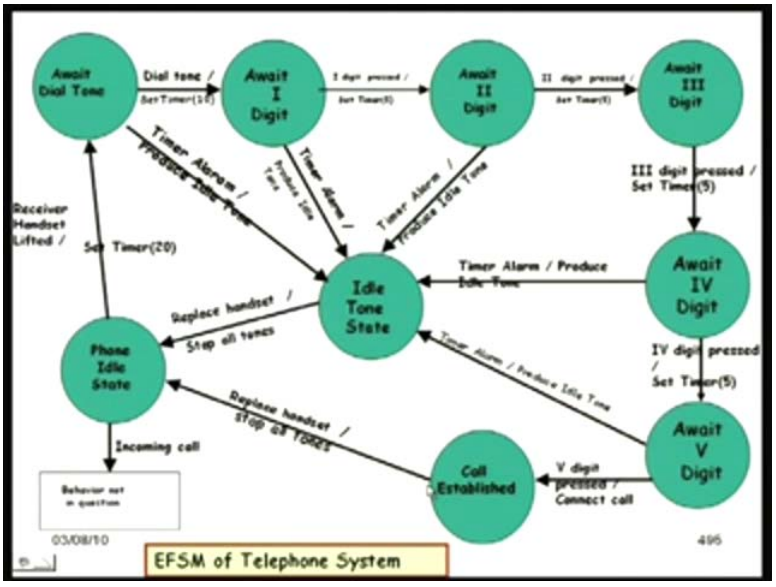


So, let us look at the next question which had 10 marks to construct extended finite state model for a telephone system. It is partial behavior is described below. So, after lifting the receiver handset: the dial tone should appear within 20 second. So, this is the event

lifting the handset and then within 20 second. So, we have to set a timer for 20 and if the dial tone appears before 20, then you wait for the digit. If it dial tone does not appear by 20, idle tone is produced.

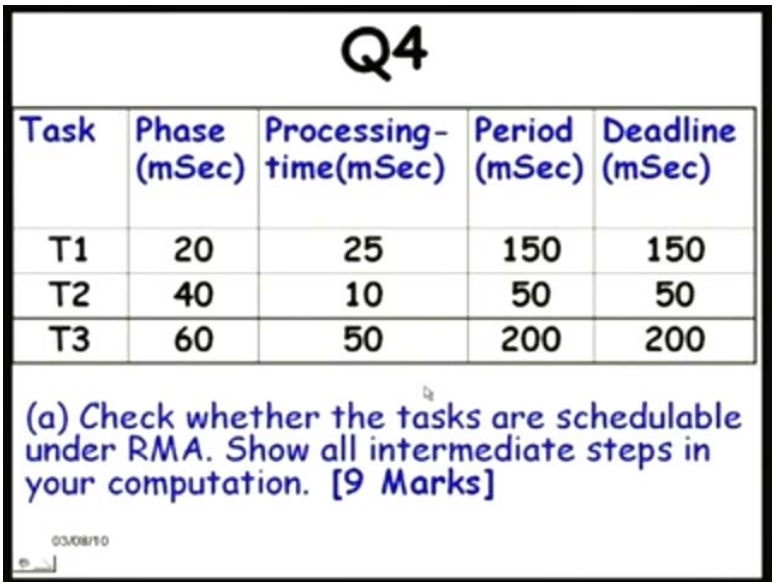
So, there is a timer to be set at twenty here, and after the dial tone appears, the first digit should be dialed within 10 seconds. Otherwise, an idle tone will appear subsequent five digits within 5 seconds of each other. I think I left that statement here which was there; otherwise, dial tone will appear, sorry, otherwise, idle tone will appear, and if the dialing of any other digits is delayed, an idle tone is produced; this is the statement. So, nothing, very hard to do. Find out what are the events and what action needs to be taken. So, somebody who has done correctly I just got that here. So, just look at it. Idle tone continues until receiver handset is replaced.

(Refer Slide Time: 29:36)



So, this is one somebody has done among you. So, basically same thing set time or and then whether dial tone appears. Otherwise, if there is a time alarm, go to the idle state is similar for each of the digits. I hope no one has difficulty here. Find out what are the events and what are the constraints on them set timers.

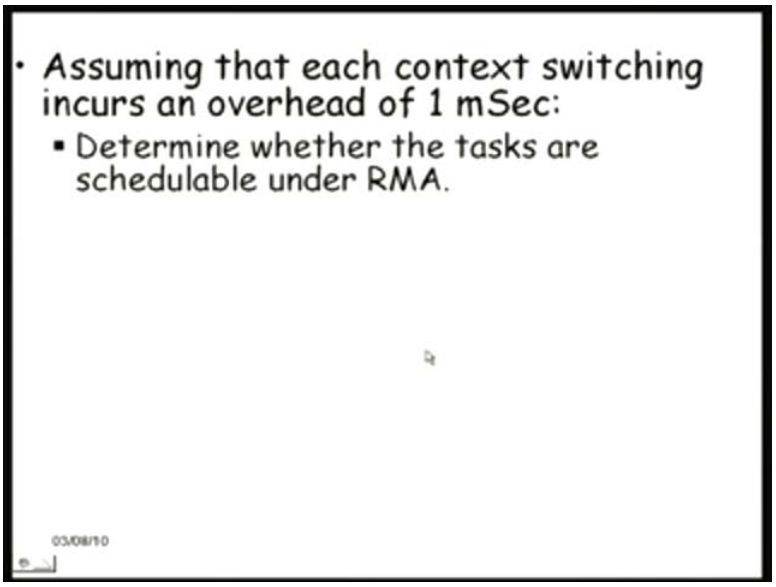
(Refer Slide Time: 30:07)



Now, let us look at the next question which had several parts here for a task set that is given. Three tasks are given with some phasing, and then, the processing time for each of the tasks is given; the periods are given and the deadlines are given. Deadlines happen to be same as period.

Now, check whether all the tasks are schedulable under RMA, and so, the intermediate steps. And these are easily schedulable even the Liu Layland criterion I think. So, these are schedulable. So, there is nothing very hard here. Apply the Liu Layland and check find the utilization for each of the tasks, and then, check the Liu Layland criterion, it will satisfy. So, task set runs under RMA, no doubt that.

(Refer Slide Time: 31:10)



Now, the next part was assuming that context switching incurs overhead 1 millisecond: Determine whether the tasks are schedulable under RMA.

When the schedule

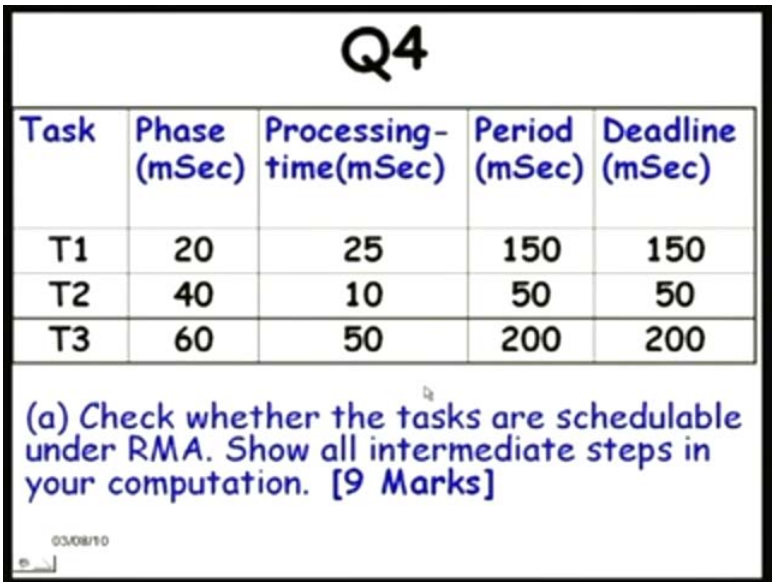
Yes yes.

(( )) do not be need to consider that arrival time arrival time

You mean the phasing.

Let us look at the question that he is asking.

(Refer Slide Time: 31:32)

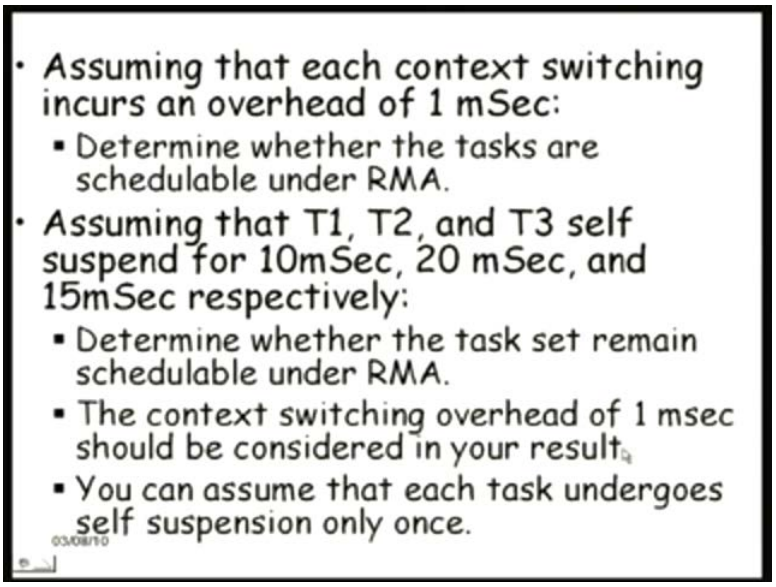


See, we have the phases given. The execution time and the period deadline is given. So, in Liu Layland criterion which is a pessimistic criterion actually, if it satisfies, then it will definitely run. If it fails, then we will have to check further. And Liu Layland it is pessimistic and even does not consider the phase. If it is satisfies Liu Layland, it will run under all phasing you can think of.

Suppose if a I write that suppose. So, if I raise that one to the (( ))

No, no, see the see the processing time, period and deadline all these are relative values; these are relative. So, from the point it arrives, from there onwards, the period starts, the deadline starts and the processing time.

(Refer Slide Time: 32:35)



Now, let us look at the next part. Assuming that each context switching incurs overhead of one millisecond: determine whether the task set would remain schedulable under RMA. These also nothing very hard here. We have got a result saying that the effect of context switch is to increase the execution time of every task by in the worst case by two times the context switch, 2 millisecond.

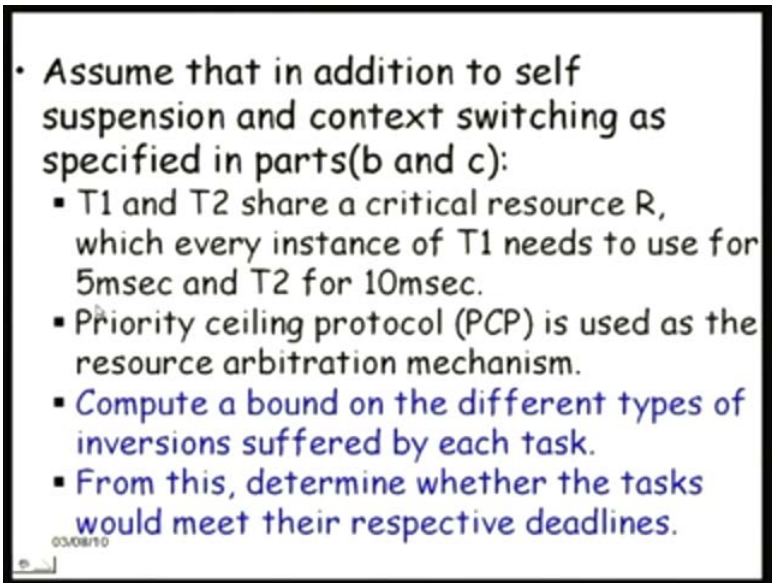
So, in effect the execution time of each task, if we increase by 2 millisecond, then we can say that whether it still remains schedulable, but of course, these are all worst case computations, because something which fails here might still be a schedulable, but that we are not worried here, because after all, all this real-time tasks scheduling, we are concerned about the worst case situation.

So, increase the execution times by 2 millisecond for each of them. Check again Liu Layland whether they remain schedulable easily remain schedulable as per the data given in the problem. Now, the third part of the question was that assuming that the tasks also self suspend themselves, that is for performing some waiting for some events or some i o, etcetera. So, they self suspend themselves for 10 millisecond, 20 millisecond and 15 millisecond respectively. Then check whether they would still remain schedulable under RMA.

So, there we had said that a task, the highest priority task we need to substitute the value for self suspension in the Liu Lehoczky criterion and check whether it will still remain schedulable, but for lower priority tasks, we also need to consider the self suspension of higher priority task, the minimum of the self suspend time and the execution time.

So, find out that for every higher priority task, the lower of the self suspension time and the execution time add them up and check whether still the Liu Lehoczky is satisfiable, and as it turns out that, it does not remain satisfiable. So, check whether you have got that, and also in this computation, we should have taken the context switching overhead of 1 millisecond which is basically increasing the execution time by 2 milliseconds.

(Refer Slide Time: 35:27)



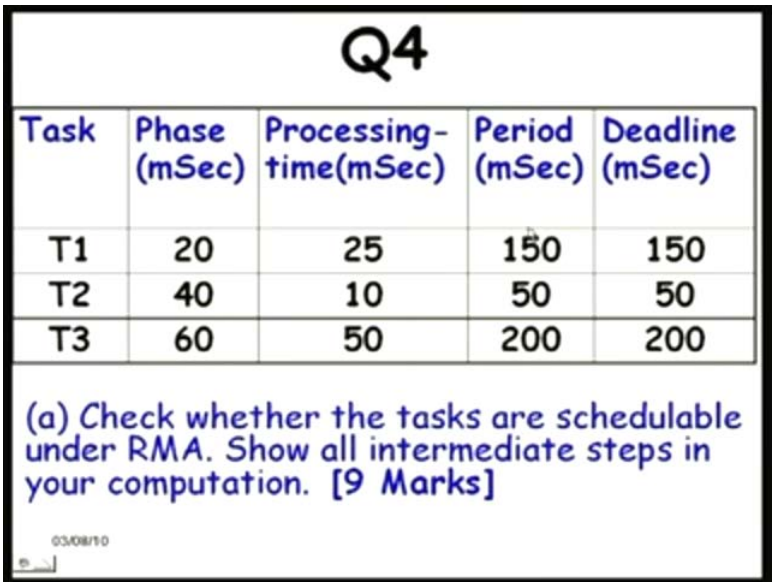
The other next part of the question was that in addition to self suspension and context switching in the previous parts, two tasks, that is, T1 and T2 also share a critical resource r. Every instance of T1 needs to use the resource for 5 milliseconds and T2 needs to use for 10 millisecond. The priority ceiling protocol is used as the resource arbitration mechanism, and then, compute the bound on the different types of inversions suffered by each task and then determine whether the tasks would meet their respective deadline based on the compute the third part here, the third bit here.

So, this is also straight forward question because you have only two tasks that share a resource, only single resource and two tasks need a resource. So, the higher priority task

T1 will be blocked by T2 directly. So, three will be a direct inversion and the inversion will be for what time? Will it be for 10 or 5?

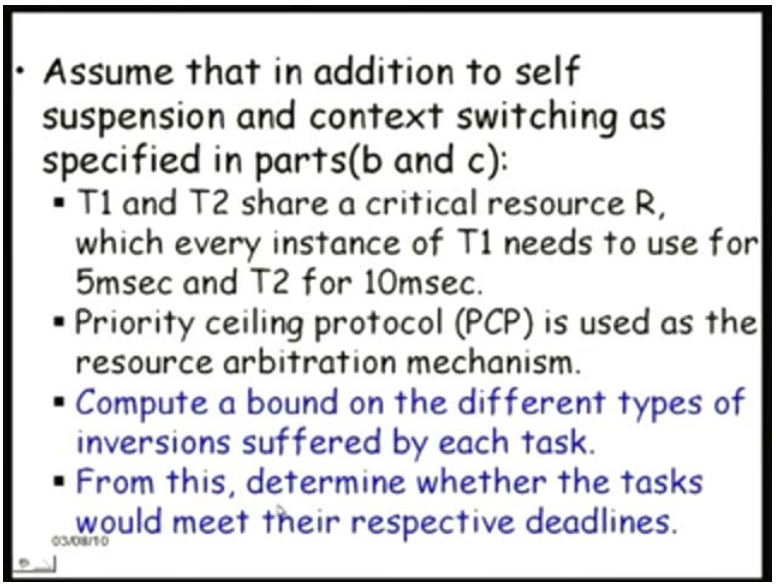
T1 will be blocked, I am, I am sorry, I think the question as it was given T2 is the higher priority. Let us, let us just check that where is the data there, yeah, yeah, this is the data here.

(Refer Slide Time: 36:59)



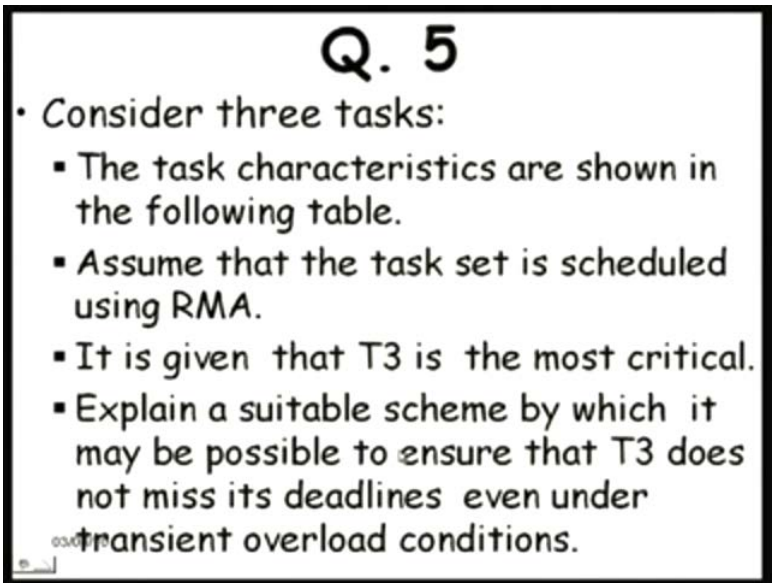
So, yeah. So, here, T2 is higher priority than T1. So, may be some of you because these were written in different order, may be some of you have missed this point. They have used T1 as higher priority than T2, no, T2 is the higher priority.

(Refer Slide Time: 37:21)



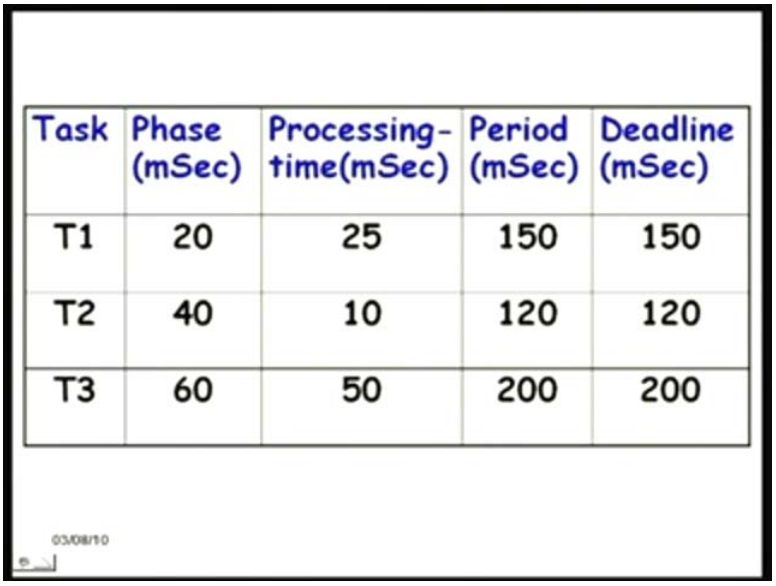
So, T2 will be blocked by T1 for what duration? What is the maximum duration that T1 will block T2? 5, but somehow many of you have written 10 millisecond. So, that is a wrong answer. T2 can be directly blocked by T1 for 5 millisecond, and then, since there are two tasks, there is no waiting task here no waiting situation priority inheritance clause will not arise. So, the other types of inversions will not occur. So, it just one direct inversion of 5 millisecond, and then, this 5 we need to add for the Liu Lehoczky criterion for that task simple calculation here, nothing very complicated, simple question.

(Refer Slide Time: 38:21)



There is another question here the last question the mid sem. Consider three tasks and the task characteristics are shown I think it is there in the next slide. Assume that the tasks are scheduled using RMA, and given the T3 is the most critical, then explain how a scheme by which T3 does not miss its deadline even under transient overload condition.

(Refer Slide Time: 38:52)



So, here, we had said that, let first look at the task characteristics. So, T3 is known to be the most critical task, but it has higher period. So, using RMA, this will have the lowest priority, and then, the higher priority tasks can cause this to miss its deadline. So, since we have to use RMA, that is based on the rate at which they occur, so, you need to split this task into two parts such that it becomes, the period becomes lower than all of them. We do not really physically split the task, but it is for our own computation using the RMA formula, etcetera, we have to split this task, and then, we show that it remains it meets its deadline once we do this logical splitting of the task. So, that was the last question.

So, please check your evaluations, and at the end of today's class, if anybody has the answers are wrongly mark because there are possibility, we quickly checked through the answers, and in any case, in any examination, there is always a finite possibility that marks are given wrongly. So, if anybody has difficulty with the correction, we will address that after the class.

Now, let us proceed with the discussion that we are having last time.

Sir one.

Yeah yeah.

Sir in the forth question.

Yes

The people undergoes direct inversion for (( )) 5 seconds.

Yes

So, (( )) doing it to add the self suspension time of T2 and there is context.

What does the question? See, you will have to just do what the question says, I am sorry.

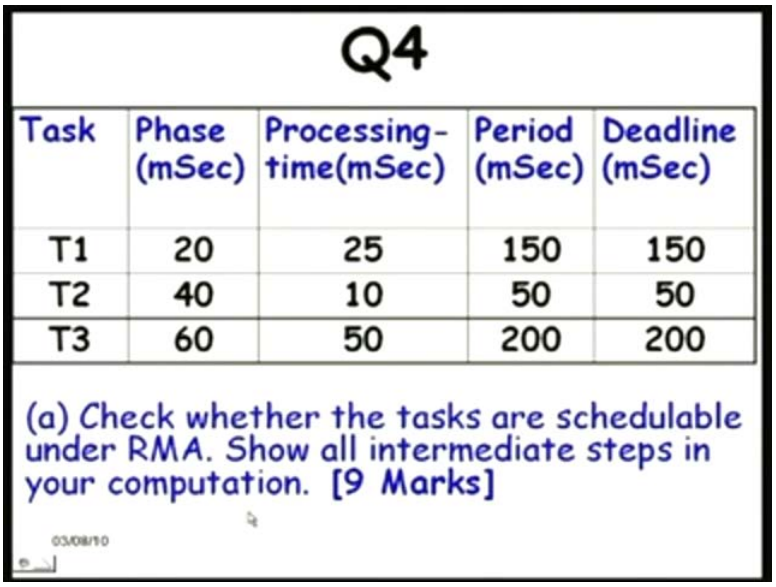
It is not the direct inversion inversion time

So, let me just get that slide here and then we will. Let me just to get. So, just reset it to the beginning, sorry, let me just get the, get the question and then we will answer your question. I think this is the one.

This one, question 4.

Sir T2 undergoes direct inversion for 5 seconds

(Refer Slide Time: 41:28)



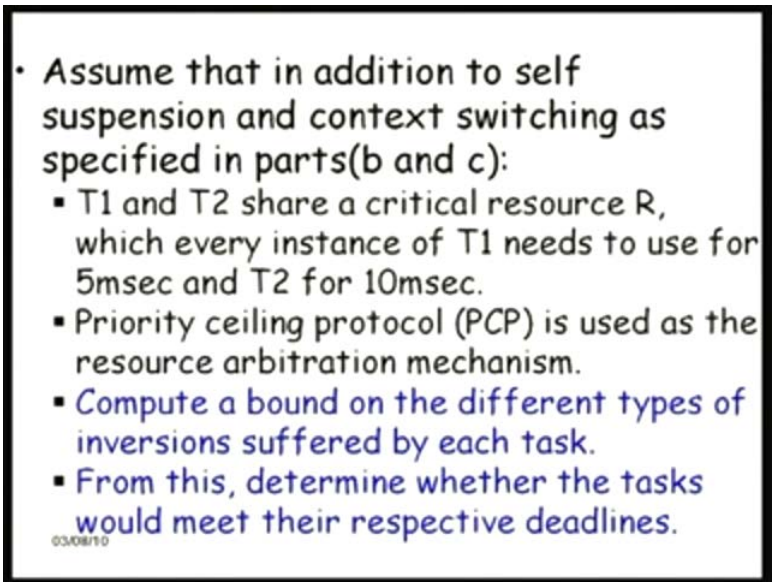
T2 would under goes direct inversion for 5 second and then you are saying whether we should consider the context switching and the…

The self suspension

Self suspension time

So, let us see the question what does it say. We should do according to the question. Last part you are saying, is not it? Yeah.

(Refer Slide Time: 41:44)



See, in addition to self suspension in context switching's time mentioned here. So, we need to consider these two.

So, in that case for how much duration T2 under goes a direct inversion?

You are saying that if it is holding the resource and then it under goes a self suspension, is that is that your question? See, the question is given that the maximum period it holds it for 5 milliseconds. So, we can make that assumption and I think it is not very clear here that it uses whether this is the maximum period. Just says that it uses for 5 milliseconds. So, there is a small ambiguity here, but for simplicity, we can assume that this the maximum period, and if you have done like you have consider that self suspension can occur. Even when it is holding the resource, then we need to consider that value. If you done like that and written that, it can self suspend while holding the resource that will be considered, not a problem. There is a bit of ambiguity here yes.

Is is adding context switching time (( ))

Yeah, context switching will any way occur you know. So, not a problem. So, you are saying that whether we need to consider more number of context switches, is it? Is that your question?

No sir. In direct (( )) if we are ask, what is the time it undergoes direct inversion?

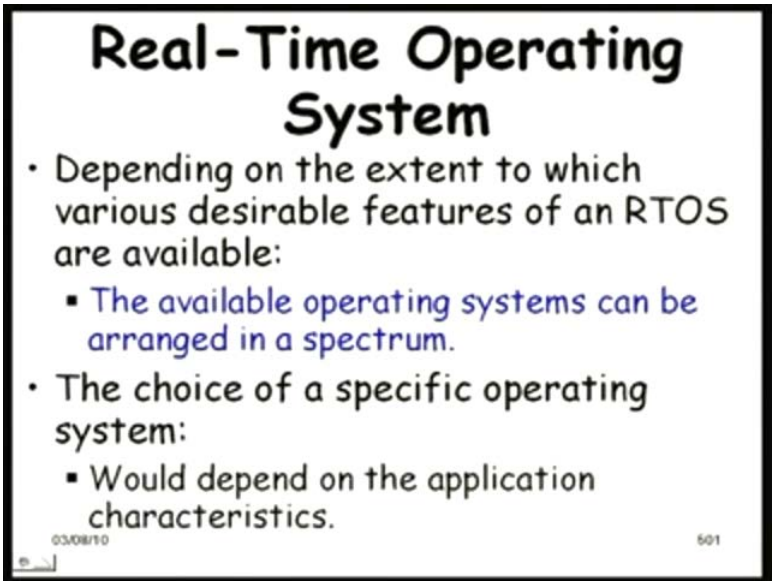
Yes

Should we add the context switching time also no know sir?

No. See, the, the, time it is holding the resource, that is the time it is blocked know.

So, at the end of the class, we will look at anybody's answers are wrongly marked.

(Refer Slide Time: 43:43)

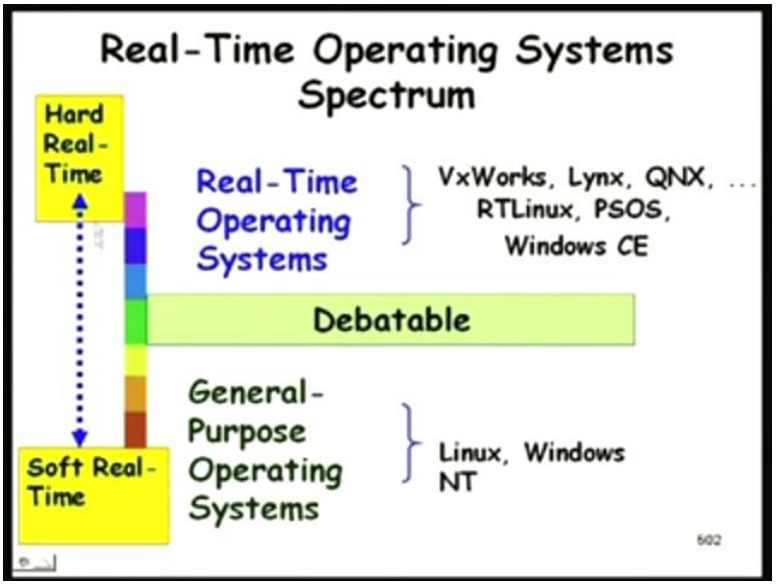


Now, let us proceed with what we are doing last time. So, we are discussing about some very basic issues in real-time operating systems, in that, what are the desirable features and what sets apart a real-time operating system from a traditional operating system. We had seen several characteristics including support for a static priority level, and then, we looked at the scheduling policy that can be supported for scheduling the tasks and then the resource sharing arrangement, etcetera, we had identified many points.

Now, if we look at the available features, those desirable features in the operating systems that are available both open source and the commercial ones. We can find that they meet those criterion or they have they support those criterion, various extents, and based on that, we can arrange those operating systems into a spectrum depending on to what extent they support those features.

And of course, once you have done that, the choice of a specific operating system will depend on the application. So, the application is not very stringent. We can choose something lower in the spectrum. If the requirements are very stringent for an application, need to choose something on at the other end of the spectrum.

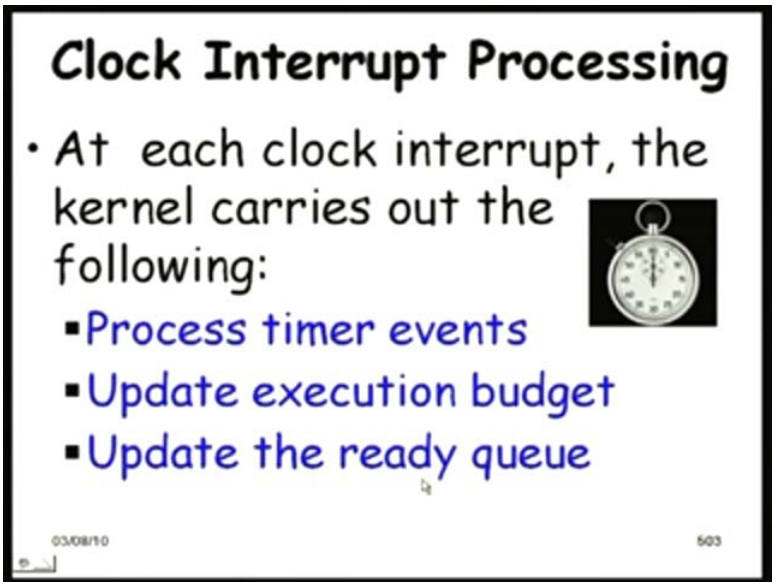
(Refer Slide Time: 45:20)



So, if we look at the spectrum of the operating systems here, we will see that there are some operating systems like v x Works, Lynx, QNX, PSOS, RTLinux, windows C C, E cos, new cos, etcetera, we are going to discuss about that. So, those have all those features, we discussed and they are the hard real-time operating systems. They can, we can develop hard real-time applications using that, and even among them, we will see that what is the size of the operating system? What is the interrupt latency? How can the device drivers be installed on the operating system and so on. Based on that, we can again classify this into various types.

So, if we need something for a embedded application, then we need to possibly select PSOS or maybe windows CE. We will see that later, and then, we have the general purpose operating systems like Linux, Windows NT, Windows XP etcetera. So, these are the general purpose operating system where we can run only soft real-time tasks. If we try to run hard real-time tasks, yeah, then definitely they will miss their deadlines.

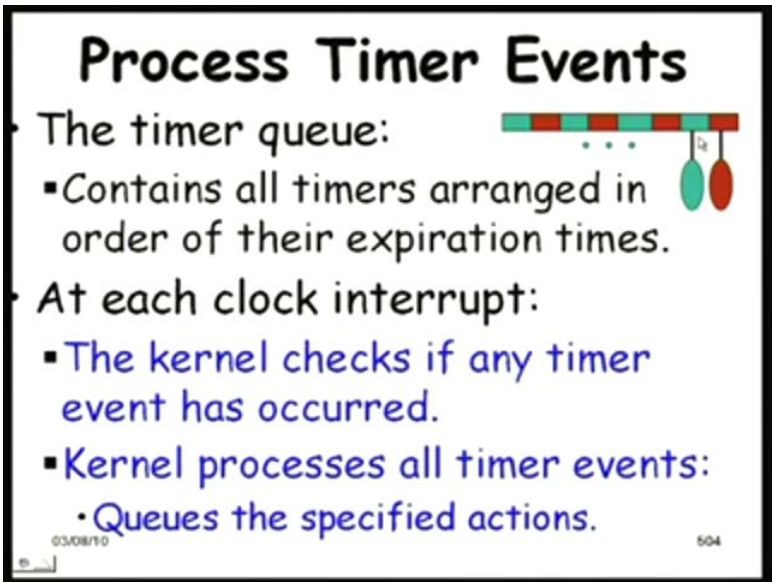
(Refer Slide Time: 46:56)



Now before proceeding further into the looking at the commercial operating systems, their features, how to develop applications and so on, let us look at one basic issues in all this operating system that is about clock interrupt processing. We had said that there is a quartz clock which based on which interrupts arrive, and each time the clock interrupt arrives, then the following activities take place. Three main activities actually - one is to process the timer events, because each of these operating system will support a set of periodic timers and the one set timers, not one, but many of them.

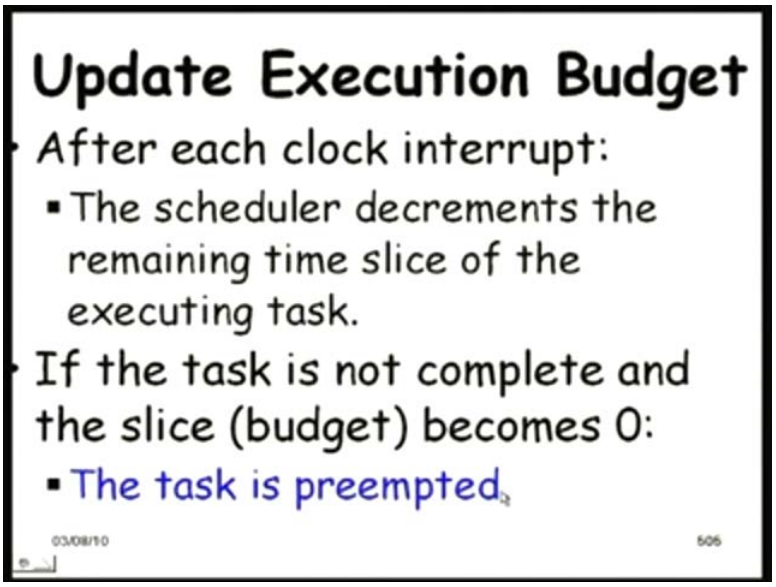
And then, it will check whether any timer event has occurred, all those timers that are there you. Some timer has fired actually, I mean timer has timed out. So, then the action for that timer must be queued. So, check the timer queue. Find out if any timer has expired and then the action that needs to be taken once the timer expires, need to be, that action need to be queued, and then, the execution budget of the currently running task must be updated. How much time it is running? Because we will see that among equal priority tasks. Round robin scheduling is typically used, and then, we use the, then the, the ready queue has to be updated, because since the last clock interrupt, there might be some tasks which have become ready due to some events.

(Refer Slide Time: 48:54)



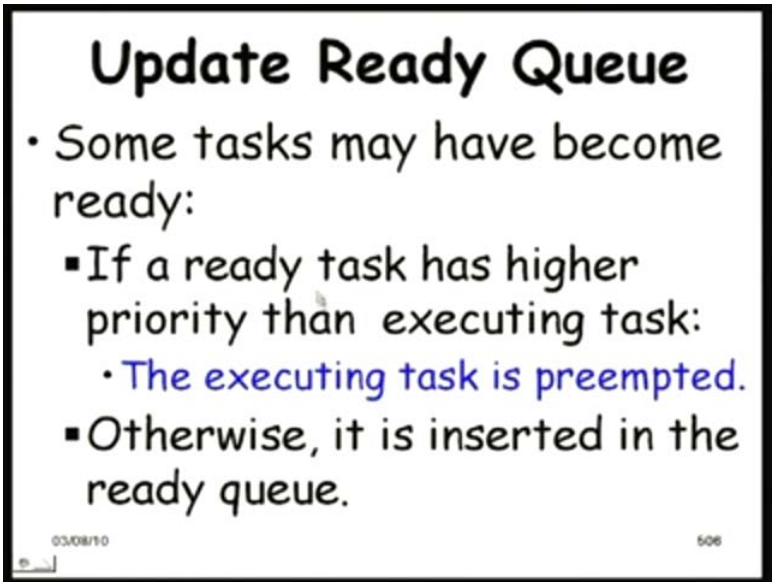
So, let us look at the first activity once a timer, once the clock interrupt occurs that is about processing the timer events. This is a queue of timers. The timers are arranged in x in, in, ascending order of their time out periods and these are the handler routines actually. So, once a timer's time is up, the handler routine needs to be invoked, that needs to be queued in the task queue. So, the timer queue will contain all the timers arranged in order of their expiration times and its clock interrupt the kernel would check. If any timer event has occurred or the timer has expired and then it will queue the specified actions.

(Refer Slide Time: 49:52)



The second task that occurs once a clock interrupt occurs to update the execution budget. Decrements the remaining time slice and if the task is not complete and the slice has become zero, then it needs to be preempted.

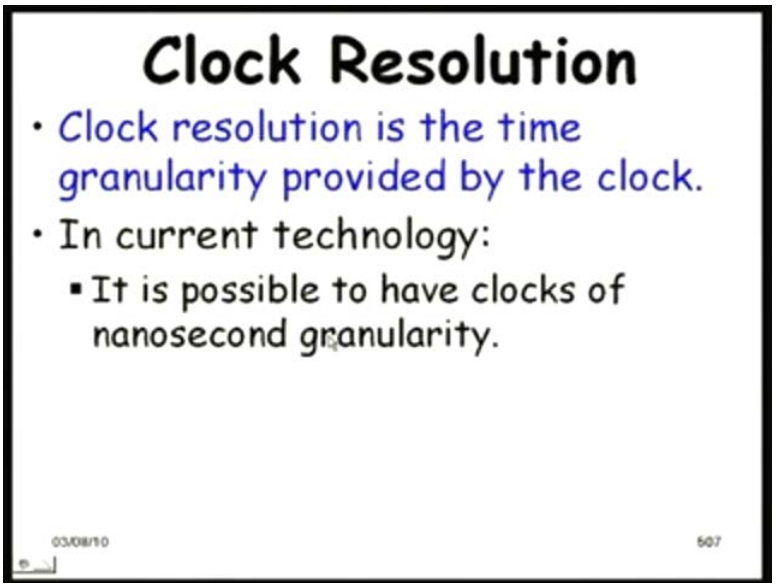
(Refer Slide Time: 50:09)



The third activity is about updating the ready queue. Some tasks might have become ready since the last clock interrupt arrived, and then, it has to check if the ready task has higher priority than the executing task. The executing task is preempted; otherwise, it is inserted in the ready queue.

Any doubts here? Any comments? Only possibly has just thinking that somebody will ask that see. So far we consider it is event driven scheduling, that is, once an event occurs, immediately responded that was the scheduling algorithm I had discussed event driven schedulers, but here, we are saying that only after on a clock interrupt, we will check whether some event has occurred.

(Refer Slide Time: 51:06)



Now, let us look at the clock resolution issue. The clock resolution is basically the granularity of the clock.

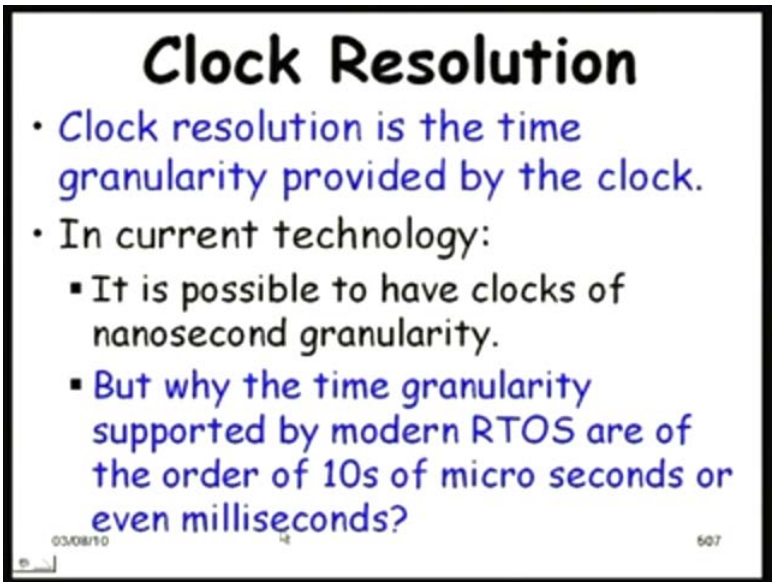
Sir, what happens if one event occurs before that clock's duration?

That is what if the event occurred and the next clock interrupt that will be handled.

(( )) that period will be (( ))

The clock interrupt is typically comes very frequently, we, we will just check that. Let us discuss about that.

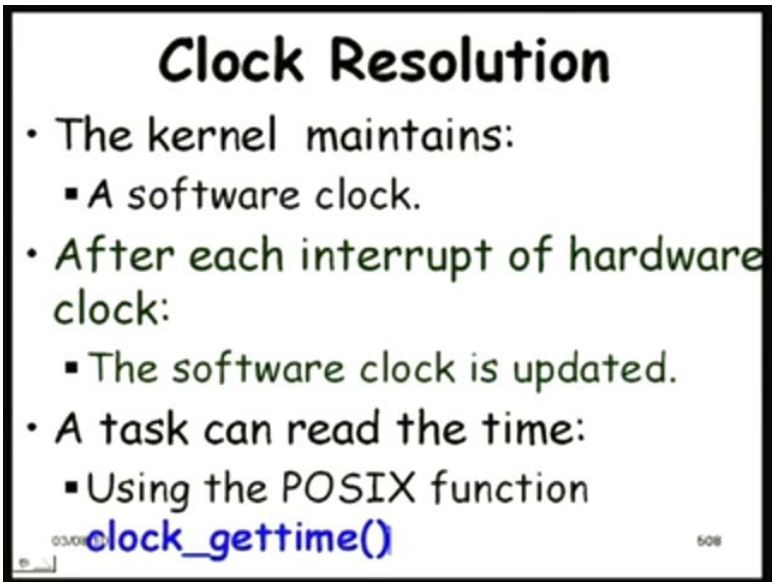
(Refer Slide Time: 51:41)



So, we are discussing about the clock resolution. Typically all these software, all these operating systems they have a software clock. They do not directly use the hardware clock actually. Based on the hardware clock, a software clock is maintained. The software is typically of lower granularity has to be lower granularity than the hardware clock. Now, for based on this clock, the operating system reacts.

Now, in the current technology, we have Giga hertz hardware clocks; we talk of 3.6 Giga hertz etcetera, 2 Giga hertz, (( )) etcetera. But the time granularity if you look at the real-time operating systems, that we are going to discuss the commercial operating systems. These are 10 of micro seconds even though good real-time operating systems. Why cannot they have a nano second granularity and this even some of them have milliseconds. What is the answer to that?

(Refer Slide Time: 53:02)



The answer is that actually the clock we are talking about is a software clock, it is not the hardware clock, and after each interrupt of the hardware clock, the software clock is updated. So, several interrupts may be used to update the software clock only once, but then again, you will ask see, why not make the software clock keep the same period at the same frequency as the hardware clock?

The answer let us look at it. Let us understand the answer that why the software clock has a lower granularity than hardware clock even the hardware clock is nano second; the software clock is milliseconds or micro seconds. Even it is very difficult to make it nano second and also it is may not be useful to make it nano second.

Let us check that answer. So, why do you think that nano second software clock is difficult? What is your thinking?

Software clock is like constantly updating in the software.

Yes.

(( ))

No you can keep on incrementing. Each time the hardware clock interrupts. Keep on incrementing the software clock.

The increment instruction itself takes more than that…

Not really.

(( ))

Increment does not really take, because know, this is a increment does not…

(( ))

Yes.

(( )) .

Not, not, really you cannot you cannot use a, you know, these are pipelined processors. So, it is not that every few clock cycle, one instruction will execute. Every clock cycle also one instruction can complete, not a problem or you can have a divider circuit. So, every timer few timer interrupt once it is interrupted, not a problem. You do not have to really interrupt at Nano Giga Hertz. You use a divider circuit and then it interrupts every few nano seconds.

So, let us just take a short break and we will come back and we will discuss why the software clock that is maintained has a lower granularity than the hardware clock, and then, few other issues before we start, really discussing about different types of operating systems, and how they can be used? What are the features for which application? What operating system to be used, etcetera? Let us stop here.