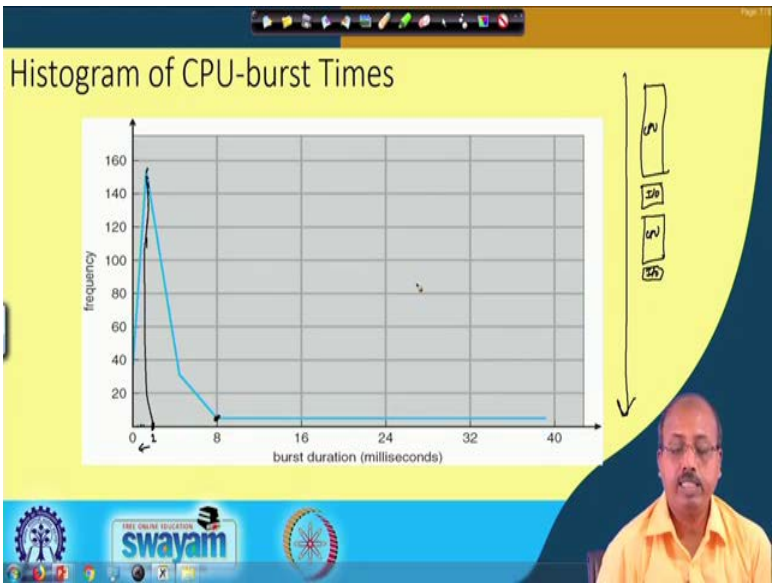
Operating System Fundamentals
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 24
Scheduling

(Refer Slide Time: 00:28)



So, next we will be looking into a diagram like as I was telling in the last class that this CPU burst size is a concerned. So, if you do and say do a statistical study about the sizes of this is CPU burst, and try to see like how many CPU bursts are of what size. So, in this diagram you can see that this if I am in the x axis. So, we have got this burst durations 0 8 16 24 32 etcetera.

So, burst durations are increasing and then on the y axis we have got the frequency like; in a typical program execution how many occurrences of such burst sizes are there. So, you basically what I mean is that, may be the first burst size is very large the second burst size there after that we have got some IO. So, this is an IO then the next burst size maybe of this size ok.

So, these are all CPU burst and then this is another IO maybe this is some IO here. So, if you take a typical execution of a process, then if we try to see like how many CPU burst we got of a particular size. Then in this diagram you see that the you get very few; you

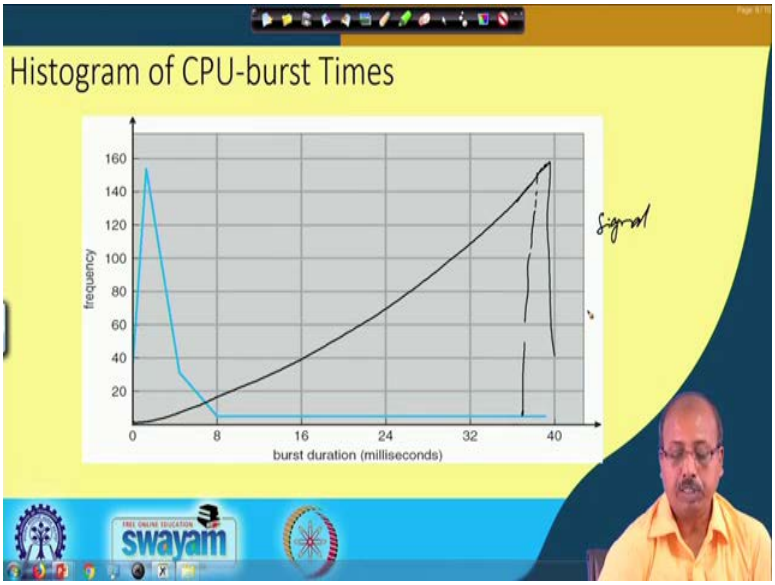
get very few CPU burst that are of larger size. Like here you see that after the burst size 8 this number of CPU burst. So, it has a become almost so, it is very small ok.

So, whether it is 0 or not that is a debatable issue, but that whether that will depend on the particular program, but. So, in general so, it becomes small. And there are a large number of bursts. So, which are of lesser size? So, if you see this is the peak. So, this is basically somewhere here so, which may be around say 2 or say 1.5 like that.

So, that way the number of CPU burst or small size so, that is very large whereas, so, below this again it is small because its if it is; so, number of CPU burst size of CPU burst CPU burst size we close to 0 so, that is also very less. So, this is, but there are some CPU bursts which as just slightly higher than 0. So, this means that this particularly happens when you have the program is just starting, then you have got maybe after initialization it goes into getting the input from the user and things like that.

So, that way this CPU burst size is very small and we have got the CPU burst here. So, after some there will be some CPU burst size for which the number of occurrences will be very high. So, here the number of occurrences very high of those burst and rest of the then it will start decreasing. So, this is the histogram showing the CPU burst time that you get typically get in a system. Of course, this situation may change for example, if you look into a program which is computation dominated.

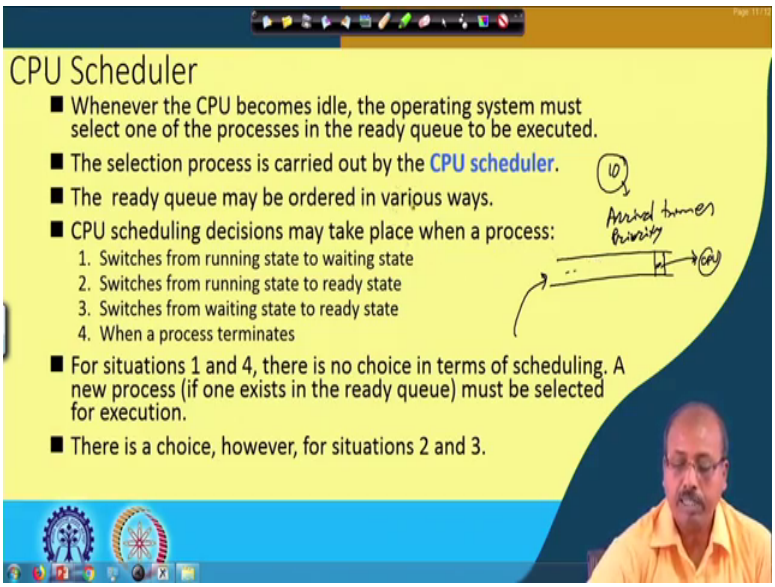
(Refer Slide Time: 03:34)



Then you will see that this part is the small, but there will be a some peak will be somewhere here maybe. So, that is you get number of CPU burst of size higher size to be more compared to the CPU burst of lower size or maybe this curve may shift further. So, you may get is a CPU burst behavior like this. So, as you are going so, this burst sizes are very high very you get some burst size which is very high and their frequencies more. So, this is particularly true when we have got signal processing type of applications because then that it will be doing lot of computation.

So, that way it is high. On the other hand this if you are looking into this IO bound jobs ok. So, it database operations and all. So, this is a typical situation that you get ok, but in a; so, because the number of CPU burst amount of CPU burst size is small for most of the burst. So, that way we have got this situation, but again for a typical program we will get a mix of the two. So, both IO bound and CPU bound things operations will be there. So, that way this burst size will be changing.

(Refer Slide Time: 04:51)



So, this is about the CPU burst now whenever we are talking about a CPU scheduling policy the whenever the CPU becomes idle the operating system must select one of the processes in the ready queue to be executed. Because if it is not then the CPU is will be waiting idle and that way the CPU utilization will drop. So, we do not want the CPU to sit idle.

So, naturally the CPU scheduling algorithm so, it should select the it will it should select one job from the from the ready queue and put it on to the memory put it on to the CPU. So, it should send it should give control of the CPU to the job so, that job is executed. So, it is the operating systems responsibility to select among the ready processes and give the CPU to one of the ready processes.

The selection this particular selection is carried out by the CPU scheduler. So, CPU scheduler will do this thing. The ready queue may be ordered in different ways. So, for example, you may say that I the ready queue I will ordered in terms of their arrival. So, when a particular job came. So, we so, there are say 10 jobs in the ready queue. So, the CPU scheduler has to pick up one out of this 10.

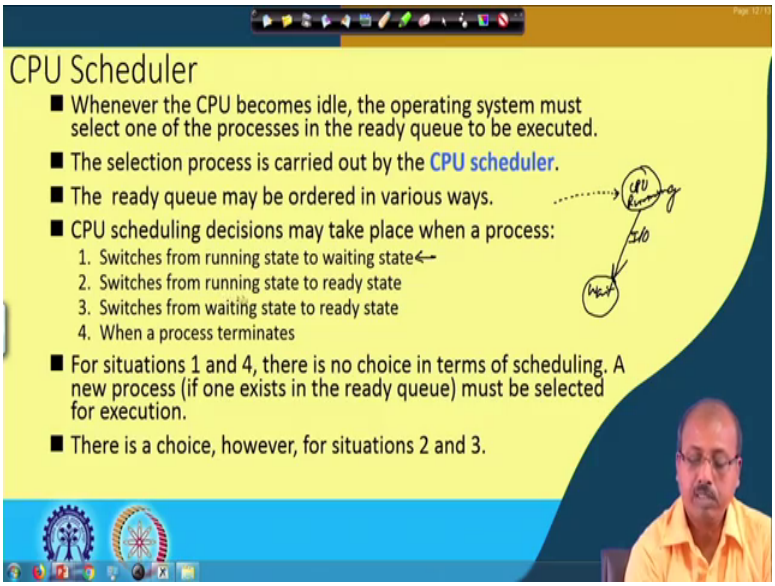
Now, if we keep the queue ordered in some fashion, then this designing the CPU scheduler becomes very simple. So, if the queue is ordered in terms of say their arrival times if they are ordered in terms of arrival times, then we know that the first job that I have in the system is the oldest job.

So, the scheduler can take this job and give it to the CPU for execution. Now or maybe they are sorted based on their priority. So, we have got priority of the jobs then. So, whenever a new jobs; whenever a new job joins into this queue. So, based on this priority so, this job will be placed somewhere in this queue. So, it may so, happen that this job is the highest priority one and as a result it comes to the beginning and by pushing back all other jobs.

So, in that also this is CPU scheduling policies. So, it can take up the first job from the queue and give it for give it a chance for execution by the CPU. So, this way the scheduling policy or scheduling algorithm will be made very simple, but the queue data structure will maintain in such a fashion that this scheduler can be implemented very easily.

So, this ready queue may be ordered in different ways that helps in running a developing the CPU scheduling algorithm and scheduling decisions may take place when a process switches from running state to waiting state, switches from running state to ready state switches from waiting state to ready state and when a process termination. So, these are the points at which we will do we may need to invoke a scheduler.

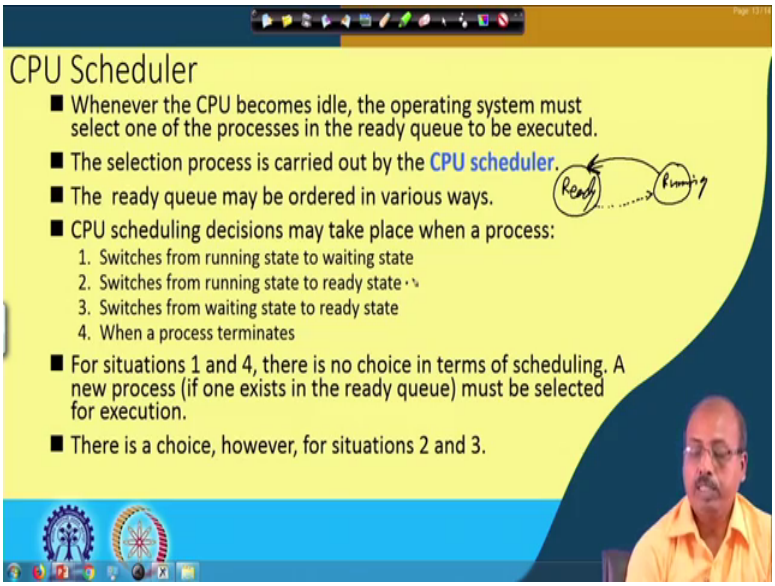
(Refer Slide Time: 08:01)



Like a process was in the running state so, it was a process was running in the CPU. So, it was in a running state now what happens is that, this process wants to go to some IO operations. So, it wants to do some IO operation. So, it cannot proceed further. So, it goes to the wait state and when this is going to wait state then naturally the next process to be selected from ready state and put on to this one.

So, the scheduler needs to be invoked at this point of time. So, that is the first point. So, when a process is running and it switches from running sate to waiting state, the scheduler needs to be invoked or when a process switches from running state to ready state so, it may so, happen that instead of going from running to wait.

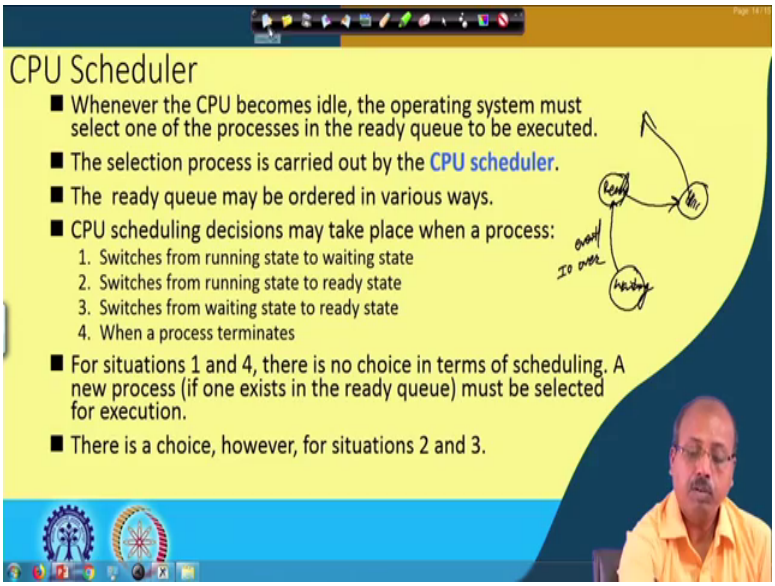
(Refer Slide Time: 08:47)



So, maybe it was running here and due to some reason or the other. So, this is this it is put back onto the ready state. And this is particularly true if you have got a time shared system in which a job is given CPU for sometime quantum only and after the time quantum expires. So, this job is taken back from the CPU and it is a CPU is given to some other job.

So, that way this job the process that was running here so, it is taken back and it goes to the ready state. So, in that case the scheduler has to select a job from the ready state and give it a chance for execution by the CPU. So, that way that job makes a transition from ready sate to running state. So, that is why that is a second point, that is when it process switches from running state to ready state then also we may need to invoke the scheduler.

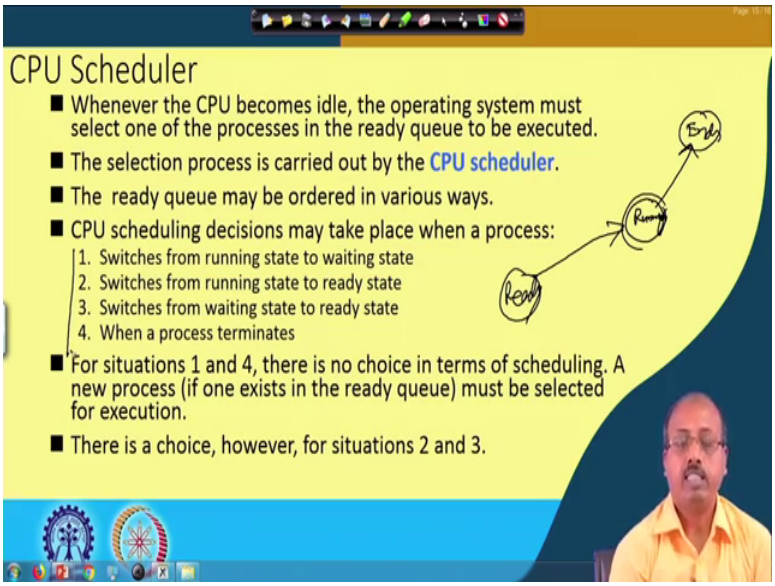
(Refer Slide Time: 09:45)



Another thing that we have is a process is was in a waiting state and after some time this wait period is over. So, wait period is over maybe because of several reasons maybe it was waiting for some IO operations. So, that IO operation is over or it was waiting for some event to occur for example, it was waiting for the some child process to terminate things like that.

So, when this is happening then this process when this event is over event or IO is over, then it makes a transition to the ready state and now it may so, happen that this job that has come to the ready state is the highest priority one. And it is even it is a priority even higher than the currently running process. So, in that case this process has to be taken out of CPU and this process should get a chance for execution. So, as a result this whenever a process switches from waiting state to ready state so, there also we may need to take some scheduling decision.

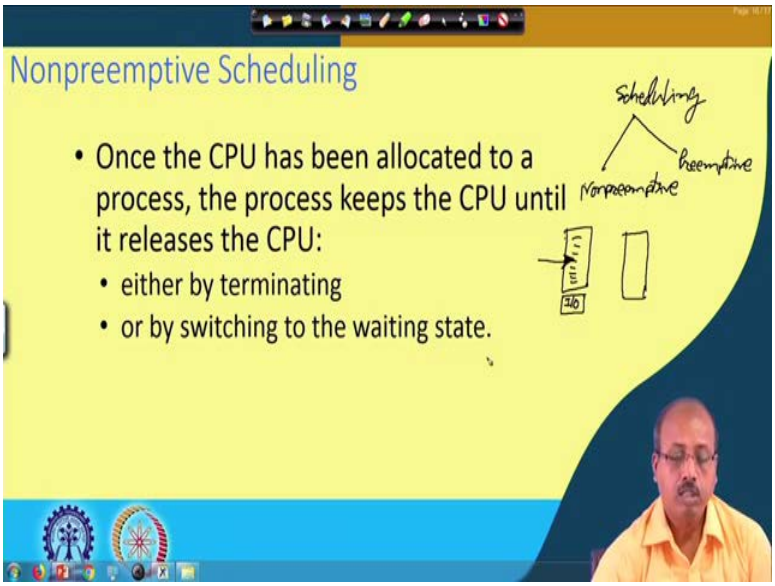
(Refer Slide Time: 10:50)



And finally, when a process terminates; so, a process was running and it has finished its execution by executing all the statements. So, it terminates so, it ends or terminates. So, again the CPU becomes free, now I have to take some job and put it on to the CPU. So, that is the some job from the ready state should go to the running state. So, when a process terminates. So, these are the four different situation or four different cases into which this scheduler has to come into picture and take appropriate action to do the scheduling operation.

So, in case of situations 1 and 4 there is no choice in terms of scheduling. So, 1 and 4 with you do not have a any choice a new process must be selected for execution and there is a choice; however, for situations 2 and 3. Like switches from running state to ready state or switches waiting state to ready state. So, we have got a choice.

(Refer Slide Time: 11:56)



So, how this scheduling algorithms can be developed so, that will be that their decision will take. But before coming to this, this scheduling can be thought about in of two different types one is called preemptive scheduling another is called non preemptive scheduling.

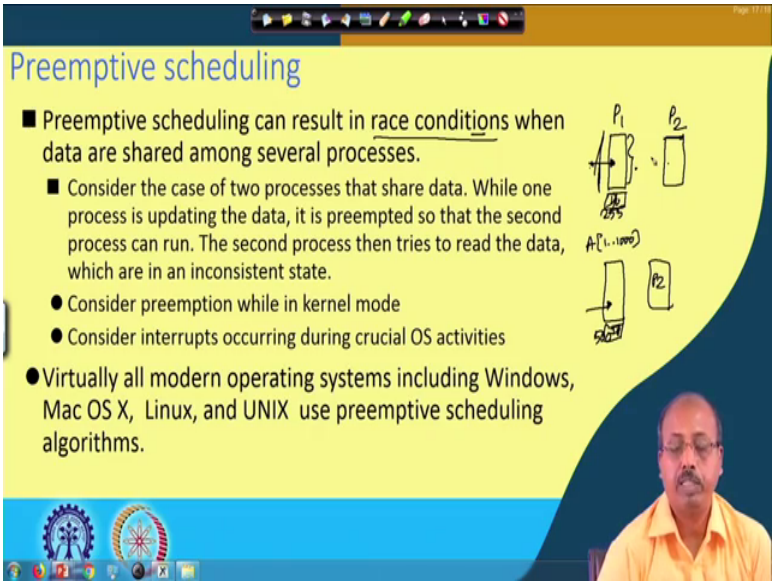
So, if you look into this all the scheduling policies. So, you can the think about two classes one class is called non-preemptive and the other class is preemptive. Non preemptive class means that the CPU that is once as process has got the CPU. So, it will be running a it will be utilizing it completely. So, a non-preemptive class once the CPU has been allocated to a process the process gives the CPU until it releases the CPU.

So, how can it release the CPU? As I said that if this is the code that the process is now executing. So, all these instructions that are there; so, they are some computational instruction. Now after this it maybe it is going into some IO corporation ok. So, if it is going an IO operation then the CPU can be released ok. So, that is a one voluntary way of releasing the CPU. Other thing is that it is it was executing and the execution is over.

So, thus this process has got no more statement to be executed. So, it terminates. So, in of these two cases so, this process will be releasing the CPU, but there is no other way by which the CPU can be taken from the process. So, you cannot stop the CPU you cannot stop the process at some intermediary point and take the CPU back from the process. So, you cannot stop the process in between. So, that is the non preemptive scheduling policy.

So, CPU is not preempted from the process until and unless is it voluntarily relinquishes the control.

(Refer Slide Time: 13:59)



So, that is the non-preemptive scheduling policy, for other category we have got preemptive scheduling policy. So, in the preemptive scheduling, it can. So, this means that you can take the CPU back from the process in between. So, it can create some conditions like something called a race condition that it can create. So, we will see how it is. So, what can happen is so, this is a situation that there are two processes that share data while one process is updating data it is preempted so, that the second process can run. Second process then tries to read the data which are in an inconsistent state.

So, this is one problem. So, this can create some race condition because suppose I have got two processes P 1 and P 2 now if like if this P 1 has got some piece of code to be executed and if I say that it will be taken out from here and then P 2 gets a chance for execution. So, maybe P 1 was updating an array A in this part of the codes. So, it was updating the array A which is 1 to 1000 there are 1000 enters in the array so, which it was updating.

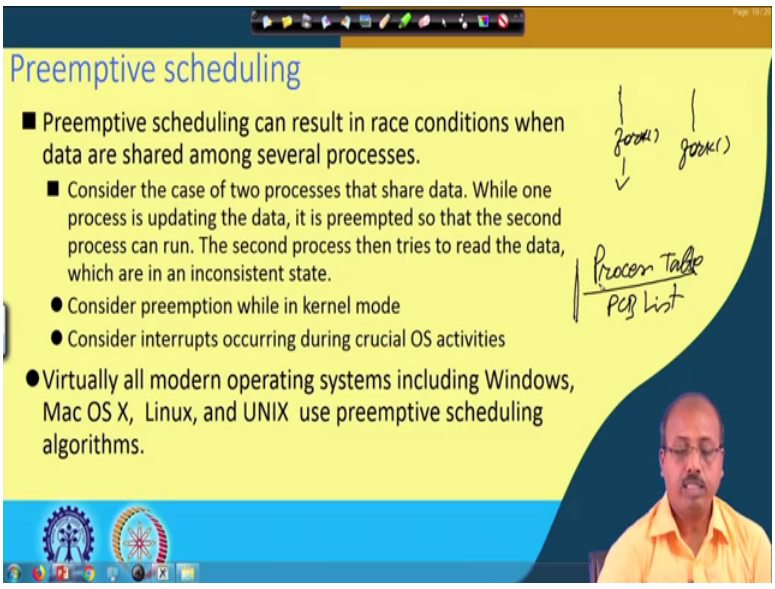
And this after it has updated for say 255 elements then at this point it is decided that the CPU will be taken back from P 1. Now if P 2 is scheduled then what will happen is that and P 2 it tries to read data for the array A then for the remaining values for 256 onwards 256 to 1000. So, it will get inconsistent values the old values that it will get which is not

correct. So, that way there may be difficultly so, for which process gets chance earlier which process gets preempts the other process and at what point.

So, in one case for the same piece of code maybe the preemption is done at this point, in another run for the same piece of code maybe preemption is done somewhere here may be after doing this updation after doing the after updating say 500 elements, then this preemption occurs. Then naturally outcome of this situation when P 1 has executed up to this much, then P 2 is executed and in this case if P 1 executed up to this much then P 2 is executed. So, these two cases the outcome will be different and it is. So, there that is the race condition. So, two processes they are racing between each others so, that it can get access to the CPU.

But, if it does that then because of this preemptive policy this as happened; so, if it was non preemptive then in either case P 1 will complete its the total updation, then possibly it will go into some IO operation by releasing the CPU. So, here also it will go into an IO operation releasing the CPU and then only P 2 will get a chance that was the situation for non-preemptive scheduling. But for preemptive scheduling we have got a different situation also we have to consider preemption while in kernel mode. So, this is another very important thing because in kernel mode of operation. So, it is actually doing some critical operation.

(Refer Slide Time: 17:31)



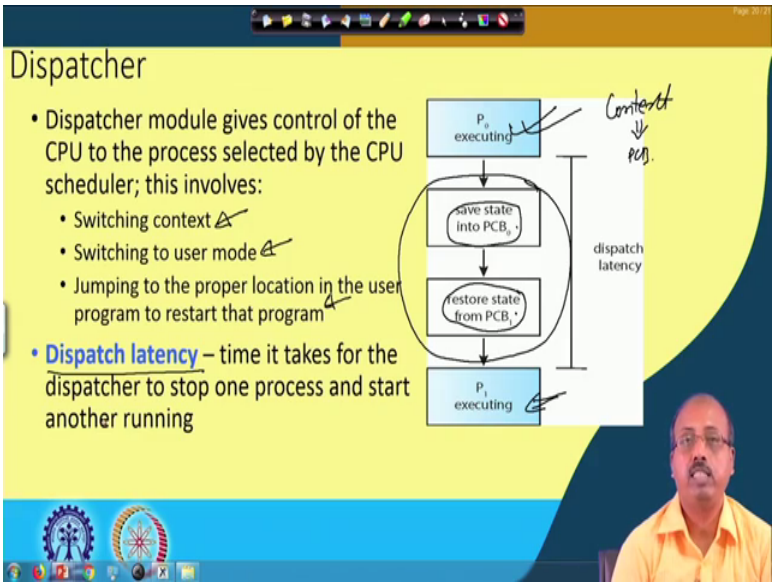
So, maybe for example, suppose a processes has given call to the fork statement and as a result of fork. So, it goes into the kernel mode of execution and there it is updating the process table a new process is getting created. So, it is updating the process table PCB list and all that is; so, all those important things it is updating. Now in between the updation if this process is preempted if this process is preempted in between then and another process gets a chance for execution and that also does a fork operation.

Then what will happen is that, this process table may go to an inconsistent state because we have got more than one process which are concurrently trying to modify the process table. So, that is a situation that has to be avoided. So, this preemptive scheduling if it was not there, then of course, my kernel power in the kernel I cannot have two concurrent executions of two processes.

So, this non preemptive kernel will help us in that way, but definitely it has got a some other difficulties that is why we do not make kernel non preemptive. But preemptive kernel basically that helps because you can respond to many other important operations and so, this is the problem. So, we have got this race condition that can affect the system performance and also interrupts occurring during crucial race activities; like it was for example, here suppose it was updating the process table and then an interrupt came.

And then if it goes into that interrupt handling, then there again that interrupt routine it may try to modify the process table. So, that can happen so, that will lead to some difficulty. So, we have got this preemptive scheduling is fine. So, it is good that you can take back the CPU from a process and give it to some other needy process that is good, but it creates lot of inconsistencies it has the potential to create lot of inconsistencies. Virtually all modern operating systems like there are windows Mac OS X Linux and UNIX they use preemptive scheduling algorithms because of the were reason that otherwise the system throughput or you become slow and you cannot be fear to a number of users number of user processes. So, that is why it is there all of them are preemptive

(Refer Slide Time: 20:02)



Next we will be talking about the dispatcher or the schedule or the scheduler. So, this dispatcher; so, this is the module that gives control of CPU to the process selected by the CPU scheduling. So, what happens is that supposed at present the process P 0 is executing at present P 0 is executing and due to some reason or the other. So, P 0 cannot proceed further.

So, maybe it is because of this IO event wait or maybe because of some. So, time slice expire whatever it is suppose it cannot proceed further. So, in the and the scheduler takes a decision that now P 1 should run ok. Now this switch over from P 0 to P 1 is not that simple because later on you would like to restart P 0 from the point at which its stopped. So, that way we have to save enough information and so, we have to save the context of the process we have to save the context of the process P 0 into somewhere in the memory and we have to know that the context is save in the process control block of the process.

So, first thing that it has to do is the save state into PCB 0 and this process 1 when it is the executing may be the process 1 executed to some extent previously also. So, it is also a half done process. So, you are trying to resume the execution of process 1 from the point at which it got suspended. So, that way again the PCB of process 1 it has got the information about the point up to which it got executed. So, from there it has to restore the state from PCB 1.

So, this thing has to be done. So, this current context has to be saved in PCB 0 and the context of PCB 1 content of PCB 1 should be copied and should be taken and copied on to CPU registers and all. So, that is the restoration of context from PCB 1. So, this is the additional things that we need to do whenever we are switching the control from process P 0 to process P 1. So, this is done by the dispatcher and this particular delay due to this copy and restore of this PCB is. So, that is known as the dispatch latency.

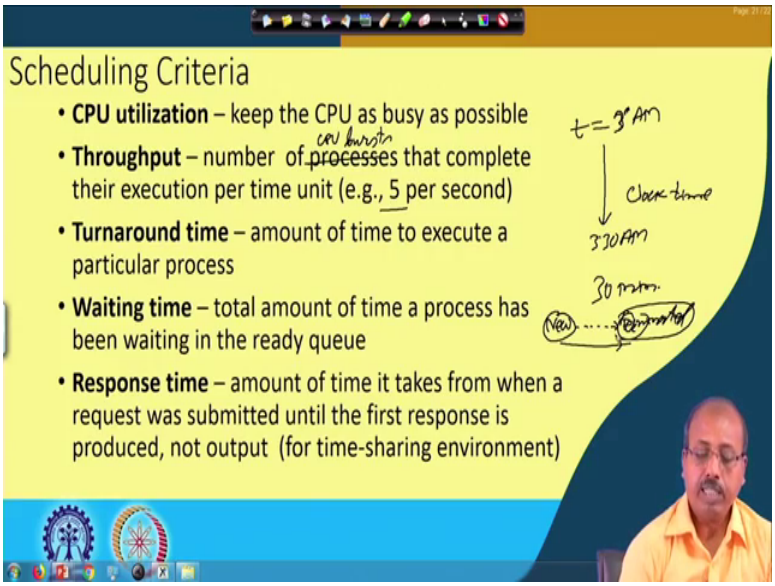
So, definitely I would like to make this dispatch latency is small and there are different processor architectures that come up to reduce this dispatch latency; for example, in different processors you may find that we have got a duplicate set of registers. So, that you do not need to save all the register values in the PCB or so, you can just note down the register bank which is allocated to this process so, that way you can just see if it is if you is the next use a different register bank then you do not need to save the context of previous process into the PCB.

So, that way we can think about different architectural support for this reducing this dispatch latency. But whatever you do so, this some small amount of time will always come into this switching of process switching from one process to another. So, this switching context is important this switching to user mode and jumping to the proper location in the user program to restart that program. So, these are the so, switching context that is copying the content from PCB to CPU, CPU register or for CPU register to PCB.

So, this has to be done in the supervisor mode because the user program should not be allowed to do this thing otherwise the system may go into an inconsistent state. So, this way we have to do this switching of context in the supervisor mode, then it has to go to the user mode and then we can start executing from that particular user particular programs point at which it was suspended. So, we can restart the problem.

So, the dispatch latency it is the time with the time taken for the dispatcher to stop one process and start another one. So, that is the dispatch latency and this dispatch latency should be small. So, that is the idea. So, how can I make this dispatch latency small and all? So, that is an issues that we will see different architecture so, they are supporting different facilities by which it can be done.

(Refer Slide Time: 24:30)



Next thing that will be looking into is what can we consider like how do you judge a particular scheduling algorithm whether it is good or bad. So, there are several criteria that we can have for this a CPU scheduling.

So, first and for foremost criteria is known as the CPU utilization; so, CPU utilization says that how much time has been kept busy ok. So, ideally as I said that we want that CPU should be utilized 100 percent, but again because of this reason that in between we have to save and restore these PCB and all. So, this with these times cannot be reading its ok. So, some amount of time will always be spent. So, you cannot have this utilization 100 percent, but if it is not 100 percent you would like to be very close to 100 percent as I as close as possible.

So, CPU utilization it is a measure like keep the CPU busy as much as possible ok. So, how much times CPU is busy? So, CPU if there is no job to be done by the CPU then naturally CPU will remain idle and that time will be in the wastage; then the throughput number of processes that complete their execution per unit time. For example, I can say that there are 5 jobs completed per unit time. So, the throughput is throughput is say 5 per second. So, per second there may be 5 jobs completed.

So, throughput is 5 percent again what you would like to do is the throughput should be high now there is a question here. So, see this throughput can be made high if the number the jobs that are executing are of smaller size then if there if the CPU bursts are small,

then definitely I can complete a number of CPU burst within a given time. So, the utilization throughput will be high. But at the same time if you have got this burst sizes very small then there will be a lot of time that will be wasted in the context switching.

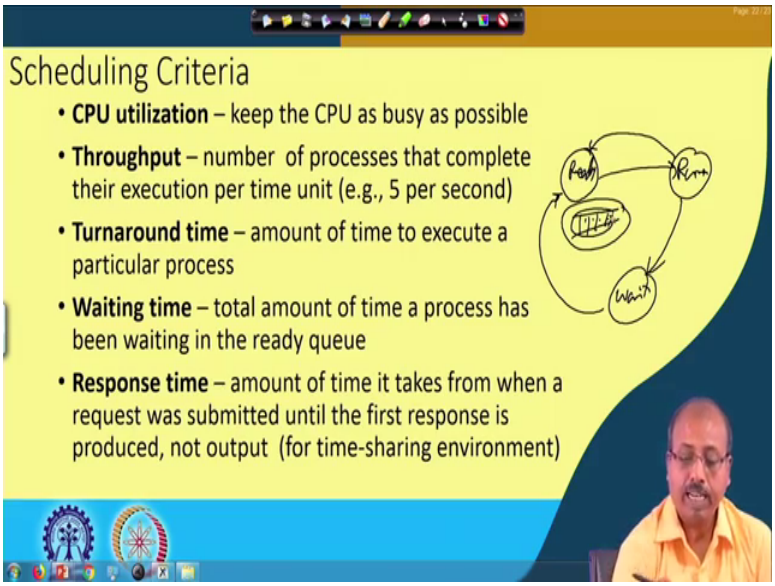
So, context switching will also take time. So, naturally throughput will not be exactly a proportional to the number of jobs that are completed, but it will be also depend on this extra context switching time that is needed. So, throughput is the number of processes that complete per unit time. So, instead of number of processes so you can say it is the number of number of CPU burst that are completed per unit time.

So, when you will be using this terms interchangeably, that is the CPU burst and processes. So, you should always keep in mind that we are actually talking about the CPU burst. Then turnaround time it is the amount of time to execute a particular process. So, turnaround time is suppose we submit a job at time t is equal to say a say 3 o' clock say at time t equal to 3 we submitted a job and when this job is complete. So, that is the two turnaround time. So, this is basically the clock time.

So, you submitted the job at say 3 AM and you got the result at say 3 30AM. So, in that case the turnaround time is 30 minutes. So, that way how much is the turnaround time? So, that the time at which a job is submitted. So, a job makes a transition from this created to. So, when the when the job gets the status new, from there the time it gets the status terminated are over ok. So, till that states of how much is the total time needed. So, there in between the job will be scheduled and descheduled several times depending on the policy that we have.

So, that way we can have this new. So, this entire time will come in the turnaround time. So, we definitely want this turnaround time to be small because we want that more jobs be done to use other users of the system they should feel that my job is not taking much time for execution so, that is the turnaround time. Then waiting time; so, total amount of time a process has been waiting in the ready queue. So, as you know that we have got this thing the processes. So, they are put on to the ready queue for a execution and they are waiting in the ready queue.

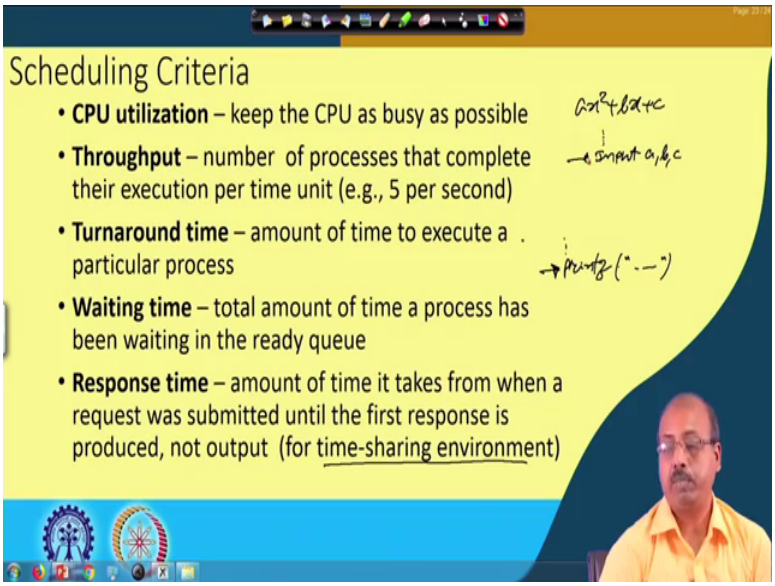
(Refer Slide Time: 29:16)



So, in they are all in the ready state and all this processes are put on to the ready queue. Now, from here so, we are taking processes to go to the running state and after some time the process may be coming back to ready queue directly or it may be going to some wait state and from the wait it joins back to the ready queue. So, both of them are possible, but ultimately that in the if the job is not complete in either way either via this path or via this wait state path, it joins the ready queue again. So, again it has to wait for sometime in the ready queue to get the next chance of execution. So, next CPU burst will be coming when other jobs are over or something like that depending on the scheduling policy.

So, the amount of time that is jobs spends in the ready queue. So, that is the total amount of time a process a waits in the ready queue. So, that will be called the waiting time of the job and naturally we would like to reduce this waiting time. And we have got another timing measure which is called the response time. So, response time is basically it is particularly important if you have got this time sharing system or multi user or interactive system.

(Refer Slide Time: 30:29)



So, basically if I take the same example like finding the roots of quadratic equation ax square plus bx plus c. So, to start with the program does some initialization and then it gets the input, it executes the input abc. Many programs they start with they at the beginning it prints some message and with that it starts. So, there is a some message to be printed and with that the program starts. So, suppose I start the I give the program for execution then how much time it takes to come to this first print statement or this first input statement where some user interaction is involved.

So, if it print something the user is able to see my program it is running and it has executed up to this point. So, it is giving the message or if it is an input statement is there. So, it is asking for input from the user. So, user is getting the feeling that the program is now asking for some input so; that means, my program is progressing my process is progressing. So, that is the response the time taken by the request that was submitted until the first response is produced it is not output. So, it is basically if not the output of the program like it has not given me the roots, but it is just given me the message that input the values of a b and c. So, and that is the response time and we get a feeling like how responsive is the system.

So, how quickly the program is responding to the users requests. So, that type of situation is a time sharing environment and in the time sharing environment we have got this response time as an important parameter. So, depending upon the environment on to

which you are judging a scheduling policy. So, one of the criteria may be important. So, for maybe for interactive systems this response time is important whereas, we are not that much important with not that much bothered about response time. So, if you are doing say a scientific calculations. So, that way we may be bothered there about this throughput or that is number of jobs completed per unit time. So, it is doing lot of number crunching. So, it is doing there.

So, we will be looking into many scheduling policies and we will try to judge them based on the scheduling criteria and that we will do in the next class.