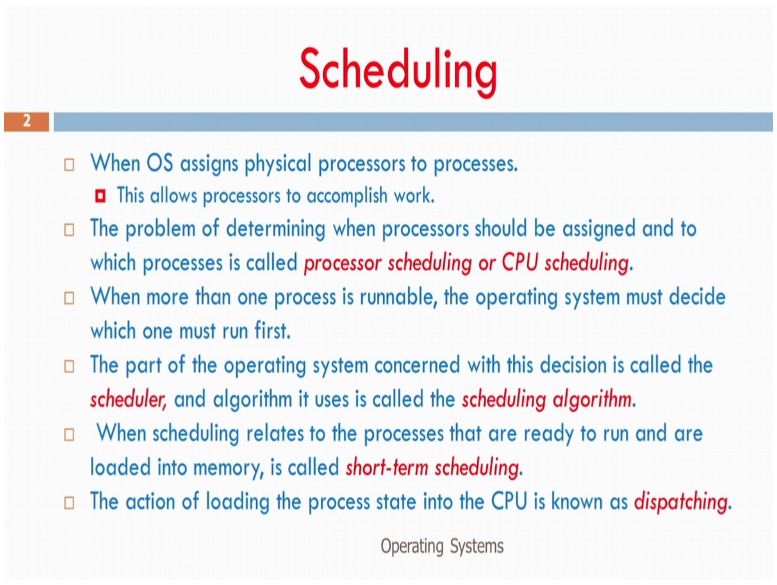
Information Security-3
Prof. V Kamakoti
Department of Computer science and Engineering
Indian Institute of Technology Madras
Basics of Unix and Network Administration
Operating Systems Introduction
Mod01 Lecture 06
Module 6: PROCESS Scheduling

So welcome to module 6 and we are looking at process scheduling. Now what do you mean by scheduling? So I am scheduling my classes tomorrow, I am scheduling a meeting tomorrow that means what we say that is you assign somebody wants a resource and you assign (())(0:32) timing at which you assign that particular element to that resource, right. I schedule a class means students want to learn hopefully. Now so I say at 9 a clock there is a class. So I schedule that class means that students need to come at a 9 a clock to attempt to that class. So I am assigning the student at to a classroom, which is a resource at a particular time, right. So this is what we mean by scheduling, right. This (()))(1:01) is a the very simple example of scheduling.

(Refer Slide Time: 1:36)


Scheduling

2

- When OS assigns physical processors to processes.
  - This allows processors to accomplish work.
- The problem of determining when processors should be assigned and to which processes is called *processor scheduling or CPU scheduling*.
- When more than one process is runnable, the operating system must decide which one must run first.
- The part of the operating system concerned with this decision is called the *scheduler*, and algorithm it uses is called the *scheduling algorithm*.
- When scheduling relates to the processes that are ready to run and are loaded into memory, is called *short-term scheduling*.
- The action of loading the process state into the CPU is known as *dispatching*.

Operating Systems

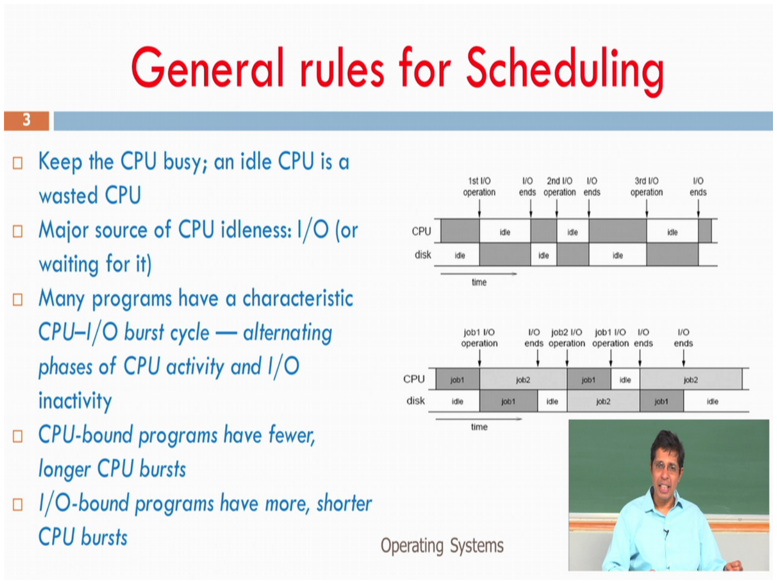
Now let us look at what do you mean by process scheduling? process scheduling here is that I want to assign a process to the CPU at some time that is why we call it as process scheduling, scheduling what scheduling process to whom to the CPU, right. So I want to schedule a process to a CPU at a given time. So this is why we call it as process scheduling, okay. Now we can also call it as CPU scheduling. Now what is the challenge here? Now, so already I told that the there are several fellows waiting several processes waiting. This can be applicable to threads also, there are several process we will say, this now processes, which covers also threads. There are lot of processes waiting in a ready queue. There is only one CPU.

Now the operating system should go and pick one process from these people waiting there. These processes waiting there and assign it to the CPU get it executed then it has to get another, right and this is the challenge. Now what is the challenge here, I want to see that all the processes get executed nobody should feel that they are waiting, right. You are going to a server; let us say you are going to a mail server. The scheduler in the mail server should see if the moment you click (())(2:34) or mail should come if you will take half an hour for that mail to come you get really frustrated, right these user level experience. So it is better to understand the role of a scheduler. The scheduler, there are millions of users trying to access to save their own mails.

Let us about let us talk take the example of a mail server. The millions of users who want to actually go and read their mails, everybody wants to look at it faster. Now there are that means every time somebody logs in a process is created. So there are millions of processes waiting on the scheduler queue and I do not have millions of CPUs, I have limited number of CPUs. Now there is algorithm, which will take one fellow, each process assign it to a CPU and then remove it back etcetera and it should be done in such a way that the user never feels that there is a delay and that is the challenge, right. So that is the entire aspect of scheduling.

Now the scheduling part of the operating system has two components, one is call the scheduler another is call the dispatcher. The scheduler looks at the queue and finds out which process I need to schedule. Once it identifies that process, the dispatcher will take that process and put it into the CPU, right. So these two are two important components of a scheduler.

(Refer Slide Time: 3:58)



General rules for Scheduling

3

- Keep the CPU busy; an idle CPU is a wasted CPU
- Major source of CPU idleness: I/O (or waiting for it)
- Many programs have a characteristic *CPU–I/O burst cycle — alternating phases of CPU activity and I/O inactivity*
- *CPU-bound programs have fewer, longer CPU bursts*
- *I/O-bound programs have more, shorter CPU bursts*

Operating Systems

1st I/O operation
I/O ends
2nd I/O operation
I/O ends
3rd I/O operation
I/O ends
CPU
idle
idle
idle
disk
idle
idle
idle
time
job1 I/O operation
I/O ends
job2 I/O operation
job1 I/O operation
I/O ends
I/O ends
CPU
job1
job2
job1
idle
job2
disk
idle
job1
idle
job2
job1
idle
time

Now why does scheduling work? Now when we execute a program, a program is a collection of CPU and IO, you even take the simple hallo world right. There is a printf hallo world. So you assign something to a string say, hallo world to a string and print that string. So there is a some computation involved where you actually assign that hallo world to a string and then immediately, there is an IO where it is going to write on your console the hallo world. So (())(4:27) even the simplest of the simplest program that you have written has a small bit of computation and the small bit of IO.

So if you look at programs in general, this is an empirical study I can write a program, which does not do any IO at all mainly, it may not make any sense, but the majority or 99.999 percent of the programs do have lot of CPU activities and lot of IO activities. Now IO you know is a real IO in the sense that it takes lot more time than the CPU, right. It is much slower than the CPU. So if I do say, 10 IO operations at that time I could do 1000s of CPU operations that is the difference. So the IO is much slower than the CPU.

So now let us talk about the mail server again I have 100s of processes. Now each process will have its CPU activity followed by an IO activity again a CPU activity followed by an IO activity etcetera. So this alternating sequence of CPU activity IO activity CPU activity IO activity, this call this is how a program executes and each CPU activity is called a CPU burst and then IO burst then CPU burst then IO burst. The CPU burst the IO burst need not be essentially equal the IO bus will take much more time than the CPU burst for reasons that I have basically talked.

Now when an IO is happening, the CPU will not do anything, because the IO is happening between the IO device and the memory. Mostly the notion of a direct memory access or DMA will be used and so the IO will never involve the CPU. So let us take the top diagram on your right hand side. Now what happens is we have only one process that process does some CPU activity then it does the first IO operation. When the first IO operation is happening on the disc, the CPU remains idle and then while the first CPU operation was happening on the CPU the disc was idle.

So the CPU is doing something disc is idle then the CPU becomes idle then the disc starts working for the first IO operation, when the first IO ends again CPU starts working till the second IO operation and in that period between the IO ends and the second IO operation your disc is idle and again now, again disc start a second IO operation and till it ends the CPU is idle and now again CPU starts operation and till the third IO operation, your disc is idle and when the third IO operation starts your again your CPU is idle till it ends and so on so for.

So the disc and the CPU are not, so the question is when the CPU is idle why cannot it do the with the CPU activity of a second process and that is what you see in the this next diagram that is below on your right hand side. So when the CPU does the first job of course disc is idle, the first job ask for an IO operation disc starts doing the IO operation of job one at that time the CPU can do the job 2, the CPU burst of job two and then in this context I just put the IO is much slower than the CPU or IO is faster than the CPU.

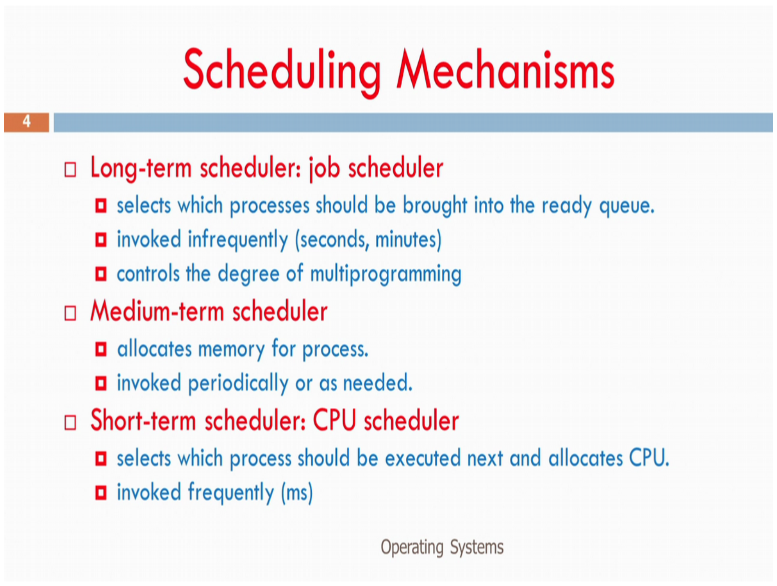
Now when the CPU uh when this job 2 wants to do an IO operation, it goes to the disc at that time the job 1 can be rescheduled on to the CPU, again when job 2 finishes job 2s IO finishes, job 2 can be rescheduled on to the CPU while job 1s IO can be done, right. So now you note that there is a overlap of the IO and the computation here. So almost at the same time 2 jobs can be executed almost concurrently and every job feels that they are given sometime, right and there IO is also happening here, right. So this particular property that any program that is executing is an alternating sequence of CPU burst and IO burst and that when an IO burst is happening, the CPU is idle with respect to that program and when the CPU burst is happening the IO is the disc is idle or IO peripherals are idle and this property and this feature is basically exploited to start executing multiple programs at the same time.

Number of processes that are executed at a single point of time the maximum number of processes that could be executed that is actually called the degree of multi programming. The degree of multi-programing is means is equal to the maximum number of processes that are simultaneously being executed being admitted into the system, they can be in of these state running state or suspended state or waiting state. Running if I have say 4 CPUs, I could have at most four processes that are running lot of things can be in waiting state, lot of them can be in ready state put sigma of all this numbers and that is what we call as degree of multi-programing, okay.

Now there are two types of jobs that we see, one is as CPU bound jobs, for example matrix multiplication is a CPU bound job why because I have n cube multiplication, while only end squared order n cube multiplications, right. If I want to multiply 2n by n matrix, but I have only n squared IO, I have to read you know two matrixes and size of each matrix is n square, right while matrix addition is an IO bound program, because I have only n square decision and I have n square IO operations and IO operations are much more costlier than compute operation.

So a simple example of what CPU bound program and IO bound program (())(10:21) is comes even from our first programing examples that we learn in life like matrix multiplication verses matrix addition. So in a IO both in the context of a CPU bond program and Io bound program you know, this scheduling is valid as a concept. Now you see job 1 is more CPU bound while job 2 is more sort of balance between it has more larger IO operation. So if I have a mix of CPU and IO bound jobs then this type of a scheduling really helps.

(Refer Slide Time: 10:59)



Scheduling Mechanisms

4

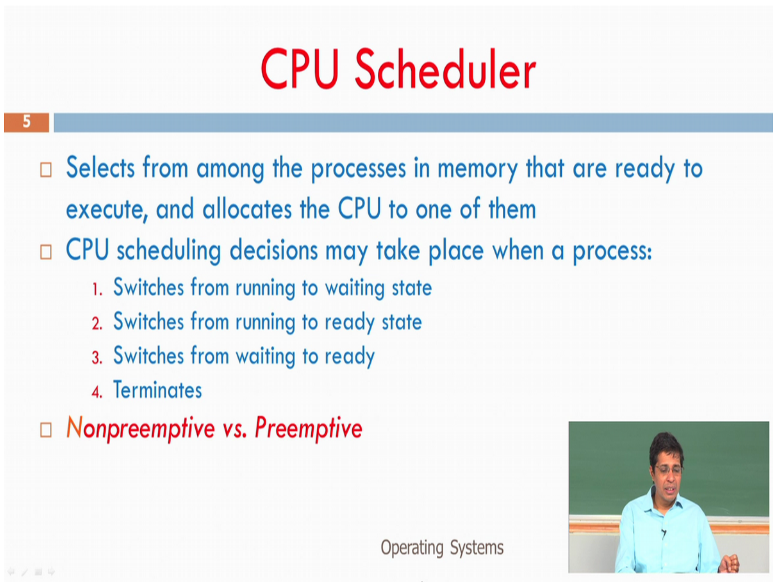
- Long-term scheduler: job scheduler
  - selects which processes should be brought into the ready queue.
  - invoked infrequently (seconds, minutes)
  - controls the degree of multiprogramming
- Medium-term scheduler
  - allocates memory for process.
  - invoked periodically or as needed.
- Short-term scheduler: CPU scheduler
  - selects which process should be executed next and allocates CPU.
  - invoked frequently (ms)

Operating Systems

So what are the scheduling mechanisms? There are three types of scheduling; one thing is a long term scheduler, which basically is done for batch jobs, right. So I submits some jobs to night and tomorrow morning I go and collect there is not interactive program, the still you do in your high performance computing system or super-computing, where you have large job to be executed large finite element say simulation you go (())(11:22) give night and the tomorrow morning you collect. So these type of schedulers are called long term, term in the sense, it is anything more than 10 minutes is a long term scheduler, okay where the program is going to be assign to the CPU for more than 10 minutes also, we can call it as long term scheduler.

There is something call medium term scheduler, which is in between short term and long term and the short term scheduler is something like a what you see on a mobile phone, I click something immediately something comes and then it **is** is gone (())(11:51) right. So this is more for real time share interactive operating systems. These are the short term schedulers are very very important. Now a long term scheduler is invoked in seconds or minutes once in a second or minute while a short term scheduler is invoke every millisecond or 10s of milliseconds, right. So this is a difference. So what we will be, so we will be covering scheduling algorithms, which are true for both long term and short term scheduling and we will see how this could work.

(Refer Slide Time: 12:25)



CPU Scheduler

5

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- *Nonpreemptive vs. Preemptive*

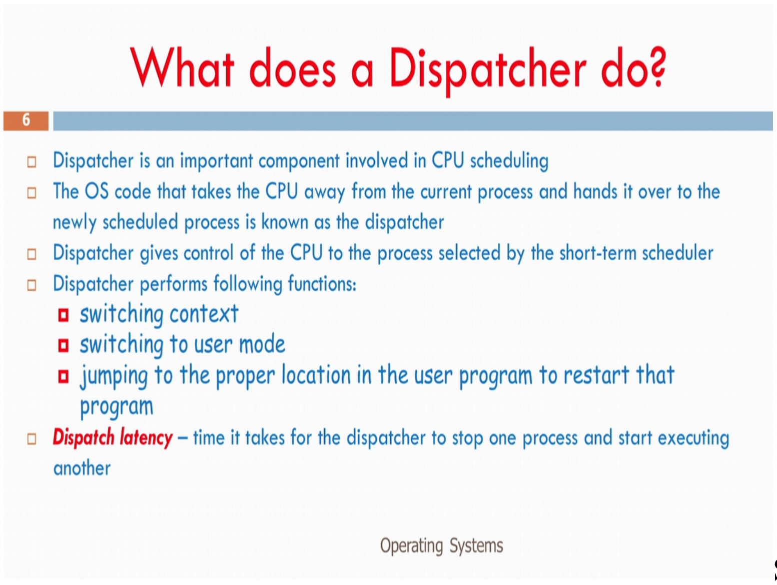
Operating Systems

So what is the role of a CPU scheduler to sum up? The role of the CPU scheduler is it, there be somebody who is running and it should switch from running to waiting state and or it can switch from running to ready state and it can also switch from waiting to ready state or it can terminate. So a scheduler for every process it can make one of these decisions, right. It can switch a process, which is currently at a running state we have waiting state, because it ask for some service from the operating system. It can switch it from a running state to a ready state, because it was an interrupt. The process did not ask anything, there was an external interrupt, so I can move it from a running state to a ready state, I can also switch it from a waiting state to a ready state because some system call is completed a process actually ask for a system call when it was running. So I put it on to a waiting state.

Now that system call is finished. So I can go it on to a ready state or the process has actually terminated or it has done some divide by zero, some exception thing, which it cannot continue I can the scheduler can go and terminate a process. Now there are two types of scheduling, non-preemptive and preemptive. These are all operating system topics, but we (())(13:40) to learn those things now, because these are security implications at a later stage. What is non-preemptive?

I cannot stop the process when it is executing to an external interrupt, right. So I cannot pull out the process when it is executing. So that is what we call as non-preemptive unless the process itself comes out. So (())(13:58) if I start executing a process, I cannot preempt it till it comes out, right and similarly, there is something call preemptive process in which I can pull out the process at any point time (())(14:09), okay. So there is a huge amount of security implications especially when there is a non-preemptive process, because when it goes in (())(14:17) and I cannot interrupt it, I really do not know what it is doing? It can do lot of things and it can hide what it has done so that at a later point, it can unhide it and (()(14:27) take it forward. So non-preemptive scheduling especially of tasks has security implications and we will look at it in great detail when we go to the remaining part of this course and subsequent courses.

(Refer Slide Time: 14:41)



What does a Dispatcher do?

6

- Dispatcher is an important component involved in CPU scheduling
- The OS code that takes the CPU away from the current process and hands it over to the newly scheduled process is known as the dispatcher
- Dispatcher gives control of the CPU to the process selected by the short-term scheduler
- Dispatcher performs following functions:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- ***Dispatch latency*** – time it takes for the dispatcher to stop one process and start executing another

Operating Systems

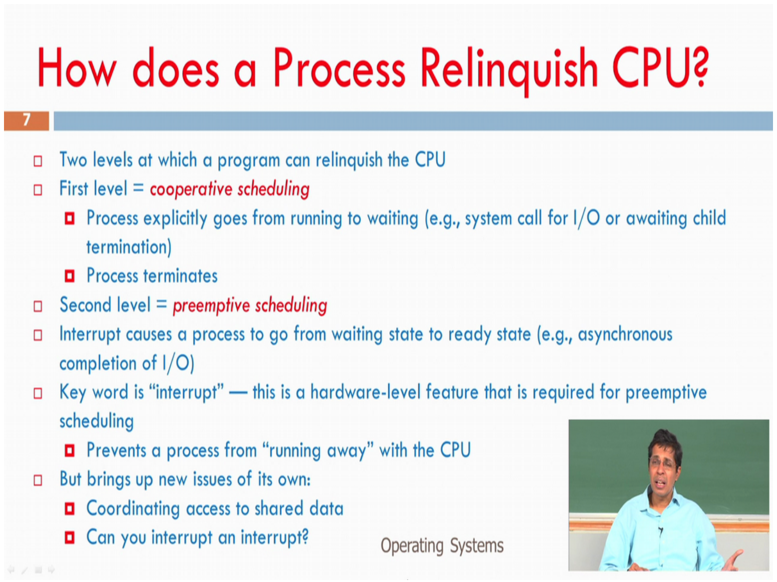
S

So now the next thing is there is a scheduling algorithm we will talk about these algorithms in some depth in the next module, but please to understand that after there is a (())(14:50) operating system decides that this is the process that needs to go to execution then it gives that link to the dispatcher and so what will the dispatcher do? The OS code that takes the CPU away from the current process and hands (())(15:04) it over to the newly scheduled process is called the dispatcher. So this is responsible for doing the entire task switching, okay and the dispatcher gives control of the CPU to the process selected by the short term scheduler or the long term scheduler. In this case, more it will be valid in the case of short term scheduling. So what it means that I switch context from one process to another process and then I also switch back to user mode, because the dispatcher if it is say kernel level thing, the scheduling is a kernel level algorithm. So essentially the dispatcher is also working at the kernel level.

So now I have to switch back to the user mode and start the program. So the user program should execute at the user level. Please, note there is a very settle (())(15:46) point here, I switch to the user mode and then give control to that program. So I do not give control to that program and then switch ask it to switch to the user mode that is a security vulnerability. So the dispatcher role is I have decided, the operating system decides which should be the next job to be scheduled. Now the this fellow the dispatches goes and does all the context switching, all the saving of the whole process information and collecting the loading of the new information etcetera and then it will first switch to user mode and then control to the program. This sequence is extremely important. So these are some small certain things that I want you all to understand so that tomorrow when we start looking at OS level security, these points need to be extremely taken care of.

So this is very important that I first switch to user mode and then give control to the user program. If I give control to the user program and then ask it to switch to means that means the user program for some point will be running in the kernel mode essentially it has more privileges and it can create issues. Now, there is of course a latency that is created and that is very important that distinguishes us between scheduling between threads verses scheduling between processes. This is dispatch latency is the time it takes so the dispatcher to stop one process and starts executing another.

(Refer Slide Time: 17:10)



How does a Process Relinquish CPU?

7

- Two levels at which a program can relinquish the CPU
- First level = *cooperative scheduling*
  - Process explicitly goes from running to waiting (e.g., system call for I/O or awaiting child termination)
  - Process terminates
- Second level = *preemptive scheduling*
- Interrupt causes a process to go from waiting state to ready state (e.g., asynchronous completion of I/O)
- Key word is "interrupt" — this is a hardware-level feature that is required for preemptive scheduling
  - Prevents a process from "running away" with the CPU
- But brings up new issues of its own:
  - Coordinating access to shared data
  - Can you interrupt an interrupt?

Operating Systems

Now we have already talked about preemptive and non-preemptive. As a process I will link with (())(17:15) CPU, because I did a mistake is call a trap, I did a segmentation overflow, I access to a page that does not exist, I accessed a segment, which is I went and start writing into a segment which is read only, I start writing into a code segment, I start executing from a data segment my stack has overflown, right. I access a page that is not there right. All I divide by zero. So these are all called traps and I come out, because once a trap comes immediately the hardware will detect that there is a trap and it will basically put the go to an interrupt service routine. Interrupt service routine the context switch automatically to interrupt service routine.

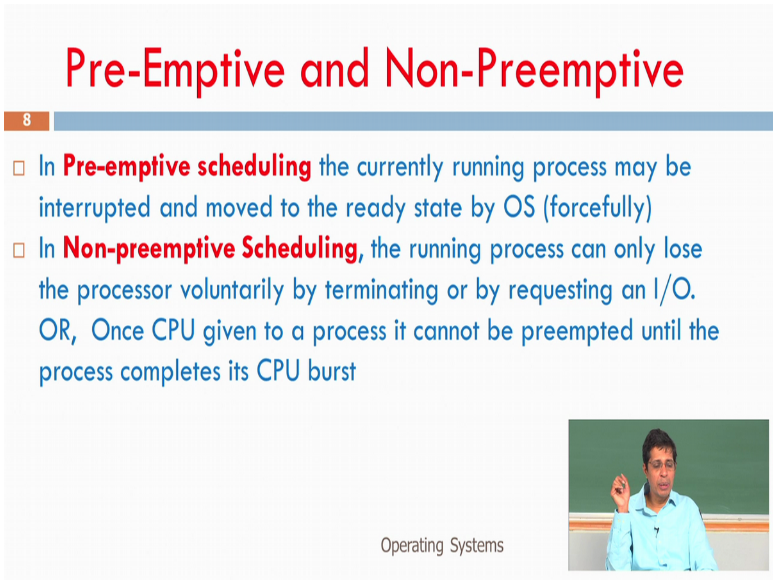
Interrupt service routine is a part of your operating system. So the operating system gets control and then it will basically pull out this process saying why did you do it, right. So this is basically what we call as a cooperative scheduling a part of a cooperative scheduling, in sense that when there is a something wrong the process cooperates with you and it comes out. It can also be that the process themself if they think they have done so much of CPU to want to give others a chance it can come out. So this is cooperative scheduling where the OS does not have a control of stopping a process that executing. This cooperative scheduling is also called as non-preemptive, but a preemptive scheduling is something where a OS goes and stops a current process and gets it out. How will a OS go and stop the current process. It cannot stop, because OS itself is a program. It has to execute to stop somebody else but somebody else is executing.

Now how it does is that it programs an interrupt for example, timer interrupt, right. Timer interrupt can get out at any point of time. So it goes and programs that interrupt. So when the operating system is in execution, it is executing on CPU, it goes and programs its interrupt and then comes out, after it comes out it cannot execute. So it is dormant some other process will be executing. The hardware interrupt will go at that particular time, since it is program it will create (())(19:20) and the moment the hardware interrupt goes, the control now comes back to the interrupt service routine, which is part of the operating system that is how in CPU an operating system which is dormant can get control over the CPU or get access to the CPU through what we call as hardware interrupt.

So the keyword is interrupt and this is a hardware level feature that is required for preemptive scheduling and so what is preemptive scheduling as we will see in the scheduling algorithms that we have going to see in the next module we will say that preemptive scheduling will stop a process from running away with the CPU meaning it will halve (())(19:59) the CPU for eternity (())(20:00) essentially not allowing the other process. So a preemptive scheduling actually avoids what we call as a starvation of a process. There process will not wait for long for getting into the CPU. So a preemptive scheduling algorithm can basically help in getting processes get access to the CPU within reasonable amount of time.

Now when we talk about preemptive scheduling, it brings up new issues of its own like coordinating, access to shear data and can you interrupt an interrupt etcetera. So these are all some questions that are coming up which you will answering the subsequent access.

(Refer Slide Time: 20:40)



Pre-Emptive and Non-Preemptive

8

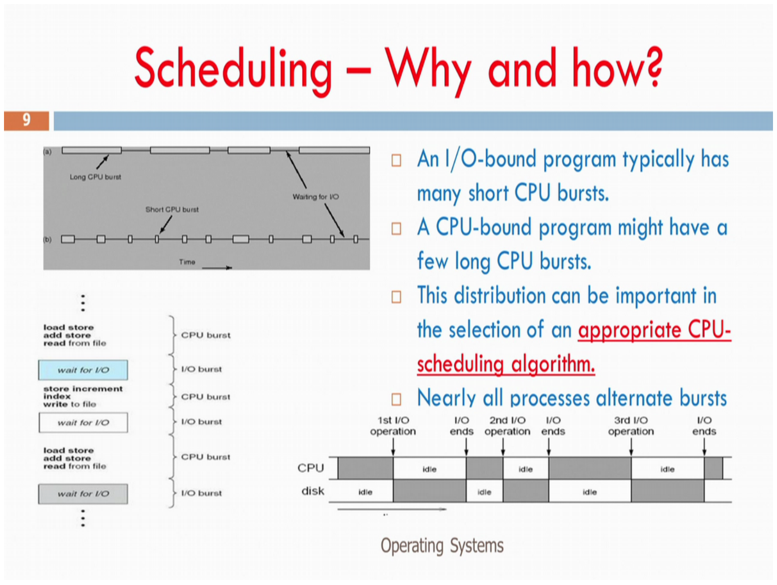
- In **Pre-emptive scheduling** the currently running process may be interrupted and moved to the ready state by OS (forcefully)
- In **Non-preemptive Scheduling**, the running process can only lose the processor voluntarily by terminating or by requesting an I/O. OR, Once CPU given to a process it cannot be preempted until the process completes its CPU burst

Operating Systems

So this slide basically summaries what I have explained so far, the difference between preemptive scheduling and non-preemptive. Note that preemptive scheduling is to remove the process forcefully and get the OS executing and that is possible only by the hardware interrupt. So you have to program the timer and the timer interrupt will one the operating system is executing, it has to program the timer and when it comes out then the process is executing, the timer will generate an interrupt based on which the current executing process is preempted.

So hardware timer based interrupt is extremely necessary for doing preemptive scheduling. In non-preemptive scheduling, the running process can only lose (()(21:23) the processor voluntarily by terminating or by requesting an IO or once CPU given to a process, it cannot be preempted until the process completes its CPU burst. So this is the difference between preemptive and non-preemptive scheduling.

(Refer Slide Time: 21:37)


(a)
Long CPU burst
Waiting for I/O
Short CPU burst
(b)
Time
load store
add store
read from file
wait for I/O
store increment
index
write to file
wait for I/O
load store
add store
read from file
wait for I/O
CPU burst
I/O burst
CPU burst
I/O burst
CPU burst
I/O burst
1st I/O operation
I/O ends
2nd I/O operation
I/O ends
3rd I/O operation
I/O ends
CPU
disk
idle

Scheduling – Why and how?

9

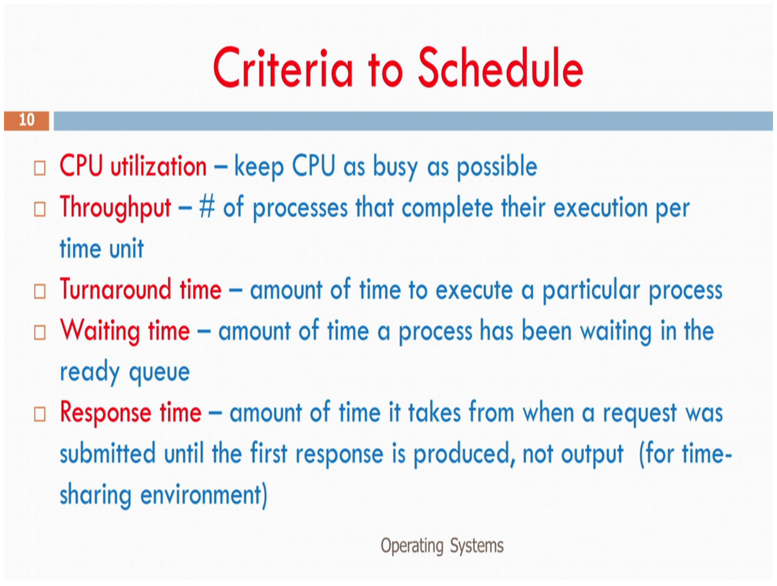
- An I/O-bound program typically has many short CPU bursts.
- A CPU-bound program might have a few long CPU bursts.
- This distribution can be important in the selection of an appropriate CPU-scheduling algorithm.
- Nearly all processes alternate bursts

Operating Systems

So as I had explained in the previous slides that a job essentially has a CPU burst and IO burst. So any IO bound program typically has many short CPU burst interlays between IO burst and a CPU bound program might have a few long CPU burst and we already saw that in the previous sections that if I have more than one processes that could be executed, the CPU burst of one can be merged with the IO burst of other and vice versa and basically, you can achieve good CPU utilization at the same time I can also do with the IO device utilization.

So I can merge the CPU burst of one program with the IO burst of another program and see those concurrency and so since this is possible, now I need somebody who will tell who should execute when and so there is a need for coordinating this activity and that is the role of a scheduler.

(Refer Slide Time: 22:40)



Criteria to Schedule

10

- CPU utilization – keep CPU as busy as possible
- Throughput – # of processes that complete their execution per time unit
- Turnaround time – amount of time to execute a particular process
- Waiting time – amount of time a process has been waiting in the ready queue
- Response time – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Operating Systems

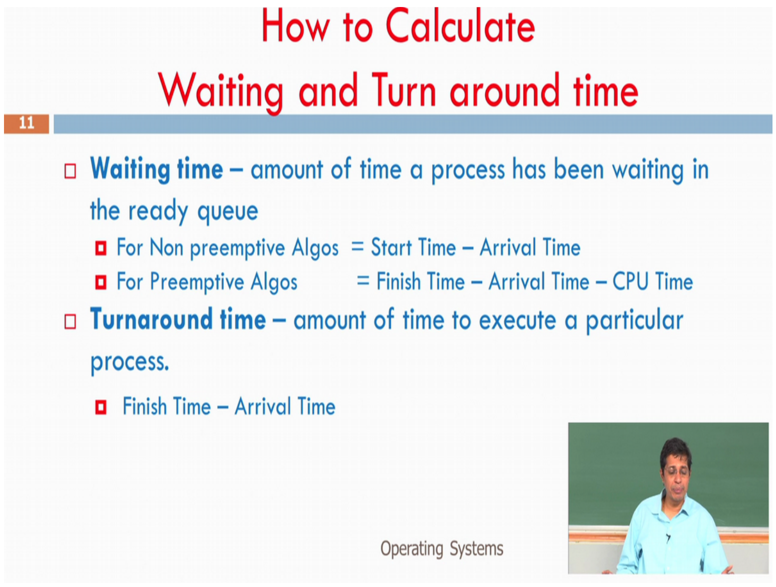
So suppose I want to design a scheduling algorithm. What will the scheduling algorithm do? So these are all the programs that are waiting. Which of the program should be scheduled next? It has to make this decision and while making this decision, there are some criteria that it needs to get out, one is CPU utilization always the CPU should be used, but that alone cannot be enough criteria, because I have a very big program and I say always that program will execute we will have maximum CPU utilization, but the other fellows will be dying (()) (23:06) or they will be starving. This is called the starvation.

So other important time that I am looking at this throughput, which is the number of processes that complete their execution per unit time. The moment I look at throughput and CPU utilization together, now I have got some notion of reducing the starvation, right. So if I have a one program that is executing for 100 units of time and I have two more programs there which is executing for 25-25 units of time then if I just put that 100 units of program forever, I will get could good (())(23:40) CPU utilization, but the throughput will be just one, but I can just do that I can finish of 125 another 25 and then this, my throughput now will be 2.5, because within the same 100 units of time 2 programs have completed of 25 ache and another program, which is you know 100 units 50 units of it is again allocated.

So if I look at throughput and CPU utilization together slowly I am bringing in the notion of reducing the starvation. The next thing is turnaround time which is amount to time to execute a particular process. There is also waiting time the amount of time a process has been waiting in the ready queue, because if it is waiting for something else in this waiting queue. If it is waiting in the waiting queue for some IO operation, it is depending upon how fast the IO device can work. So that is not the problem of scheduler. The problem of the scheduler is it has to keep this fellow if somebody enters the ready queue, it should be sent out as early as possible. If a process spends more time in the (where) ready queue then the scheduler is not so effective with respect to that process and then the response time amount of time it takes from when a request was submitted until the first response is produced, please note that, it is not the output full output, I say something that fellow say yes I am here and they put some window saying (())(25:04), we got this right and that becomes extremely important.

The response time is I ask for service how quickly I get a response saying (())(25:14) there service exists you can, because that gives us more continent in a file save operating system, somehow if the service could not be met at least some file save action will be taken. So in a real time operating system this response time is extremely important.

(Refer Slide Time: 25:29)



How to Calculate Waiting and Turn around time

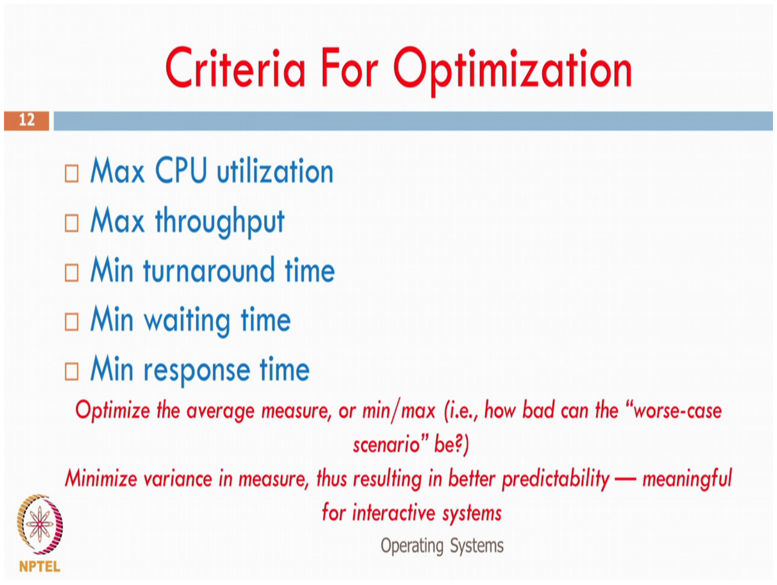
11

- **Waiting time** – amount of time a process has been waiting in the ready queue
  - For Non preemptive Algos = Start Time – Arrival Time
  - For Preemptive Algos = Finish Time – Arrival Time – CPU Time
- **Turnaround time** – amount of time to execute a particular process.
  - Finish Time – Arrival Time

Operating Systems

So how do we calculate the waiting and turnaround time? The waiting time off for a non-preemptive process is the start time minus arrival time. So the process comes in, it is staying in the ready queue that is arrival time and then it starts, after it starts, it never comes back, because it is non-preemptive thing. It goes and completes. So this start time minus arrival time is the waiting time on your waiting queue. For preemptive algorithms it is finish time minus arrival time minus CPU time, right. So after it arrives, it stays on the ready queue and at times goes the CPU again it comes back again it goes to CPU and again it comes back. So finish time minus arrival time minus CPU time. So this is the total waiting time. So the turnaround time waiting time is different from turnaround time. Turnaround time is nothing but finish time minus arrival time. So this is how we calculate this.

(Refer Slide Time: 26:23)



Criteria For Optimization

12

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

Optimize the average measure, or min/max (i.e., how bad can the "worse-case scenario" be?)

Minimize variance in measure, thus resulting in better predictability — meaningful for interactive systems

NPTEL

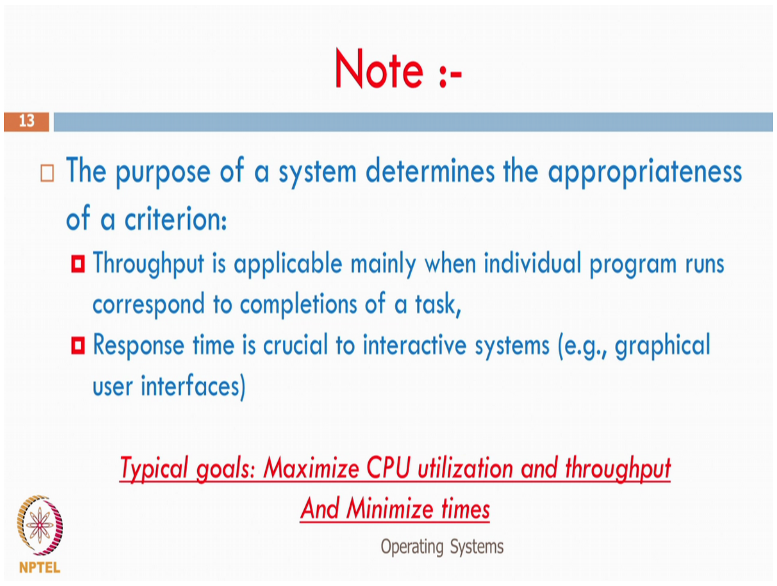
Operating Systems

So to sum up I need to maximize the CPU utilization, I need to maximize the throughput, I need to minimize the turnaround time, I need to minimize the waiting time, I need to minimize the response time. These are all the criteria based on which I need to do my scheduling algorithm and I can do many things, I can either do min max that how bad can the worst case seen arrow be, those type of things I can take care, right. So I would like to have a control over the worst case behavior. So I could design algorithms which will optimize in the sense that the worst case behavior will not be really worst, I could also design algorithms in which the average case will be excellent, the worst case can be bad, but it will happen once in a (())(27:06), the average where you can go and reboot that system etcetera. The average case would be very good. So I could design skip designing algorithms in which I could have verities of ways by which I could look to these criteria for optimization, but the most important thing is a second point.

I need to minimize the variance in measure, I cannot say one time it will be so fast another time it will be very slow, right. I should be either slow always or I should be fast always, right. So then only people will get a confidence in your system, if it is so non-predictable sometime, it is extremely fast sometimes extremely slow then variance is there then the confidence in your operating system you cannot say it is a slow one, it cannot say it is a faster, right.

So there should be some determinism in the variance that now my process enters, the time by which it will come out, there should be some fixed (())(28:03) binding limits and that is what we say as minimize the variance in this measure. So I could not have these measure one time it has one time the measure is 100, one time the measure is 0 then I do not know what to do with this measure. So this is very very important. So these are all the criteria for optimization and when we start doing this optimization, when we start looking at different ways by which this criterion could be managed, each way of managing this criterion basically gives me a different scheduling algorithm.

(Refer Slide Time: 28:36)



Note :-

13

- The purpose of a system determines the appropriateness of a criterion:
  - Throughput is applicable mainly when individual program runs correspond to completions of a task,
  - Response time is crucial to interactive systems (e.g., graphical user interfaces)

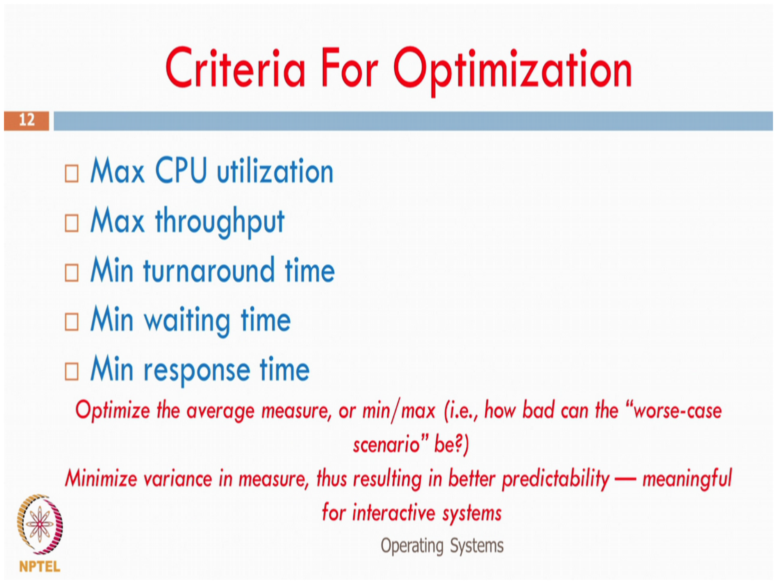
Typical goals: Maximize CPU utilization and throughput
And Minimize times

NPTEL

Operating Systems

The purpose of a system determines the appropriateness of a criterion, see that is very very important. So as I told you, when I am looking at program right, I am very happy about throughput, when I am looking at real time system I am not bothered about throughput and bothered about response time, but when I am looking at a batch right, I am more happy with that throughput as a programmer simple programme writing a conference paper and tomorrow morning I need a simulation to come out, I am looking I am interested in the turnaround time. So different users who use the operating system we have different perspective of what is efficient scheduling and that is very very important, okay. So typically people will look at maximizing CPU utilization and throughput and minimize the response times.

(Refer Slide Time: 29:26)



Criteria For Optimization

12

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

Optimize the average measure, or min/max (i.e., how bad can the "worse-case scenario" be?)

Minimize variance in measure, thus resulting in better predictability — meaningful for interactive systems

NPTEL

Operating Systems

So this is where this (())(29:27), okay. So with this we come to the end of this module, the takeaway from this module is that scheduling important and there are some scheduling algorithms we will discuss more about the scheduling algorithms in the next module, while in this module we have talked about some basic principles of scheduling and also some theoretical framework and certain optimization criteria for scheduling. Thank you.