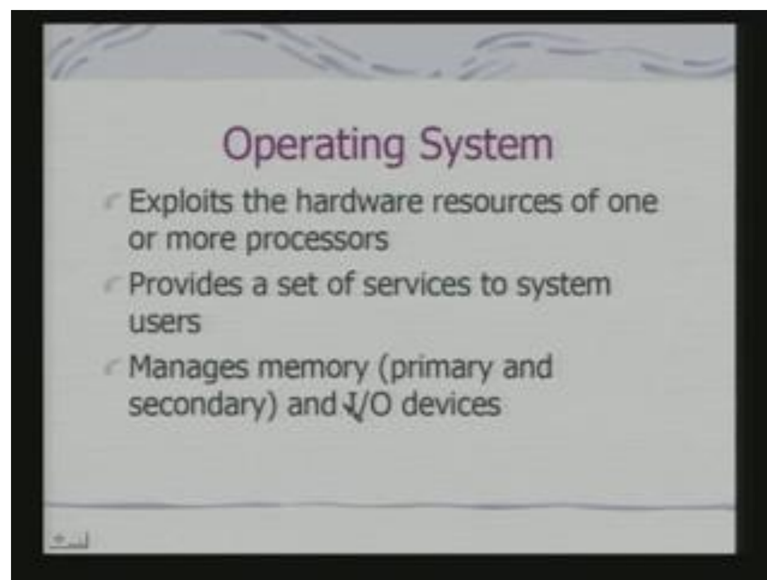
Embedded Systems
Dr. Santanu Chaudhury
Department of Electrical Engineering
Indian Institution of Technology, IIT Delhi

Lecture - 20
Fundamentals of Embedded Operating Systems

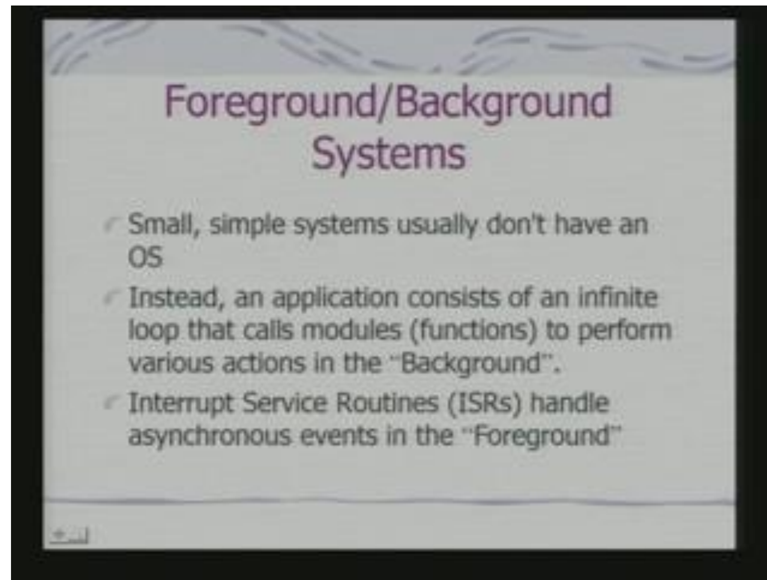
In today's class, we shall discuss Operating Systems for Embedded Applications and Embedded Appliances. Obviously, there would be spatial requirements, when we are considering embedded systems. We shall start with reviewing the general characteristics of operating systems before going into the specific needs of embedded systems.

(Refer Slide Time 01:46)



What is an Operating System? Operating system is that layer of software which exploits the hardware resources of one or more processors which is their in the Embedded System or for that matter any general purpose computing platform. Therefore, it provides a set of services to system uses because, system is making use of hardware resources through that software layer which is your operating system. OS manages memory primary as well as secondary memory and IO devices.

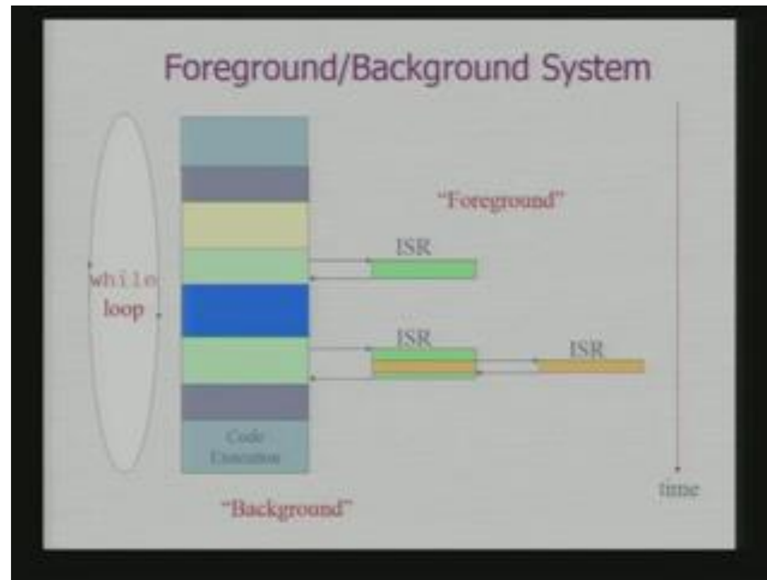
(Refer Slide Time 02:24)



Small simple embedded systems usually do not have an OS. Instead an application consists of an infinite loop that calls modules or functions to perform various actions in the background, because it is a common flow with different functionalities to be checked to. And interrupt service routines handle synchronous events in the program this is the basic structure for the more common systems.

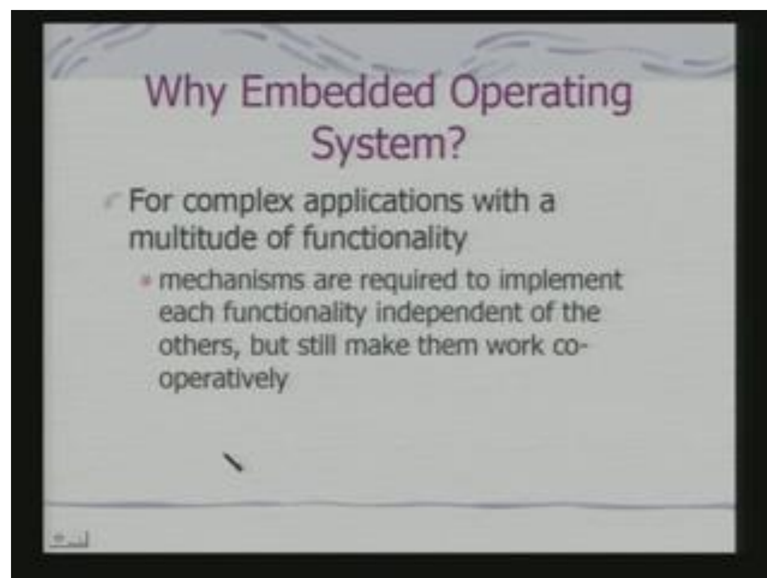
But of the other hand, if you really have complex functionalities to support then in that case we really need to have an OS. The basic model of this kind of simple embedded systems is software management software is basically referred to as foreground background systems.

(Refer Slide Time 03:25)



So, the structure wise this is a general while loop. Because, this software is expected to run for ever, which has got the different modules and this modules gets invoked within the while loop. Asynchronous events which are generated from the external environment or handle by the interrupt service routines. So, this is the basic structure of management software. When, we really do not need to manage multiple concurrent tasks.

(Refer Slide Time 04:06)

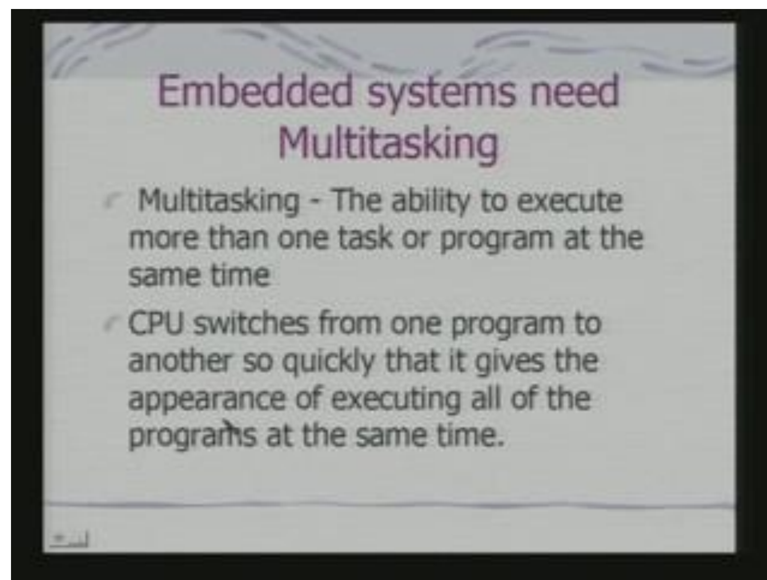


For complex applications which supports a multitude functionality. We need mechanisms to implement each functionality independent of others. That means, this

functionalities will not be link to each other through say proceed your calls. So, they have to be implemented independently.

And but still, we need to make them work cooperatively if this is the condition, then we really need and OS to manage that embedded system. And we had already seen that external world has got concurrent event stream or data stream.

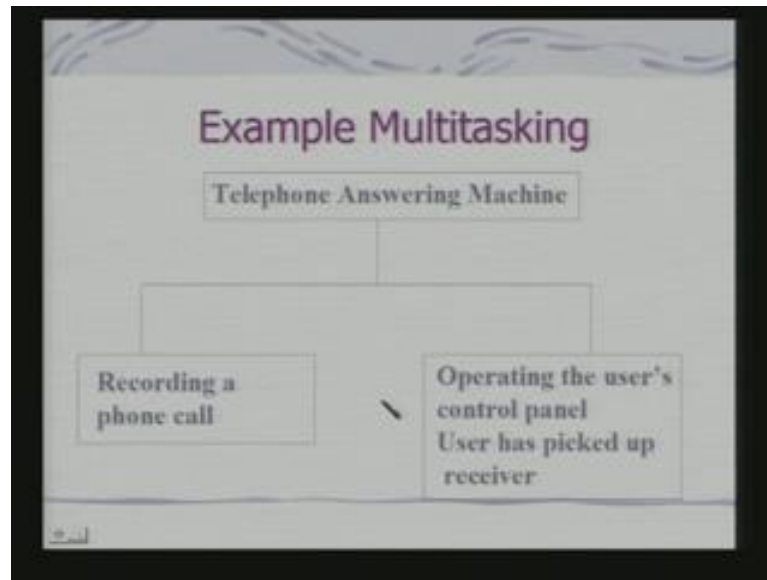
(Refer Slide Time 04:50)



So, if you need to deal with concurrent event or data stream will require multitasking. Because, with each stream we need to have a special functionality associated. And in Multitasking system CPU switches from one program to another quickly. So, that it gives the appearance of executing all of the programs are the same time.

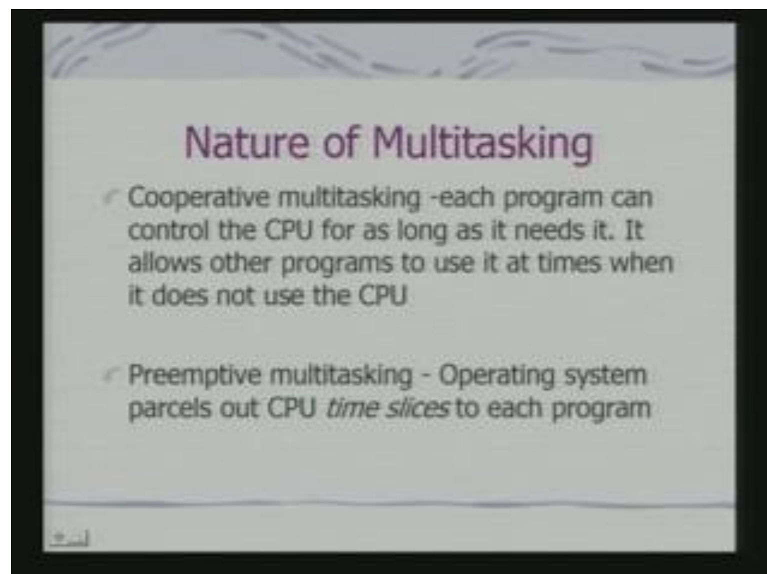
In fact, we have a same processor and on the same processor these tasks are being time shared. And these tasks are actually independent modules. And these therefore, functionalities are getting implemented or realized through independent modules.

(Refer Slide Time 05:29)



So simple example of multitasking is here, where you have got a telephone answering Machine, it can record a phone call also the same time. It may meet the user control panel inputs other requests. And for example, user has picked up the receiver. So, that is the external scenario which it needs to create to as well as it record a phone call as well. And these two can be consider independent concurrent task and that is to be managed by the OS resident in this telephone answering machine.

(Refer Slide Time 06:13)

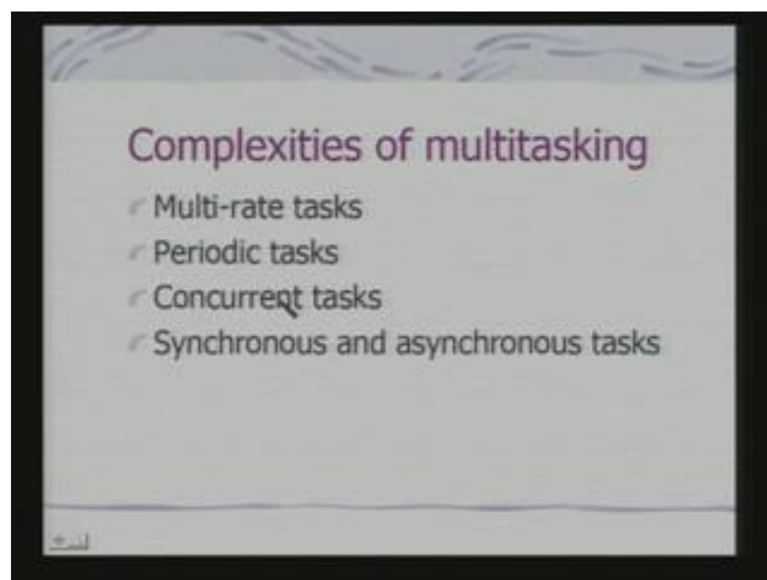


So, what are the different kinds of multitasking systems that you encounter? One is cooperative multitasking each program can control the CPU as long as it needs it. It allows other programs to use it at times when it is does not use the CPU. That means, the other programs get the CPU and can do the task only when one of the programs is not using.

So, it is a complete cooperation with the complete knowledge about each other, which may not be always true. Side by side you have got preemptive multitasking. In preemptive multitasking, a process, which is in execution or each program, which is in execution can be interrupted that is preempted.

And another process can be schedule to run in its space. And in this condition OS provides each running program with specific time slices. So, each program runs for its time slice and then it is preempted out. It is not that it will a process will completes it is job or task with in a time slice allotted. It will require possibly multiply time slices to finish its task. Now, when these processors are running concurrently they may have different characteristics as well.

(Refer Slide Time 07:42)



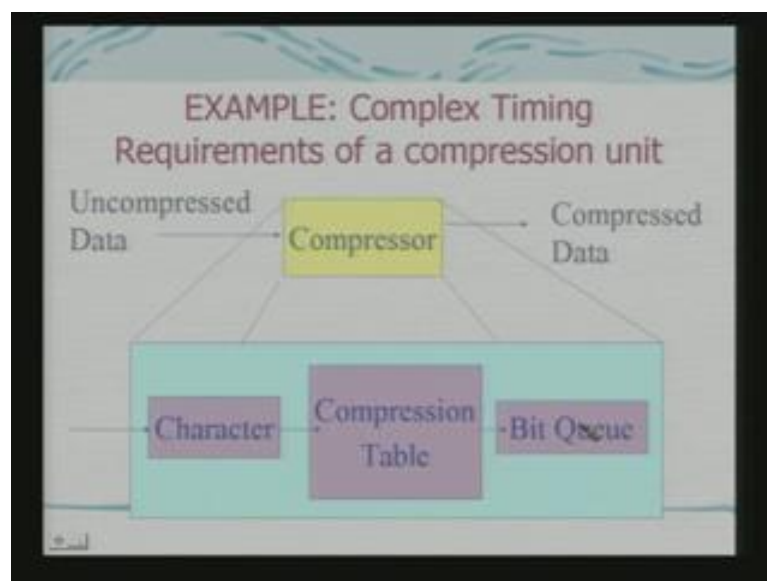
When, we talk about periodic tasks it means the tasks have got as specific period with which their rife and there to be serviced If you consider a simple decompression task for decompressing a video. Then, that task arrives at each frame time interval. If you have

30 frames per second or 25 frames per second then these tasks would arrive at 40 millisecond interval. So, these are an example of a periodic task..

Now, when a process is periodic it may have other processors in the same OS which are also periodic. But, there period can be different from that of a particular periodic process. So, if an OS needs to handle multiple processors which come with multiple periods. We actually encounter, what is call multi rate tasks.

And, all these task are in a way concurrent, because they have to be in state of execution at the same point of time. It is not that one task will be finished then only I can created to another task. There also synchronous and asynchronous tasks, a task which is spooned by an external interrupts. Or an external user action would be typically an asynchronous task and that needs to be created to in this scenario as well. So, all these things together make the problem of multitasking more difficult.

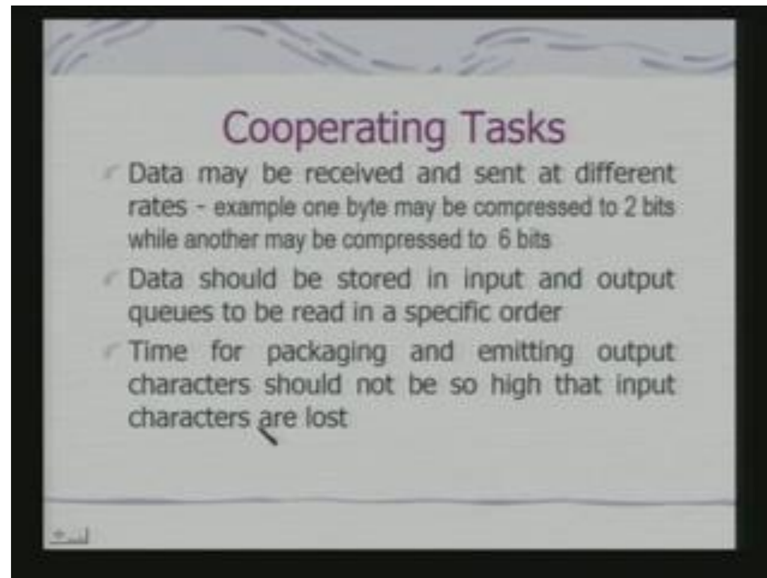
(Refer Slide Time 09:24)



Let us take a simple example of a compressor, a compressor which is compressing data. So, you have got uncompressed data and there is a compressed data’s output. And this has to work in time. So that, you should not need any of the input data which is coming in an uncompressed form if, you look into the overall thing the depending on the character.

And if you using a compression table and if you use a variable length coding scheme like half bank coding then a character would give rise to data of a particular length. For that length is not same for each and every character. So, that problem needs to be handled.

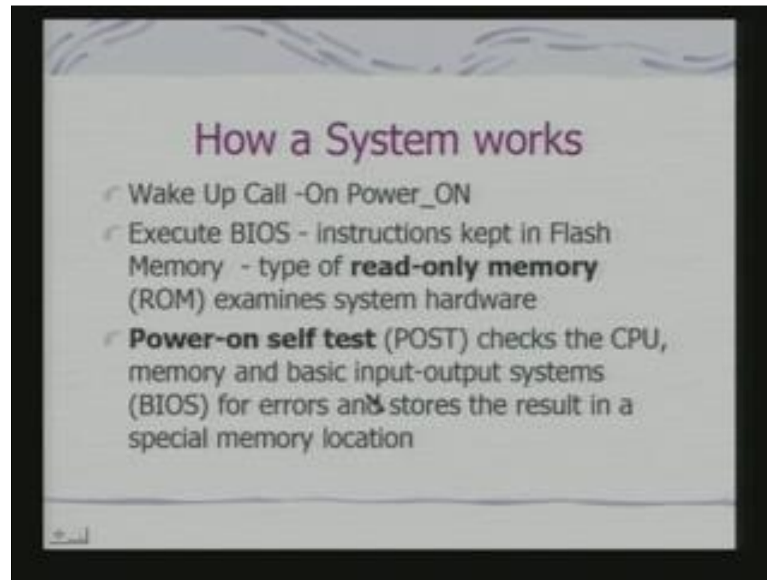
(Refer Slide Time 10:11)



So what we say, that the input the data may be received and sent at different rates. So, there are two processors which are managing the reception and communication. So, these two processors are working at different rates and then it to cooperate such that no input data is really missed. So, time for packaging and emitting output characters should not be so high that input characters are lost.

So, this puts in a constraint on the compression scheme. And these processors have to cooperate, so that the compression can take place in proper order. Now, let us look at with this background. Let us look at the fundamental issues of the OS. We shall base or discussions primarily around considerations of an embedded system.

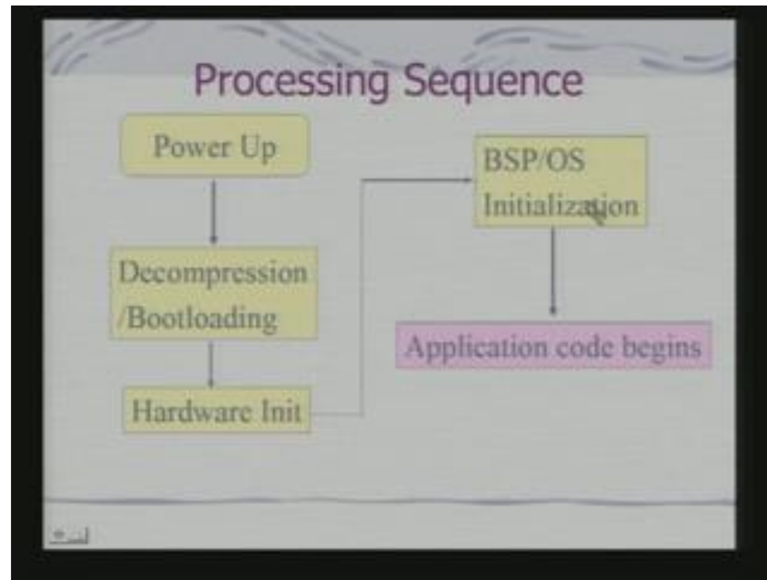
(Refer Slide Time 11:13)



So first, what happens, when you power up an embedded system. So typically, there is a power on situation. And if there is an OS in a power on situation you execute, what is called a basic input output routines, which may be part of ROM. ROM may be flash memory as well we had discussed is other otherwise also. And these memory initializes examines the system hardware.

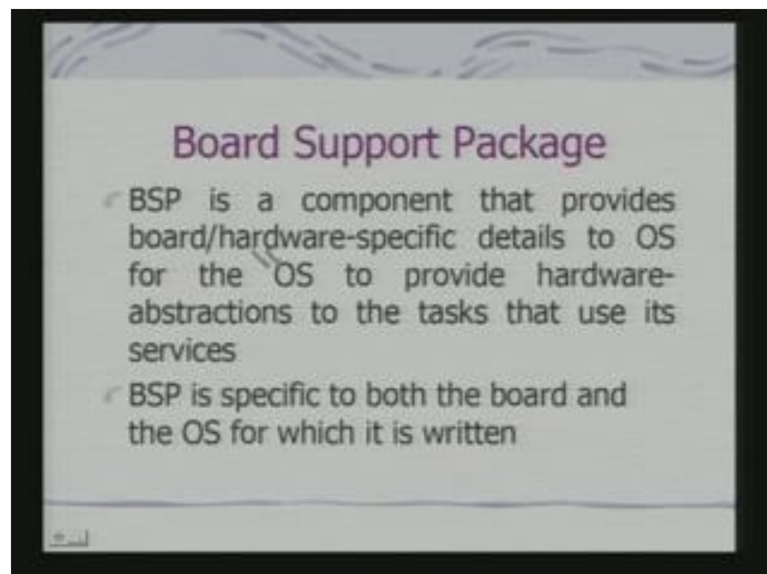
In most of these embedded systems, there is a power on self desk block. We check the initial when it boosts up it checks the hardware. Typically, memory test is performed, in order to ensure that the embedded system does not give rise to unsaved state or kind of an erroneous situation, because of the hardware fault. So, if you detect the hardware fault it would about.

(Refer Slide Time 12:11)



So basically, what we are looking at, in terms of the processing sequence. The moment the power is up. So typically, you do what is called Decompression or Bootloading. This is basically your basic IO routines and the OS which is part of your ROM or your Flash is loaded onto the RAM for faster execution. Then you going to, what is called this BSP or Board Support Package or OS initialization and then only the control gets transfer to application code.

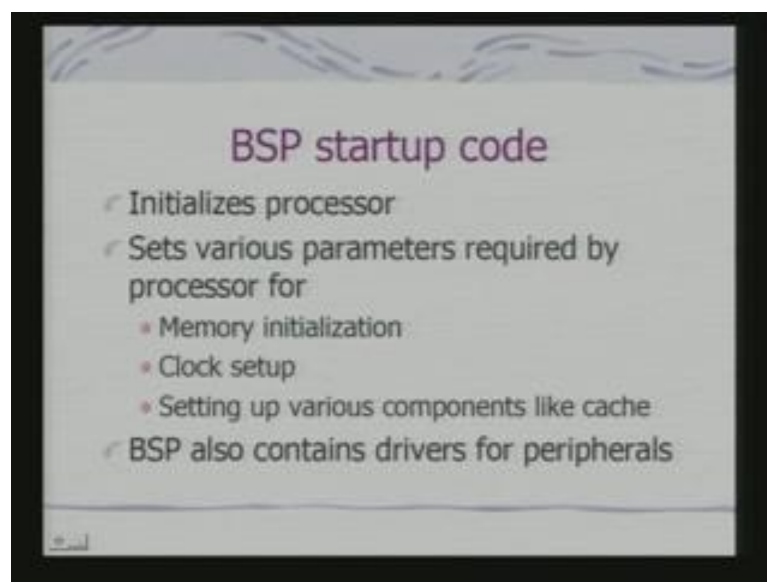
(Refer Slide Time 12:54)



What is this BSP or OS initialization? BSP is a component that provides board or hardware specific details to OS for the OS. To provide hardware abstractions to the tasks that use it is services and BSP is specific to both the board and OS for which it is being written.

In fact, this can be configured also for the board for which it is being written. And if you see, what it takes about is, it provides board and hardware specific details to the operating systems.

(Refer Slide Time 13:26)

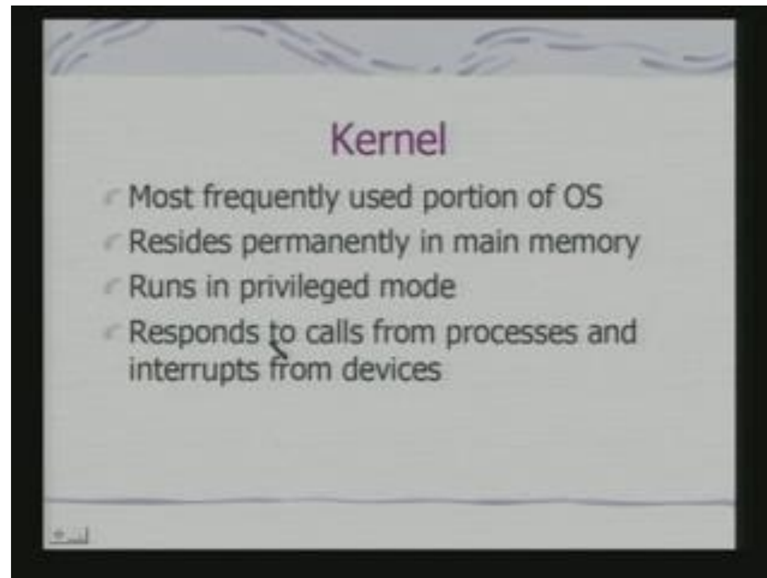


So, the basic component of the BSP is a startup code. And we are already discussing the startup code for a RAW system and what kind of functions it would support. So, it basically initializes the all the components of the system. Does the memory initialization does a clock setup and setup various components like cache.

And BSP also contains typically drivers for the peripherals which now becomes memory resident. So, these BSP are the Board Support Package forms a very important component for all operating systems which are targeted for embedded appliances. And for each appliance is the target card you may need to configure your board support package for a particular OS.

So, for example, if I am talking about ecash, ecash is a OS targeted for embedded appliances. But, for a particular card I need to properly set the board support package. So, that on that card or the target system ecash can operate properly.

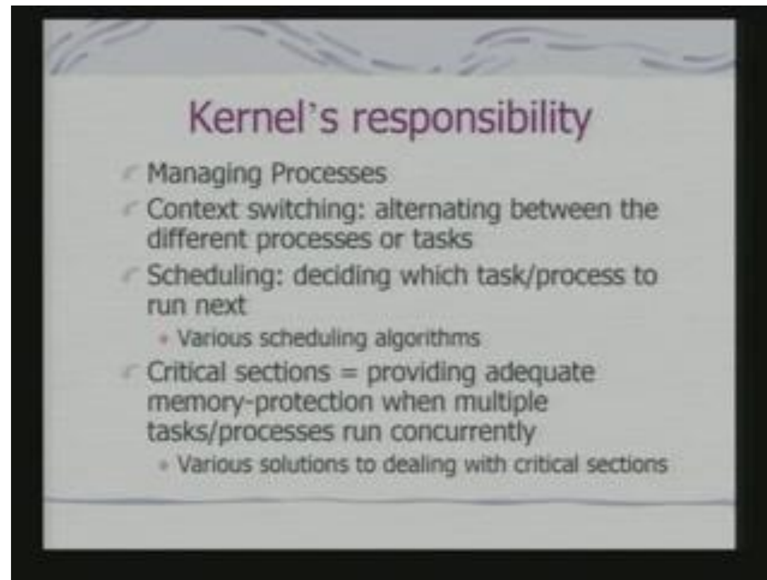
(Refer Slide Time 14:44)



Then the control; obviously, gets transferred to the kernel the OS kernel. OS kernel is the most frequently used portion of OS resides permanently in main memory. That means, in the RAM area and even if you are implementing a kind of a virtual memory between flash and a RAM possibly, we would not like the kernel to be part of that kernel would be typically lock in the RAM area of the embedded system.

It runs in a privilege mode, because it supports the system functions. So, if you remember ARM processor. ARM at various modes of which the privilege modes or system modes. So, the kernel on an ARM would run in privilege mode. And it response to call from processors and interrupts from devices. These processors could be the application processors. So, they can make the system calls to make use of OS services.

(Refer Slide Time 15:39)

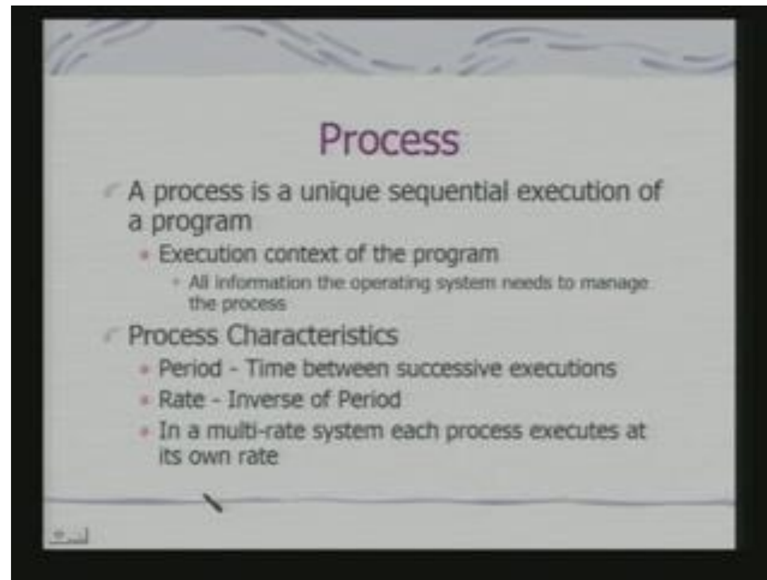


So, what are the responsibilities of kernel? Kernels basic job is to manage processors. In fact, all processors are created by the kernel. In fact, they are deleted or killed by the kernel as well. Since there are many processors and we are talking about multi tasking system. And therefore, what is needed? The kernel needs to manage, what is called context switching, alternating between the different processors or tasks.

This read to this concept of scheduling. When we are context switching which process or task should run next. This is the principle of scheduling and kernels implement a variety of scheduling policy. Then we have critical sections, providing adequate memory protection, when multiple tasks processes run concurrently, because there may be shared resources.

If there are shared resources shared by two or more processors then access to the shared resources have to be disciplined. So that one of them, are is really using the shared resource. That job is also handled by kernel and there are various solutions dealing with critical sections and these solutions differ from OS to OS.

(Refer Slide Time 17:20)

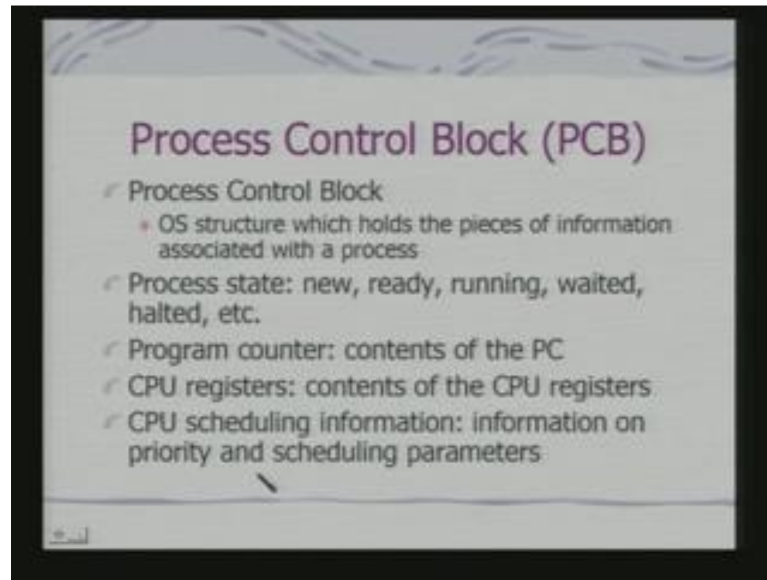


So now, we are talking about multitasking and processors selects formally look at, what a process is? Process is a unique sequential execution of a program. Now, if a program even in a general purpose system is same program if it is being executed by two different users. We actually shall have two different processors associated with the same program.

Even the same program instantiated into two different processors, but the same user will also have distinct existence. So each process therefore, has got an execution context. So, all information this execution context is collection of all information the operating system needs to manage the process. And, what are the different process characteristics? One will be period that is a time between successive executions.

Because, we said that there can be periodic processors associated the period is rate, inverse of period is actually the rate. And in a multi rate system each process would executes at its own rate. So, this information is important for the kernel as well, because kernel schedules the processes through context switching. So, this is one aspect of the information about the process.

(Refer Slide Time 18:44)



In fact, this all these information about the processes as stored in what is called a process control block. Each process has got it is own process control block. This is the OS level data structure which holds the pieces of information associated with the process. And the process has got different states new, ready, running, waited, halted etcetera and what are the states depends on again the definitions each OS really adopts.

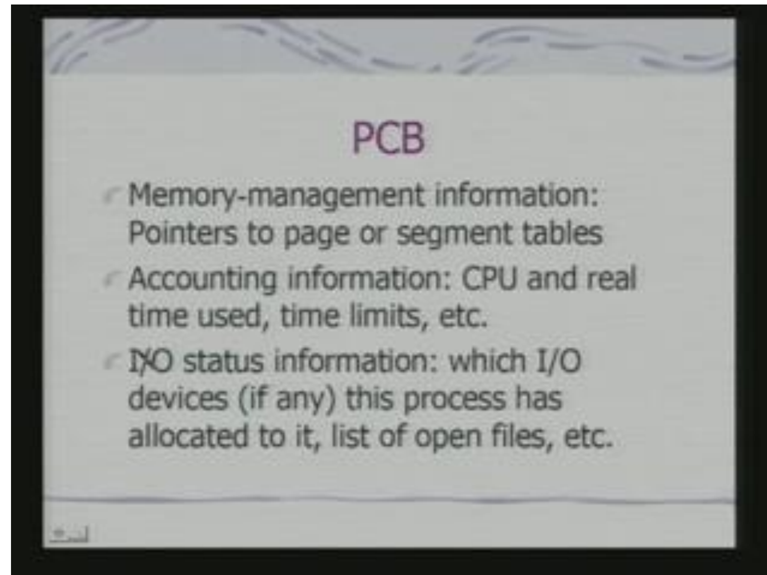
In a generic sense, when a process is admitted, it becomes a new process. A new process may not be always ready to run after some transformation meeting some other requirements. And the demands of the new process it becomes ready to run. When the process is actually in execution it is state is running. When the process is waiting for some external input or access from the device it state is waited.

The process can also be halted for some reason and it can be also put to different kinds of other states. In fact, in a general purpose a system a process can be moved out of the memory and put to the hard disk waiting for it is time to come. So, there may be variations of these states that we have talked about. So, this process state information is stored in process control block.

Then, the program counter program counters points to the current instruction of the process which is being executed. So that will be stored in the PCB why, because when there is a context switch. These PC value has to be restored. Then also, the CPU registers which captures the state of the process. And also the CPU scheduling information

priority and scheduling parameters for which the period as well as the rate can be important for the OS as well.

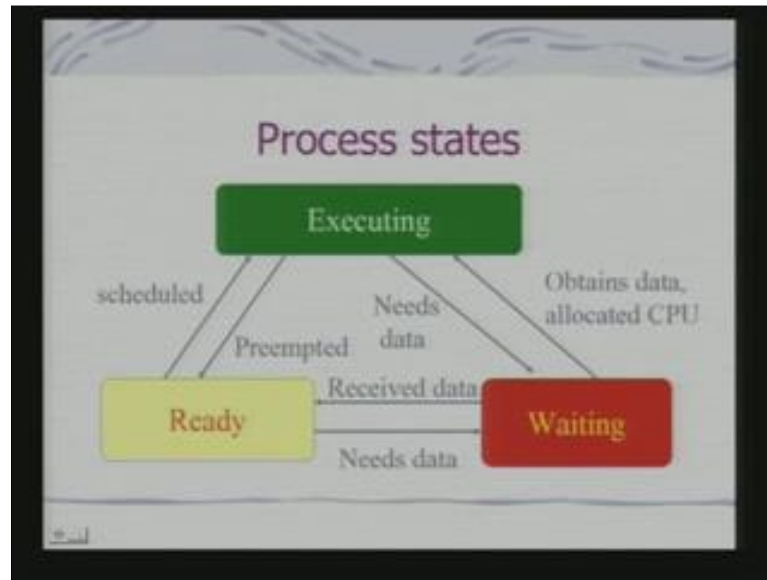
(Refer Slide Time 21:04)



It also has memory management information. We had already talked about in the context of memory that your memory can be organized in terms of pages or segments. And there has to be page table for each process. So that pointer to the page table becomes part of your PCB also accounting information.

As well as IO status information; that means, which IO devices if any this processes has been allocated to it. If it is using a file system the list of open files extra, because this records the resource usage information, resources other than that of CPU and memory.

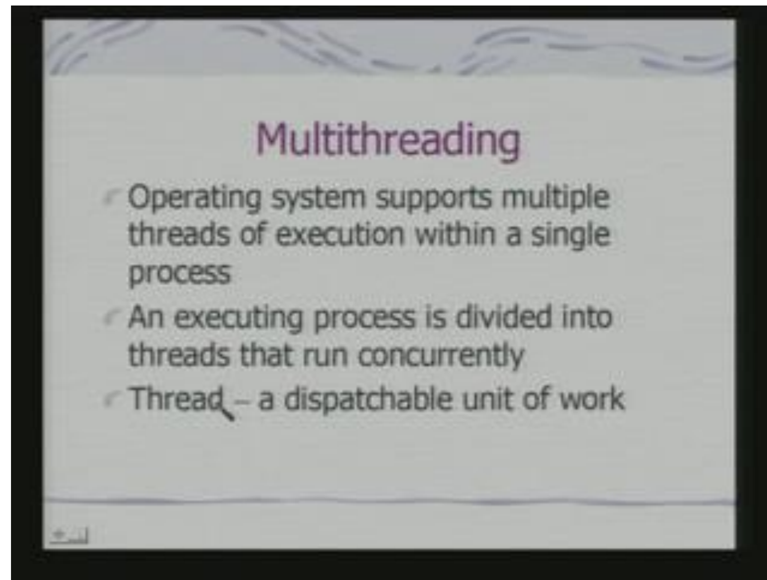
(Refer Slide Time 21:47)



So, if you look at typically a process state transition diagram, this a generic diagram. A process is in ready state from ready it goes to executing state. If it needs data, so it goes to the waiting stage and then again, it can come back to the ready state before it gets scheduled.

In fact, the OS actually schedules processes which are in ready state. So therefore, all such processes join what it commonly called a ready queue. And from the ready queue OS picks up the process which is to move into the executing state and, these process is also known as dispatching.

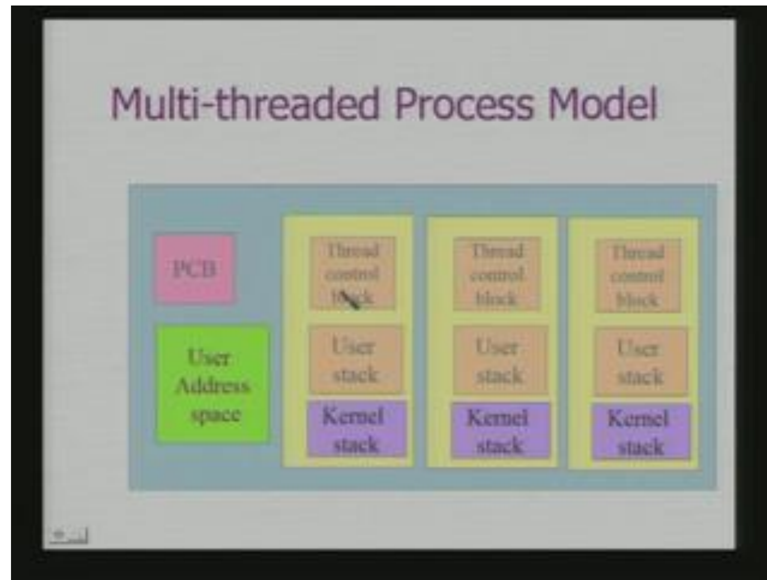
(Refer Slide Time 22:30)



There are many Operating Systems which also support tasks or concurrency beyond the level of processors. So, what we say supports multiple threads of execution within a single process itself. Because, what is a thread of execution? A thread execution is a sequence of instructions being executed. And many of the OS is support multiple such sequences of instructions to be managed concurrently with in the boundary of a process.

So under those conditions, and what happens, an executing process is divided into threads that run concurrently. And thread becomes dispatchable unit of work, what is dispatchable unit of work? A schedulable unit of work, what I told you is that, if something is a ready queue from the ready queue. When we put that into an executing state it becomes the process of dispatching.

(Refer Slide Time 23:39)



So, if you looking to this model, then what we really find, for a process I have got a PCB or Process Control Block. And along with the process that is, what is associated and user address space that, the address space of the process. And in these address space of the process threads relief. Threads are concurrent sequence of instructions concurrent threads of control. So, you have got with each thread just like PCB a thread control block as well.

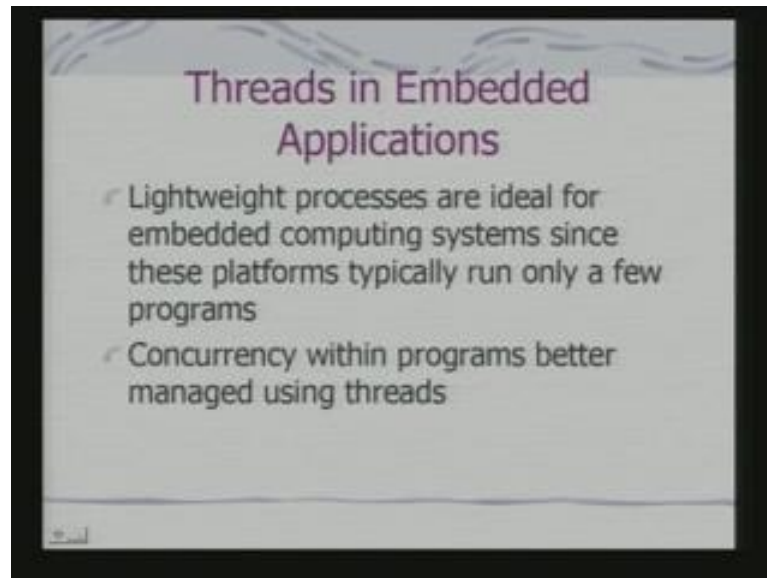
And, what is interesting and important to note is that, each thread can have its own local variables, because its following own sequence instruction sequence. So, for each method it invokes along a thread there will be local variables. There will be parameters pass to those methods. There be written addresses corresponding to those methods. So, each thread will have its own stack.

User stack each thread will have its own portion of the user stack. Now, apart from the user stack there will be also the kernel stack. Because, kernel stack what is kernel stack? Kernel Stack is a stack which is not really resident in the User Address space not strictly in the User Address space. It is in the address space which is being managed by the Operating Systems.

So, let us say if this thread is under execution and an interrupt occurs. So, the Interrupt Service Routine would be process outside the boundary of this process. So, when that transfer takes place. The information the current state of the thread has to be saved,

where will it be straight? It would be saved in the kernel stack and not in the user stack. User stack will typically maintain the local variables parameters corresponding to precede your calls.

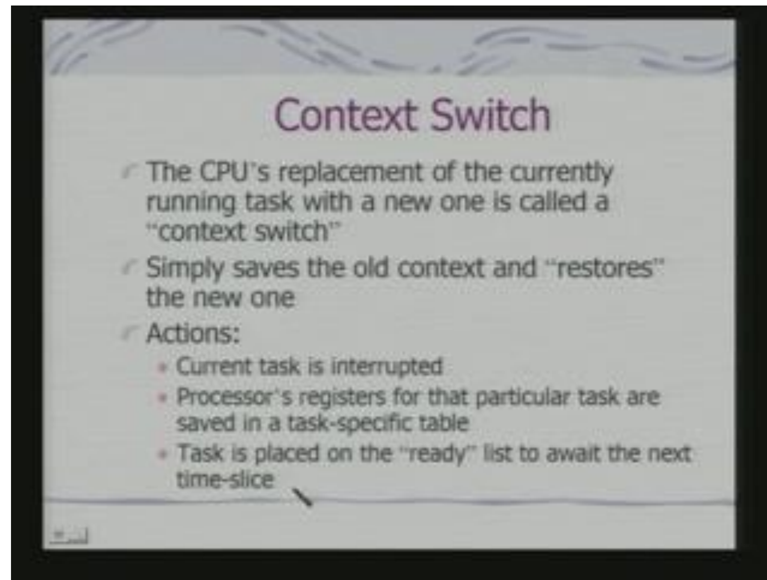
(Refer Slide Time 25:57)



In fact, light weight processors or threads are ideal for embedded computing. Since this platforms typically run only of few concurrent programs. So, the application level concurrency is expected to be handled more through the threads. And concurrency within programs becomes better managed. Because you can understand also, why should it be better managed, because the moment I switch from one thread to another, what happens, the amount of information that I need to change is much less.

Because my PCB, there is Program Control Block is remaining same. And that is why switching between thread to thread is much faster compare to switching between processors. And that is the reason why, you will find all these OSes which are targeted for Embedded Applications do support multithreaded processors.

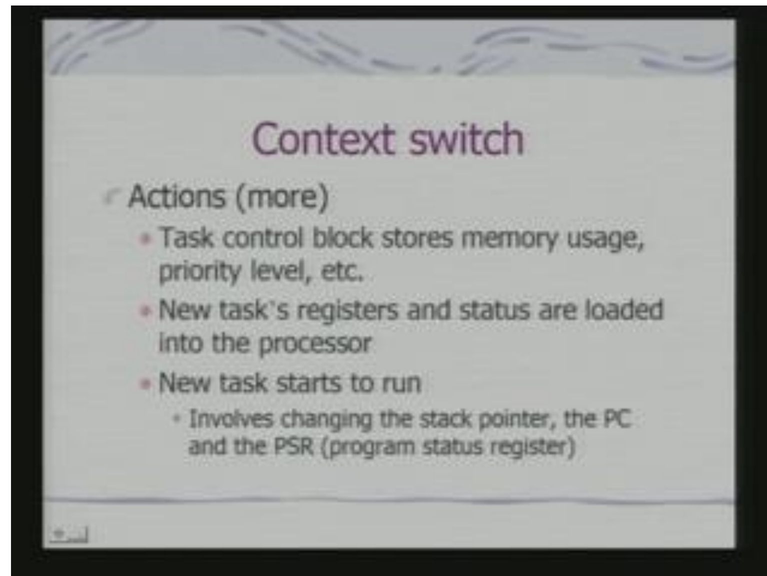
(Refer Slide Time 26:59)



Obviously, related to all these things is context switch. So, what is context switch? Context switch is CPU's replacement of the currently running task with a new one. Now here, you need to save the old context. And restore the new one and so the actions, what are involve in a context switcher current task is interrupted.

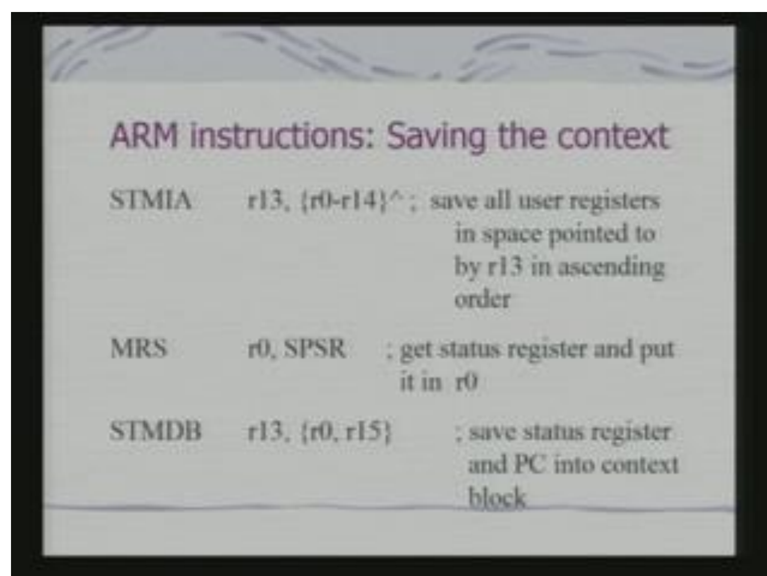
Processor registers for that particular task are saved in a task specific table. In fact, this information is typically will be in PCB. In fact, PCB has to be saved in a task specific table. And the task is placed on the ready list to avoid the next time slice. So, slice; that means, next time when the OS schedules the task.

(Refer Slide Time 27:44)



So, task control blocks stores memory usage priority levels all these information. And new task's registers and status are loaded into the processor wire from they are loaded. From the PCB of the process which is currently being dispatched. And then, new task starts to run. And all these things involve changing the stack pointer the PC as well as program status register. So, you can understand that all these context switch; obviously, involves and overhead in terms of the instruction cycles which are involved for this job.

(Refer Slide Time 28:25)

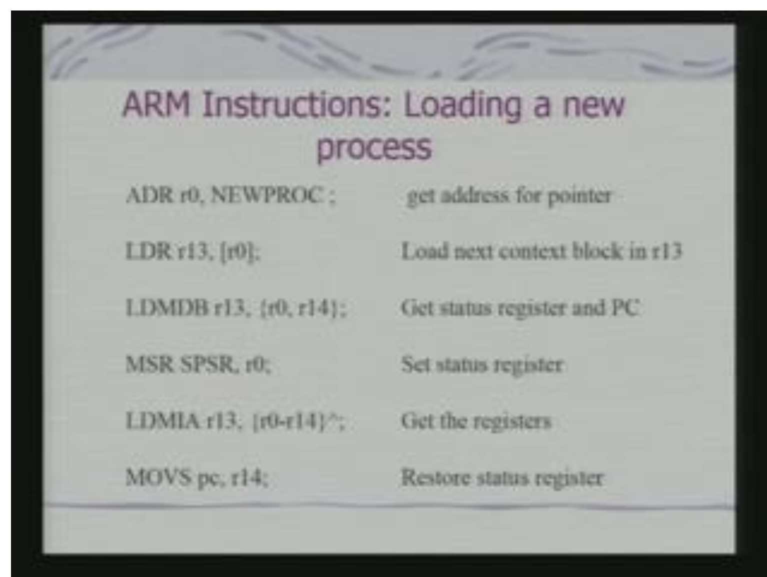


Let us look at simple ARM instructions, because there are supports in the processor also for facilitating this kind of a context switch. The interesting thing to note is this, these are all multiple memory access instructions. In fact, multiple blocks of memory is accessed. So in a typical in this case, the register R0 to R14 is being saved point in space pointed to by R13 in ascending order. That is depending on the post operator that is being associated.

Now, what is the interesting feature of this, when is multiple memory transfer take place through an instruction. This process cannot be interrupted via on interrupt and that is absolutely essential. When a context switch takes place the context switch actions should not be interrupted by an external interrupt. If it gets interrupted, then actually there would be an inconsistency. Because, another process block comes into the picture, because your Interrupt Service Routine would corresponds to a another process.

In a standard processor, you need to disable the interrupt before you can execute this multi byte transfers. Because, you are PCB's would occupy multiple memory locations. One reason by ARM provides this kind of multi byte transfer a multi memory location transfer instructions is to facilitate this kind of context switch.

(Refer Slide Time 29:59)



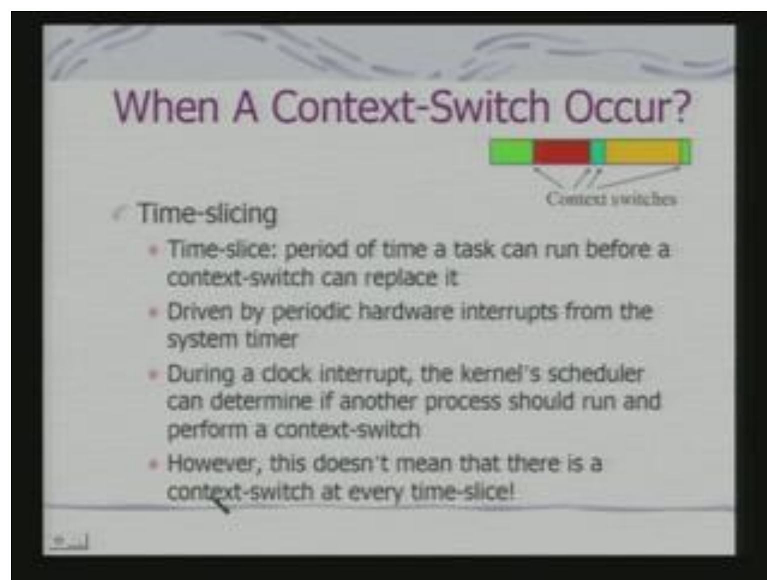
And the other thing is that the same thing happens when you are loading a new process. Let us look at the sequence of instructions of what how it works out. You get the address for pointer, pointer for the PCB of the new process. Then, you load the next context

block in R13. Because that is, where it should point to the pointers stack pointer. Get your status register and the PC, because that is from the location you will be getting. Then, the get the status register, you have to load the SPSR currently on to the, from r0.

Then, you get the registers load the registers. And then move the R14. Basically the PC current PC would be the value from the link register. And so your new process gets ready for execution. So, this is just a sequence of codes for loading a new process. This type of code will be part of the kernel.

And you see that your processor also supports variety of these instructions. This multiple memory move instructions to facilitate this kind of actions. So, this is what is context switch, context switch always have some kind of an overhead. So, if you need to move small number of bytes for a context switch. Context switch will be faster and that is precisely, why context switching involving thread is faster, then that involving processors.

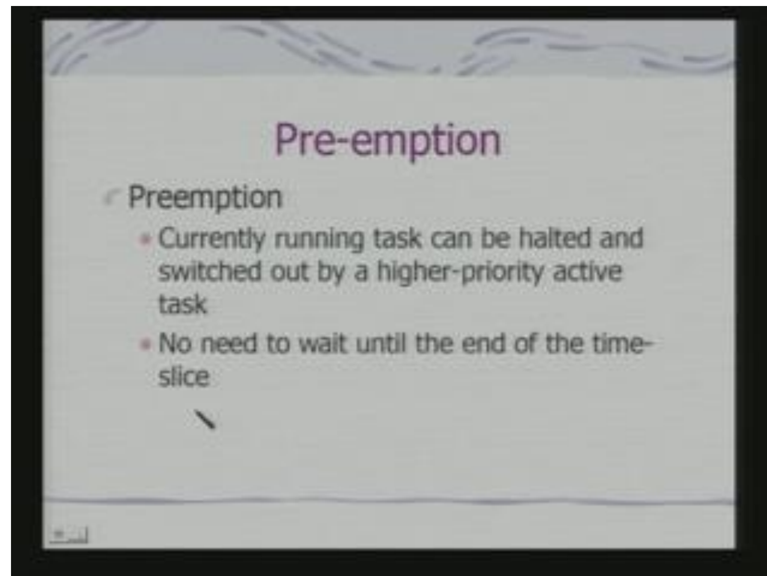
(Refer Slide Time 31:37)



So, when a context switch occurs; obviously, when a time slice expires. Then, it can be driven actually it would be driven by periodic hardware interrupts from the system timer. And during the clock interrupt the kernel scheduler can determine if another process should run and perform a context switch. In fact, when such an interrupt occurs which comes into play is a scheduling routine of the kernel.

And this scheduling routine finds out from the ready queue which process to run using variety of scheduling policies, if this scheduling routine becomes complex that also introduces overhead. However, this does not mean that there is a context switch at every time slice. If multiple time slices are allocated to a process there one be a context switch at each time slice.

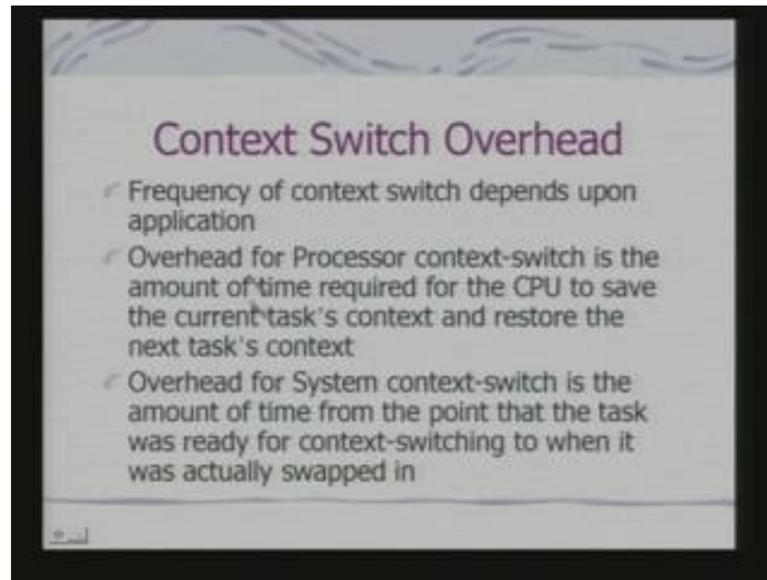
(Refer Slide Time 32:32)



And pre-emption it is, what we are already talking of preemption it means that, currently running task can be halted and switched out by a higher priority active task. And no need to wait until the end of the time slice this is fundamentally important. Pre-emption, if it is really preemptive scheme, then I can do preemption even when the time slice has not expired.

So, we are already talked about preemption, but let us try to understand now preemption in the context of exact operational mechanism. That is preemption means forcing out a process even when it is time slice has not expired.

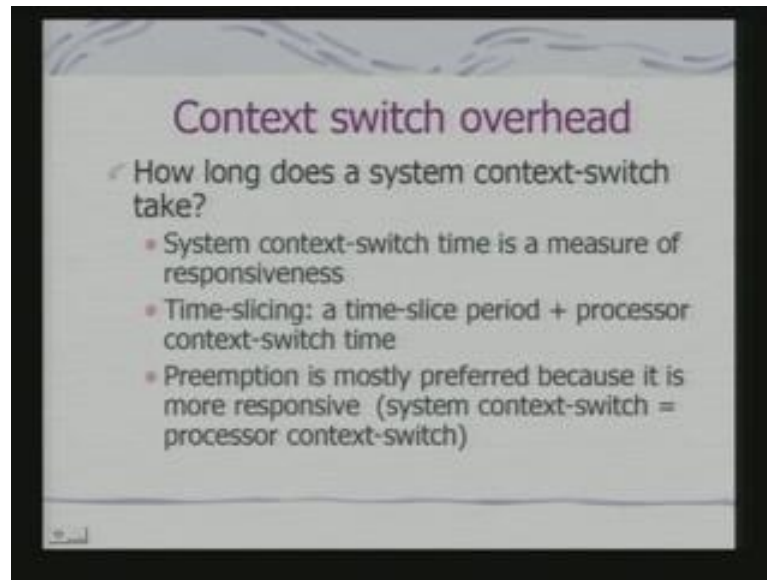
(Refer Slide Time 33:16)



So, context switch overhead we had already have a ID about it. And therefore, frequency of context switch depends upon application overhead for processor is the amount of the context switch is the amount of time required for the CPU. To save the current tasks context and restore the new tasks context. And overhead for system is context switch is a amount of time from the point that the task was ready for context switching to when it actually swapped in. So this is a two kind of time, we are looking at.

This is the time when there is a decision to context switch. And the time actually consume by the processor in saving and restoring the context. And this is the time from the point where task becomes ready to actually the task being swapped in. So, there are these two overheads; obviously would like to have this frequency of context switch minimum. So, these overhead is also minimum for an Embedded Appliance.

(Refer Slide Time 34:24)



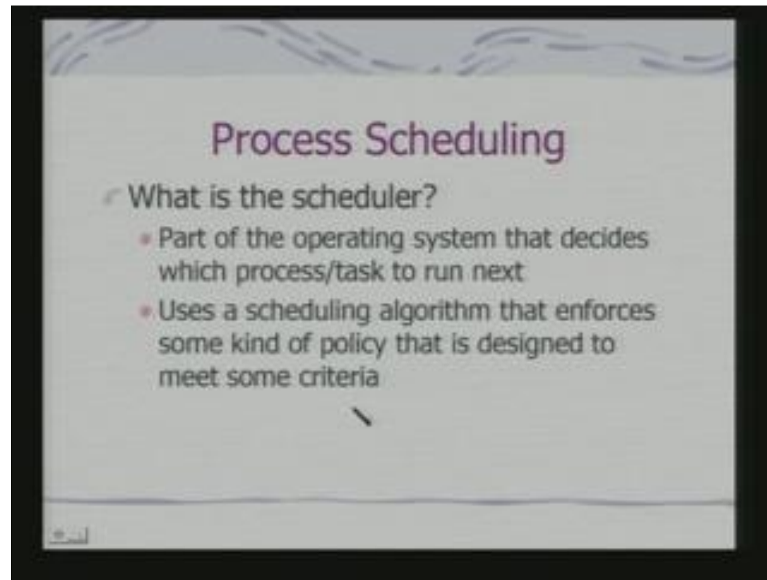
So in fact, system context switch time is called is a measure of responsiveness of the system. And the time slicing a time slice period plus processor context switch time is exactly. The switching time that may be their between the two processors; that means, why in a processor execution it is allocated a time slice.

Next time, another process can come into being, only when the time slices expired. And the context switch has taken place. So, this time altogether measures a time. Pre-emption is mostly preferred, because it is more responsive why, because you do not wait strictly for the time slice to expire. You do not wait strictly for the time slice to expire; you only pay for the context switch time.

In fact, if you looking do it these gives was an idea of what happens, when an interrupt occurs. Interrupt is what always a context switches, when an OS is managing the interrupts. So, the context switch time is actually the time required for saving the registers and loading the new registers. And therefore, that response time is gets determine by this time period.

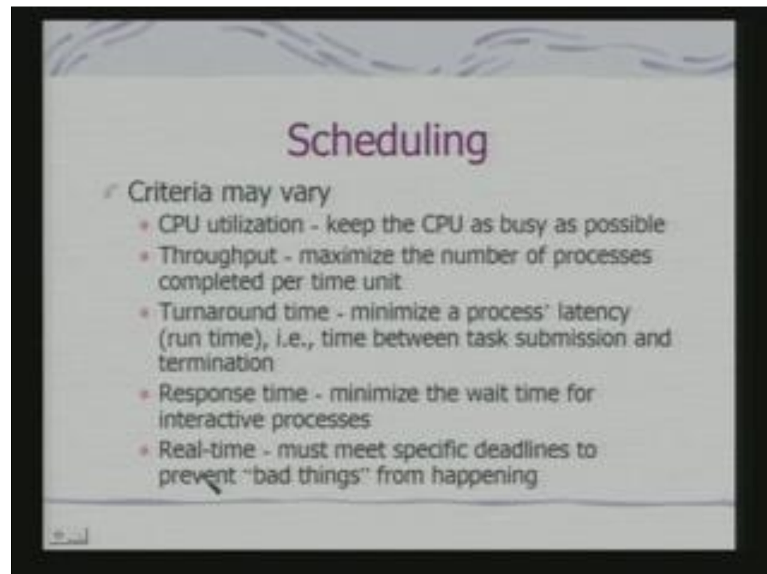
In fact, if you remember ARM architecture, when I talk about FIQ, we had a copy of the registers. So under those conditions, when we doing a context switch that many registers need not be saved and loaded. I have a fresh copy them and therefore, for that mode or that hardware signal the response time will be much less compare to other modes.

(Refer Slide Time 36:12)



So, now at context switch the OS routine schedules the things. So, what is the scheduler? Scheduler is that part of the Operating System that decides which process a task to run next. It uses a scheduling algorithm that enforces some kind of a policy that is design to meet some criteria.

(Refer Slide Time 36:30)



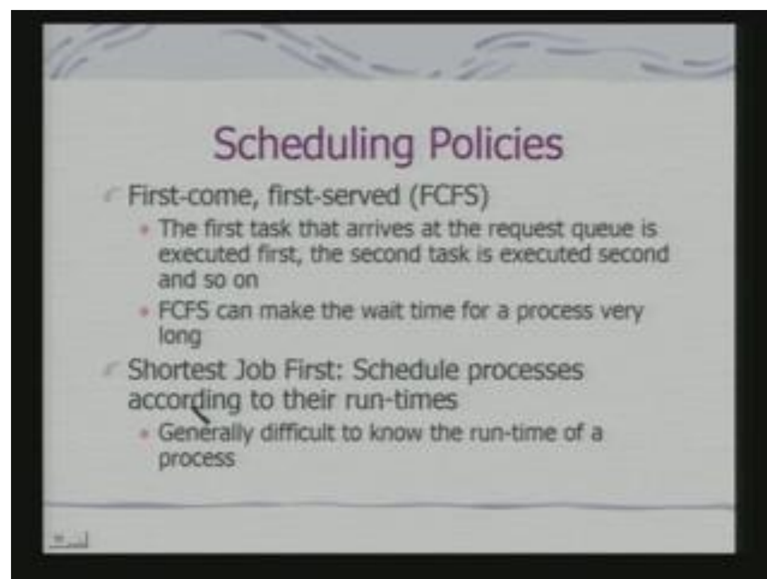
Now, this criteria may vary one of the most important criteria would be CPU utilization. Get the CPU as busy as possible. If a process is doing an active it is waiting and currently no process are running really on the CPU. Then, there is a CPU is wasting time.

Throughput is maximize the number of processes completed per unit time. Turnaround time minimize a process's latency that is the time between task submission and termination.

Now, even if you maximize the throughput the turnaround time may not be minimized. Because, it is depends on how the process have been tilt with response time is minimize the wait time for interactive processes. And real time would be that specific dead lines must be met. We are already talked about hard and soft deadlines. And those must be met.

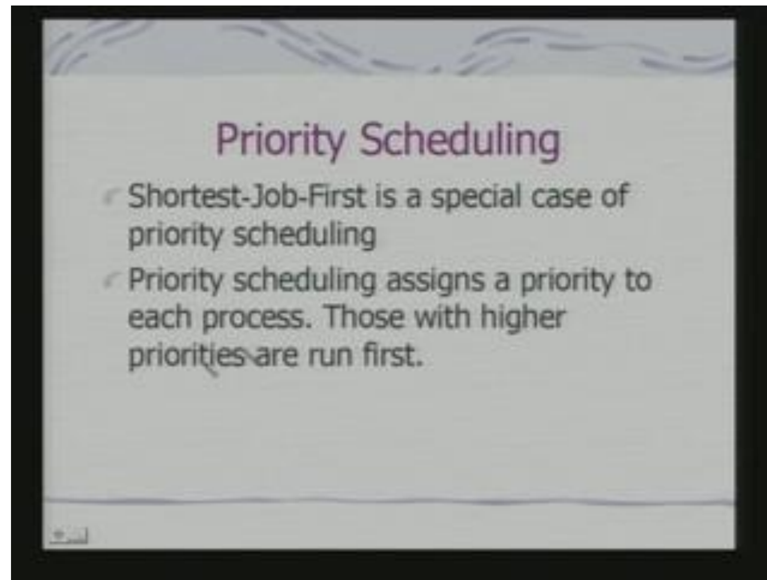
So, when we are looking at the real time. We may suffer from the problem of low throughput or may be less CPU utilization, in order to meet the real time constraints. So, all these criteria it is not that all these criteria's can be met. Simultaneously by following a policy, a policy determines which criteria to be met for the purpose of scheduling.

(Refer Slide Time 37:55)



Standard scheduling a policy is which you find in most of the computer systems. And the OSes are First Come First Serve. Then, you have got the shortest job first schedule processors according to the run times; obviously, this is extremely difficult to do because you really do not know the runtime of the process occur.

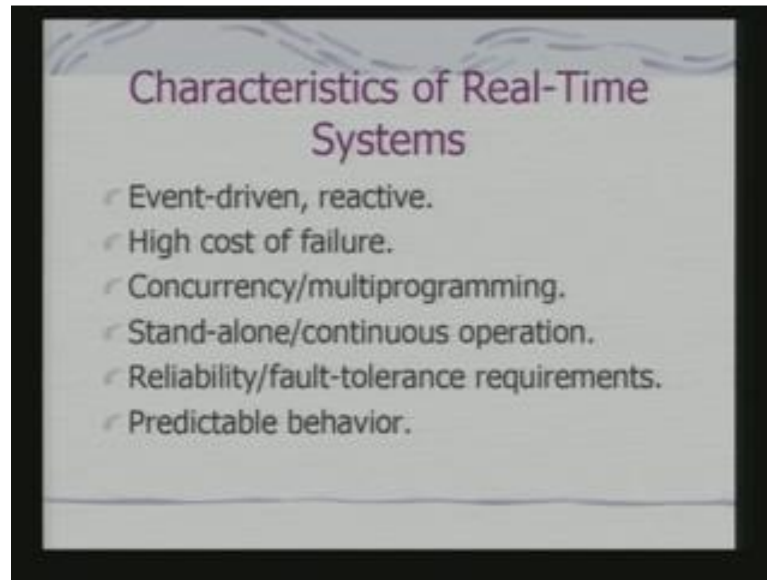
(Refer Slide Time 38:24)



And this whole thing is based upon also, what is called priority scheduling? Your shortest job first is a special case of priority scheduling. It means process can have associated with priorities. And the higher priority process will be schedule, if it is in the ready queue. So, priority scheduling assigns a priority to each process and those with higher priorities are run fast. But, obviously, these kind of a scheme scheduling polices cannot meet our real time requirements.

So, Real Time Scheduling would be different and scheduling policies would be different. In fact, the most fundamental characteristics of a real time OS is the scheduling policies that the OS follows.

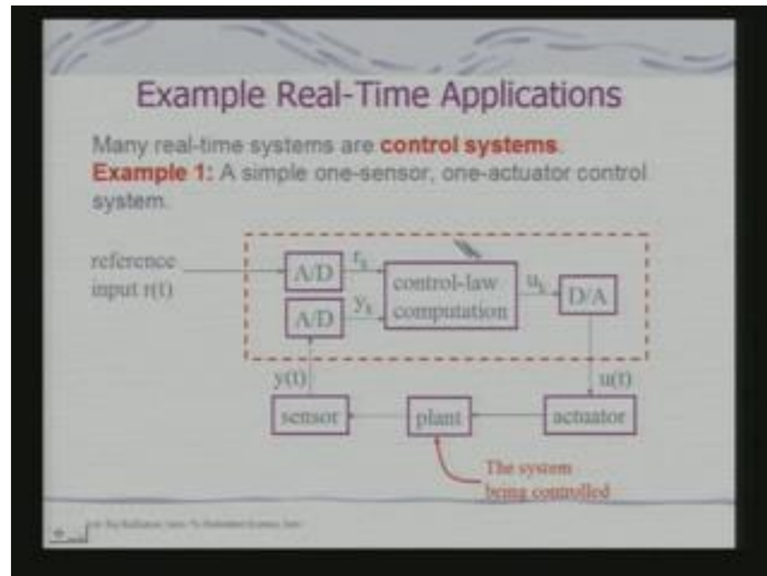
(Refer Slide Time 39:12)



To meet the deadlines, what are the characteristics of real time systems? Let us have a quick review Event driven reactive. There is a high cost of failure, we must meet the deadlines. There would be concurrency and multiprogramming, because there are multiple concurrent streams in the external world. That is also stand alone continuous operation without any kind of user interventions.

Reliability and fault tolerance requirement, I would like that to run reliable. So, I am also predictable behavior. We should not because it depending on the system load, the scheduling of the processors can vary in a general purpose OS. But, I would to like to have for a real time OS a predictable behavior. The processor should get scheduled predictably, so that the deadlines can be met.

(Refer Slide Time 39:59)

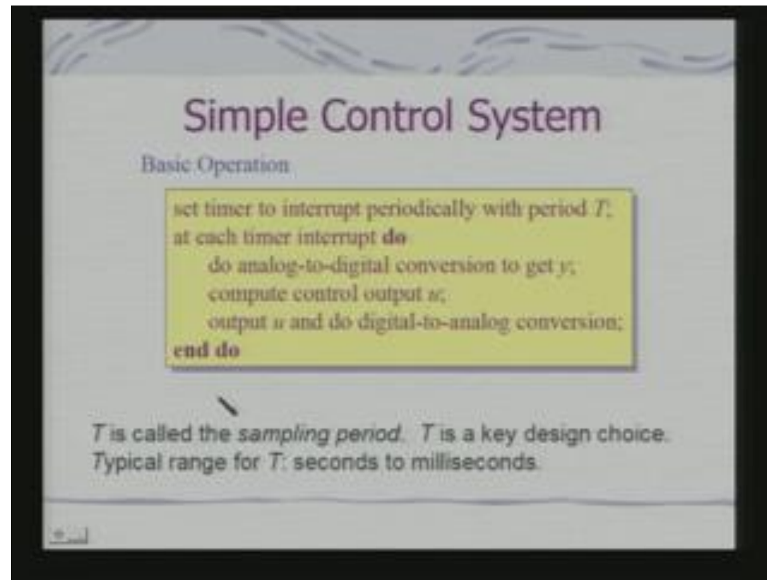


So, let us look at some examples and with reference to the examples. We shall understand how the constraints emerge. So, we are familiar with control systems these examples of a very simple control system. We have got one sensor one actuator control system. So, this is a reference input, this basic controller computation.

And so, your input comes in I buy is an analog input which gets to analog to digital converted, these a reference input. And this is the output of the plant which you are trying to control. So, you have these two inputs for the control block which implements a control algorithm. It would produce the control output which would activate the actuator for controlling the system. This is a very simple generic control system model.

Now this constraint, if you are implementing this. Then, there would be timing constraint on these. And, how does the timing constraint comes in? Timing constraint comes in because your input is being sampled at regular intervals. And you are taking control action in response to that sampled input.

(Refer Slide Time 41:15)

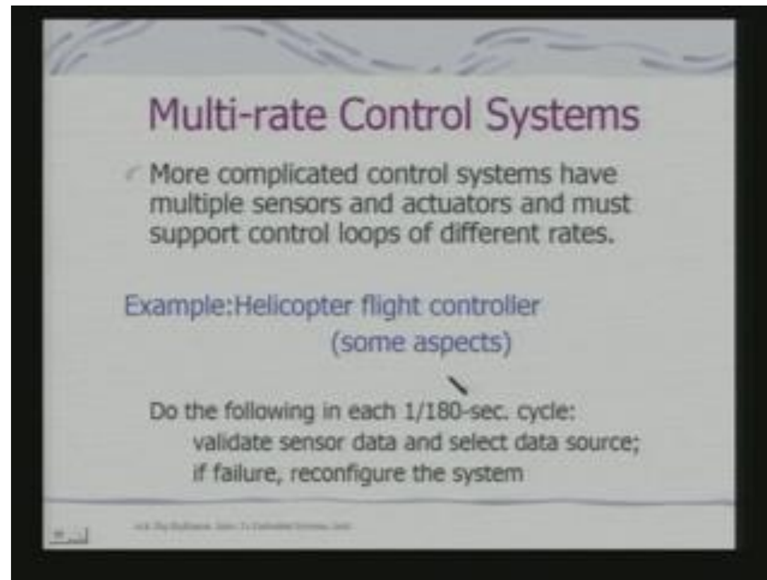


So, what we say the basic operation is set timer to interrupt periodically with period t and at each timer interrupt do analog to digital conversion to get y. Compute control output u output u and do digital to analog conversion end do. This is the basic functions, which happens and so your control law is implemented, where a microcontroller likes a PIC. And you have set the timer on the PIC to interrupt, so that this control action takes place at regular time.

And what is required is? That these action is to be done, satisfying the constraints timing constraints. So, what we say, T is called the sampling period. And T is a key design choice. It may be seconds to milliseconds, because at the same time. There may be other processes running which may take the users input for controlling the plant may be setting the temperature.

So, when we are scheduling these routine must made the deadline, what is a deadline? Deadline is given by the period of the sampling. So, this job is a periodic job with the deadline determine by the period. I should not miss any input sample.

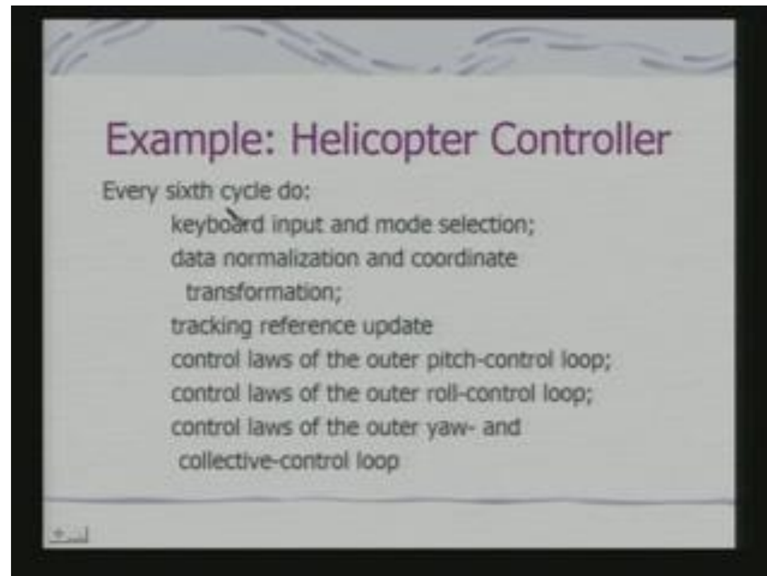
(Refer Slide Time 42:32)



Then, we have Multi-rate Control Systems. Let us consider an example, what we say it is a Helicopter flight controller. So, a typical Helicopter flight controller can is it will have sensor. Let us say this just an example at each cycle, so 1 by 180 second cycle. Validate sensor data and select data source if failure reconfigure the system. So, I am taking data at this sampling period.

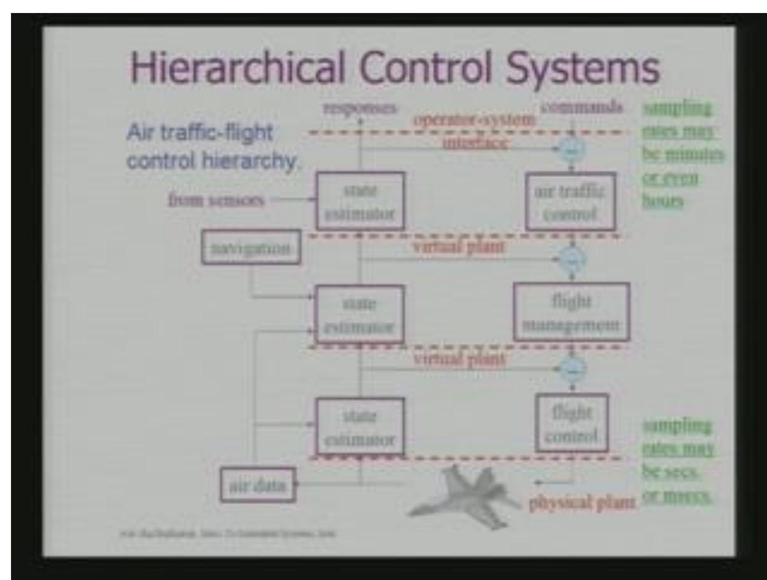
But, I might not really like to have the control being at worked out at the same frequency, because control output can be generated in a slower rate that is slower rate. Because, I would like to may be look at the inputs over a number of sampling periods to decide upon a control action.

(Refer Slide Time 43:27)



So, the control action may be generated for every sixth cycle. So therefore, the whole system becomes a multi-rate system, because you are sampling the inputs. You are checking for the fault of the inputs at a particular rate and you are generating a control action at a different rate. These a basic difference between the first system, which was doing the basic control action at the sampling period itself.

(Refer Slide Time 43:56)



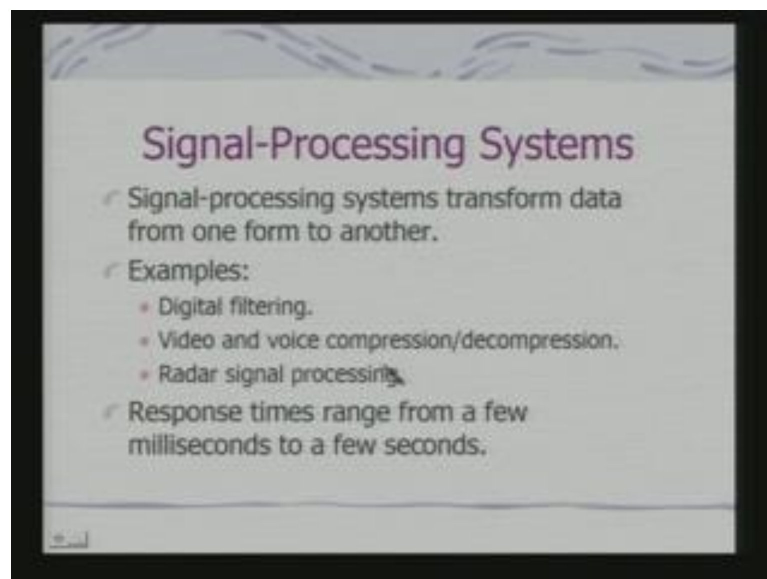
Then, we can have a Hierarchical Control System. This is a typical Air traffic flight control hierarchy. Here, when the operator is interface with the system and is working.

The sampling rates may be minutes or even hours. And when you come down at this level the physical plants sampling rates may be seconds or milliseconds.

And what is happing here, you basically take the various inputs to a stack estimators. And on the basis of that you decide upon the control action. So, each of these control layers of the control actions have to be executed at multiple time periods. The timing constraints are multiple, because all of them meet to satisfy your sampling intervals and their hierarchal and there not independent, because one is based upon the other.

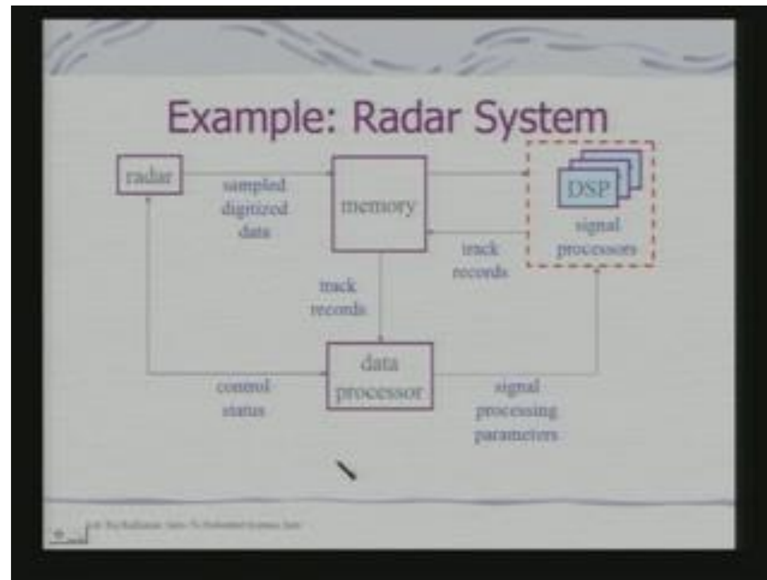
So in this case also, we have got a multi-rate, but multi-rate, but hierarchal system. So, the issue is that, when we are talking about OS managing all these things. It beans that it has to manage processors with respect to multiple deadlines. And further requirement is that there all periodic processors, because your sampling at fixed period at multiple levels.

(Refer Slide Time 45:13)



Single Processing Systems also has got various constraints associated with it. Because, their also sampling the input. I cannot miss a voice sample. If I am sampling the voice at a particular rate, I must compress satisfying that sampling rate. So, response times can range from a few milliseconds for few seconds.

(Refer Slide Time 45:35)

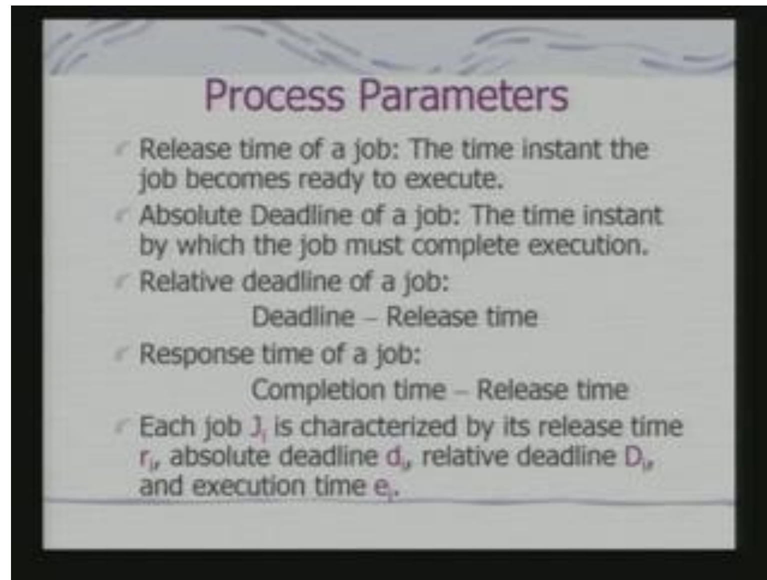


Let us look at an example Radar System, which is a very common defense based applications. So here, what you are doing? Your tracking you have to track depending on the radar input you need to track. So, all these things and these data goes to other DSP. So if you look in to it, this is a separate DSP processor. This is a separate data processor. And this data processor is providing, the signal processing parameters to the DSP.

So, if you have the OS, then OS is managing what OS is managing both DSP as well as your data processor. And it needs to schedule processors at the two processors at two different rates depending on the timing constraints the system imposes. Now, this data processor also needs to take care of the control because your radar may be moving from one direction to other direction.

So this is how, what we trying to illustrate that, how the different kinds of timing constraints emerge from the real life requirements. And OS needs to meet these kind of timing constraints.

(Refer Slide Time 46:49)

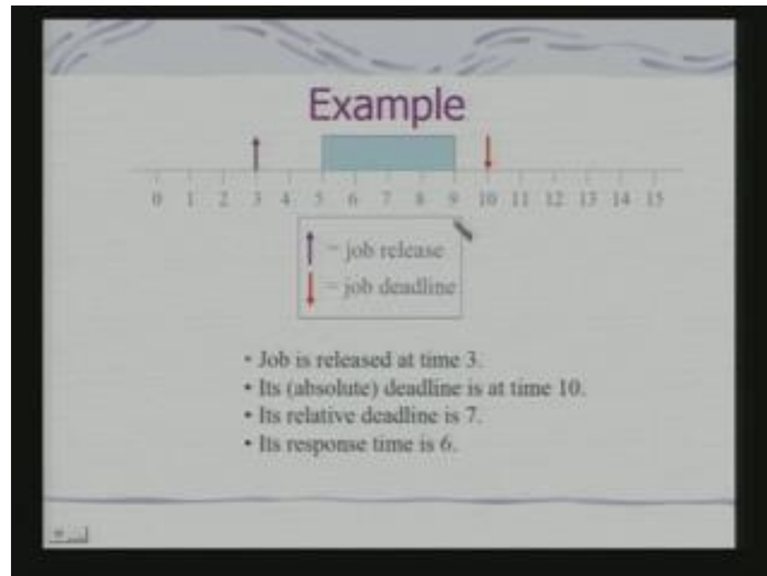


So, what will be the process parameters, if this is the kind of scenario, first what we say release time of a job. The time instant the job becomes ready to execute. Then you have got absolute deadline of a job. The time instant by which the job must complete execution. Then you have, what is called a relative deadline of a job, what is relative deadline? Relative deadline is a deadline minus the release time.

Then, you have got response time of a job, what is really response time? The completion time minus the release time, because is completion time. If I am really satisfying the deadline would be before the deadline or just coincident with the deadline. And therefore, a each job Ji is characterized by its release time ri absolute deadline di relative deadline dj DI and execution time ei.

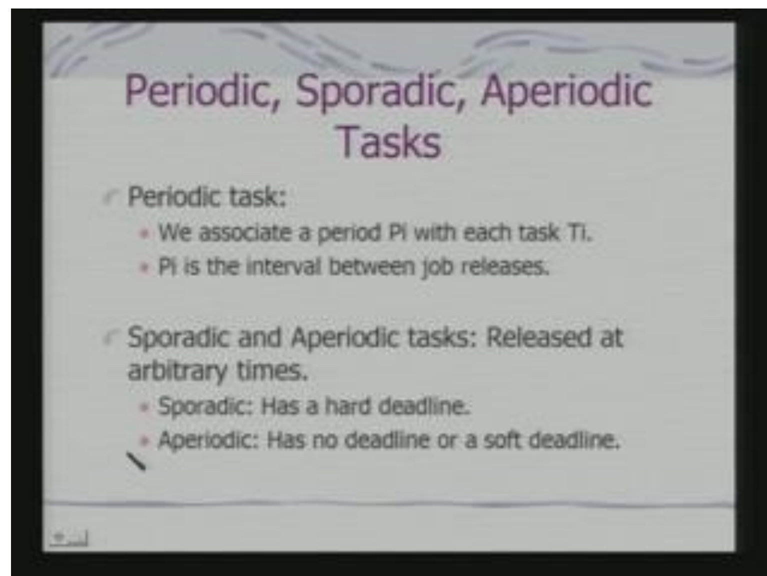
So, if you remember the examples at, we have seen there would be jobs with multiple there will be multiple jobs. And each job will have different parameters. Further, simple control scheme there would be same parameter, but when we are looking at multi-rate controls hierarchy controls or even radar processing system. The jobs will have different parameters depending on the scenario.

(Refer Slide Time 48:16)



So this is an example that, this is where a job comes in this is the release time. This is where I have got the job deadline. And this is a time period, where the job gets executed the way schedules gets executed. So, it has got a response time which is 6 because 9 minus 3 gives you the response time.

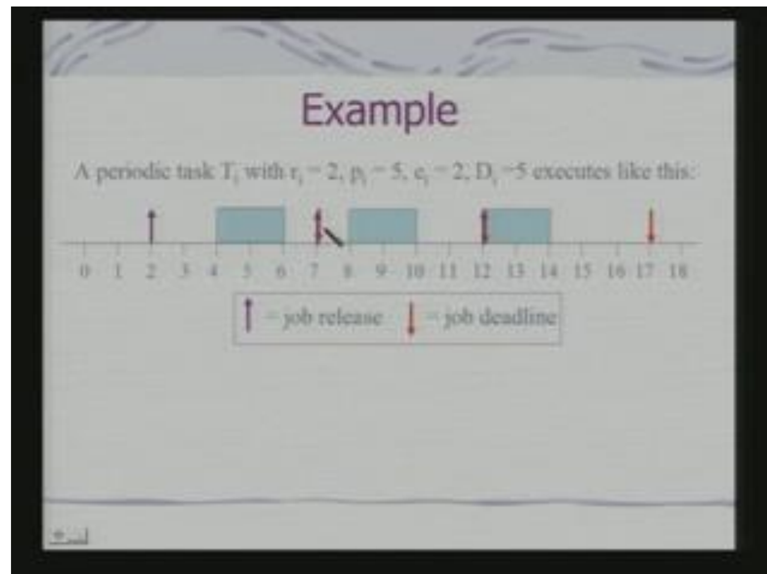
(Refer Slide Time 48:39)



Then, the tasks can be periodic. So, we associate a period Pi with each task Ti and Pi is the interval between the job releases. Sporadic and Aperiodic tasks which are tasks released at arbitrary times. And we distinguish between them, because a Sporadic has got

a hard deadline. And Aperiodic typically has no deadline or a pretty soft deadline. So, this is an example of a periodic task.

(Refer Slide Time 49:13)



So, if I and the tasks get released at regular intervals and what I have shown is the deadline coincides with the release. Because, these task related to these data must be finished before the next job arrives. And this is true for maturity of the sampling driven tasks, because before the next sample arrives the pervious sample should be processed.

(Refer Slide Time 49:40)

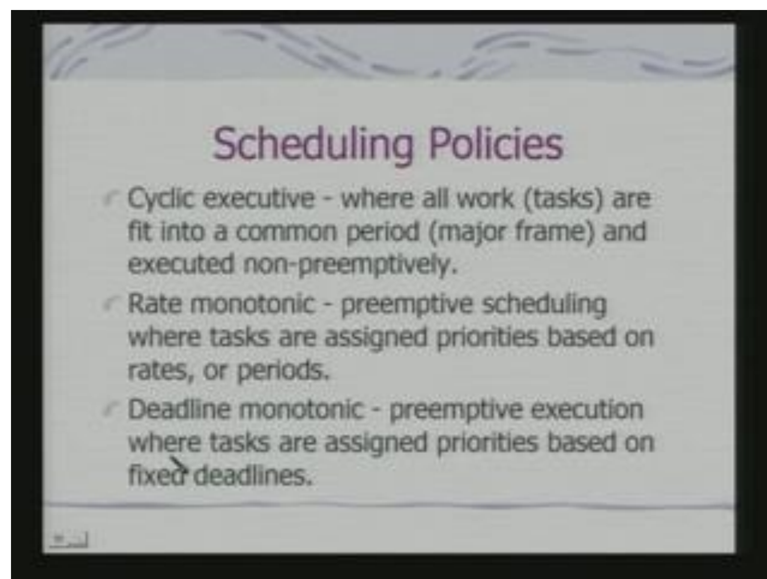
Scheduling Real-time Tasks

- Schedulability is the ability of tasks to meet all hard deadlines
- Latency is the worst-case system response time to events
- Stability in overload means the system meets critical deadlines even if all deadlines cannot be met

Now, when you schedule a real-time task and important issue becomes schedule ability. It is the ability of the task to meet deadlines or not. Because, it is not that all task can be admitted to meet the deadlines. Latency is the worst case system time to events. And stability in overload means the system meets critical deadlines even if all deadlines cannot be met.

So, there is a distinction and you meet those deadlines even when load on the system is high. Now, these are the parameters by which you can actually evaluate the scheduling policies. And schedulability is an issue at on the basis of which, you will decide add on OS decides whether to admit a task or not.

(Refer Slide Time 50:33)

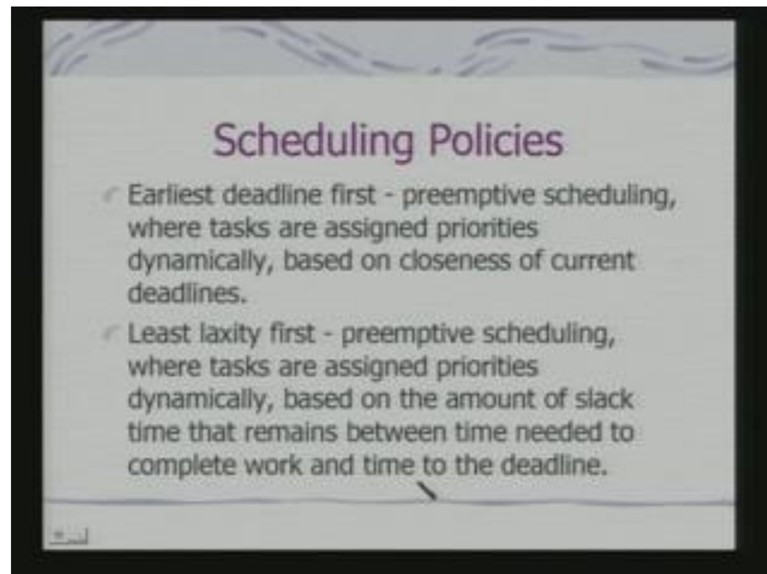


Whatever different scheduling policies? We shall look at these policies in more detail in subsequent classes; we are just introducing the policies today. Cyclic executive, where all work tasks are fit into a common period and executed in a non preemptive fashion if there are multiple periodic task. If you can compute LCM of this periodic tasks find that period. And then, schedule all these task together in that period. So that execute in a non preemptive fashion we get a cyclic executive policy.

Rate monotonic is a preemptive scheduling by tasks is assigned priority based on rates or periods. In many cases preemptive can actually mean that when a task arrives with a higher priority, I may interrupt currently executing task even when the time slice has not expired. Deadline monotonic is another preemptive execution, where tasks are assigned

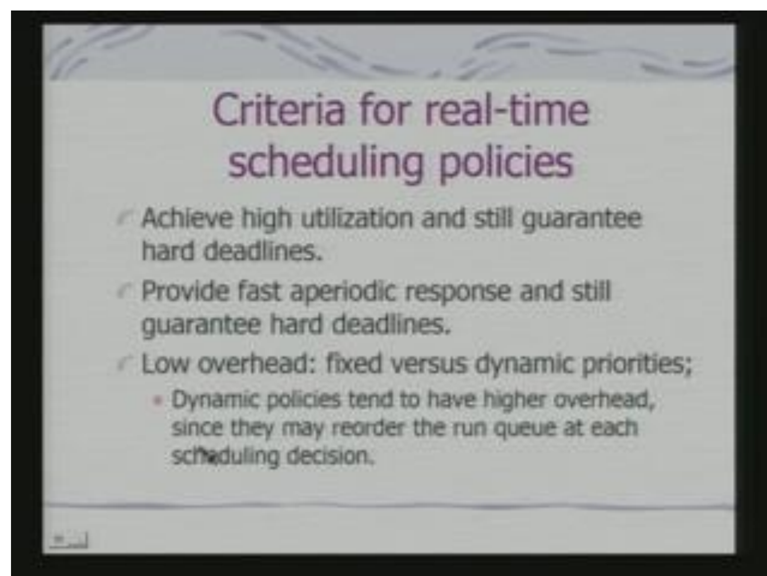
priorities based on fixed deadlines. In fact, variation of these is Earliest deadline first or EDF.

(Refer Slide Time 51:35)



Preemptive scheduling where tasks are assigned priorities dynamically based on closeness of current deadlines. Related to this you have got least laxity first it is again a preemptive scheduling, where tasks are assigned priorities dynamically. Based on the amount of slack time that remains between time needed to complete work and time to the deadline depending on that gap you decide which task to be scheduled.

(Refer Slide Time 52:10)



And all these scheduling policies, because the issue comes up is which scheduling policy it will select. So, you would like to have high utilization and still guarantee hard deadlines. Now, there can be a conflict on these criteria, because if you are trying to meet the deadline you may not able to admit jobs to increase your CPU utilization. CPU may be waiting for a while, so that you can meet the deadlines of periodic task or even for a periodic task with deadlines.

So, the other objective if comes in is provide fast a periodic response and still guarantee hard deadlines. This is the most difficult thing, because you have might of schedule the task. Now, suddenly and a periodic task arrives and you need to give a first response time. Then, there should be low overhead because you scheduling should not consume lots of time, because it is a fundamental problem. Because, if the scheduling itself consumes time.

Then, you will be missing deadline because of you scheduling policy. So, you cannot have a very complex scheduling mechanism. So, what we say the dynamic policies tend to have higher overhead and since they may reorder the run queue at each scheduling decision. So, each time a process arrives and you need to reschedule the processors. The moment you are rescheduling the processors, what happens, there is a time spent.

Because, you have to reorder the queue reordering the queue at least means manipulation of the data structure. Because, in that queue you have kept your PCB's, so that data structure has to be manipulated. So that introduces overhead, you also have an overhead, because of calculation of the policy that you need to do. So, these give us a very basic introduction to OS, OS targeted for Embedded Systems.

And, what we should understand is that, this scheduling in terms of the real time constraints is the key problem. And, what we have looked at? We have looked at the basic policies. We need to analyze these policies to understand this implications and this analysis we shall do in the next class.

Any questions,

Student: ((Refer Time: 55:01)) sir if that any process cannot meet the deadline can you list the problem.

So, the question is whether if that a process does not meet a deadline, what shall I do with the process. See if you remember we talked about schedulability. Now, when we are admitting a process a OS can check for it schedulability. If it can be scheduled, then only the process should be admitted. Otherwise I need not admit that process; that means that process does not join the ready queue.

The process may be started, the OS checks for its schedulability. If it cannot be schedule then it need not join the ready queue. So, it will be in a kind of a suspended state find any other questions.