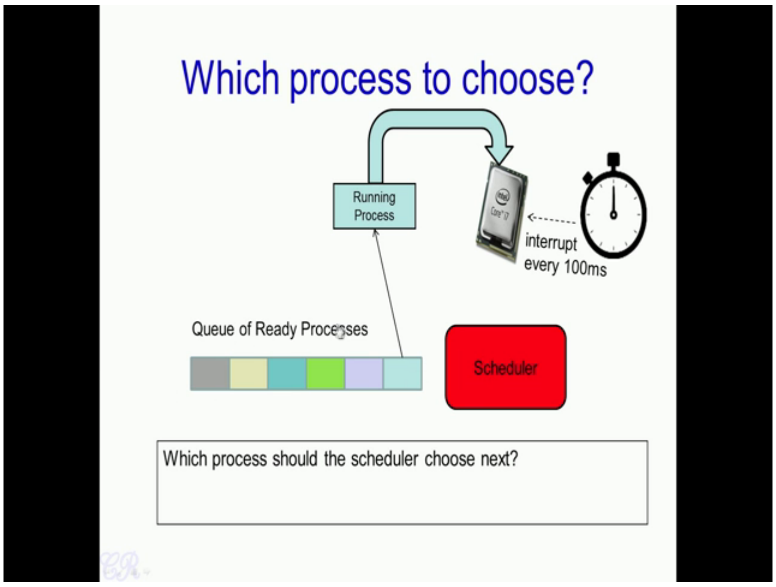
Introduction to Operating Systems

Prof. Chester Rebeiro
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Week – 05
Lecture – 18
CPU Scheduling

Hello. In this lecture, we will look at CPU Scheduling Algorithms.

(Refer Slide Time: 00:22)

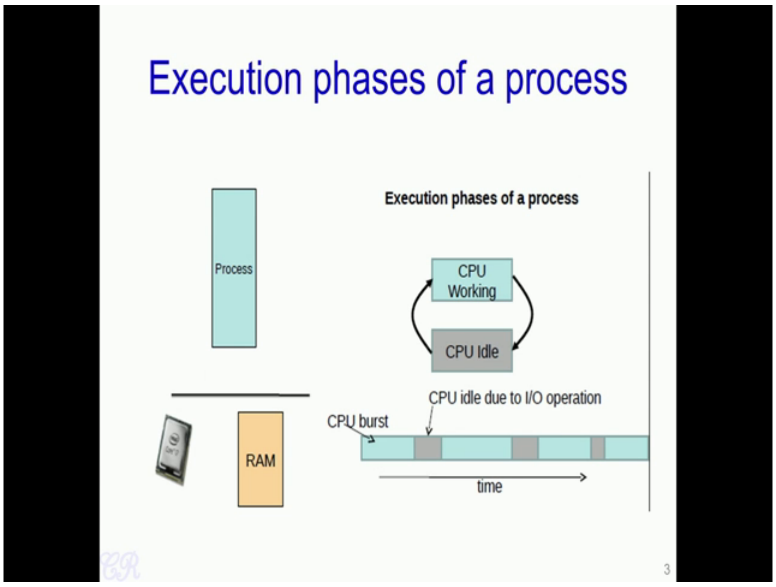


Which process to choose?

Which process should the scheduler choose next?

We had seen in operating systems that a scheduler present would choose a particular process from the ready queue and that process is assigned to run in the processor. Now the question that we were going to analyze now is how the scheduler should choose the next process; essentially, how the scheduler should choose a process to run on the CPU from the existing queue of ready processes.

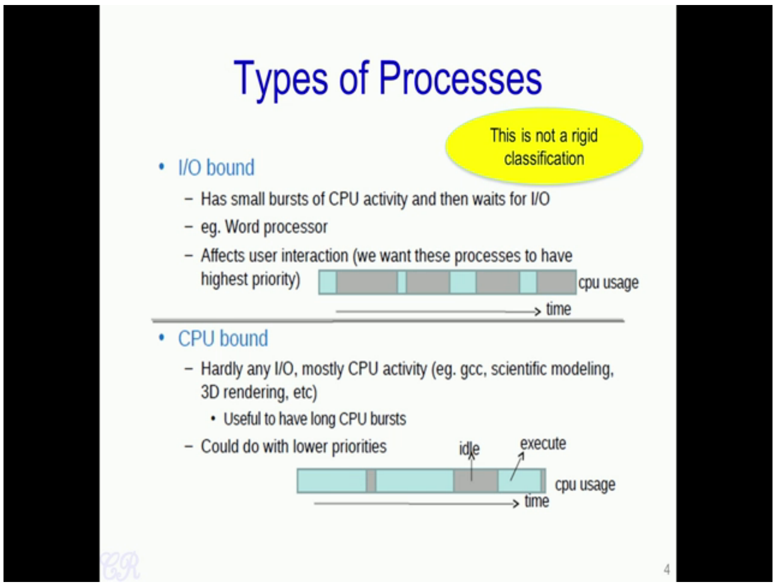
(Refer Slide Time: 00:53)


Execution phases of a process

Now to analyze this, we first look at execution phases of a process. Now any program has been known to have two phases of execution; one is when it is actually executing instruction which is known as a CPU burst, while the other is when it is blocked on an IO are not doing any operations, so in this particular case, the CPU is in idle.

Thus, as we look with over time, a particular process would have some amount of CPU burst in which it execute instructions on the processor, then it would have and some idle time in which it is waiting for a IO operation, then they would be burst of CPU time and so on. Thus, there is always this inter-reading between CPU burst and idle time that is a waiting for an operation.

(Refer Slide Time: 01:51)



Types of Processes

This is not a rigid classification

- I/O bound
  - Has small bursts of CPU activity and then waits for I/O
  - eg. Word processor
  - Affects user interaction (we want these processes to have highest priority)
- CPU bound
  - Hardly any I/O, mostly CPU activity (eg. gcc, scientific modeling, 3D rendering, etc)
    - Useful to have long CPU bursts
  - Could do with lower priorities

cpu usage
time
idle
execute
cpu usage
time

So, based on this cases of execution one could classify processes into two types. One is the IO bound processes, while the other is the CPU bound process. Why do you make this distinction between IO bound and CPU bound processes; essentially, this is from a scheduling perspective. We would like to give you IO bound processes a priority with which the allocated the CPU; essentially, we want that IO bound processes with lesser time for the CPU compared to CPU bound processes. So, why this is required, we can take with an example.

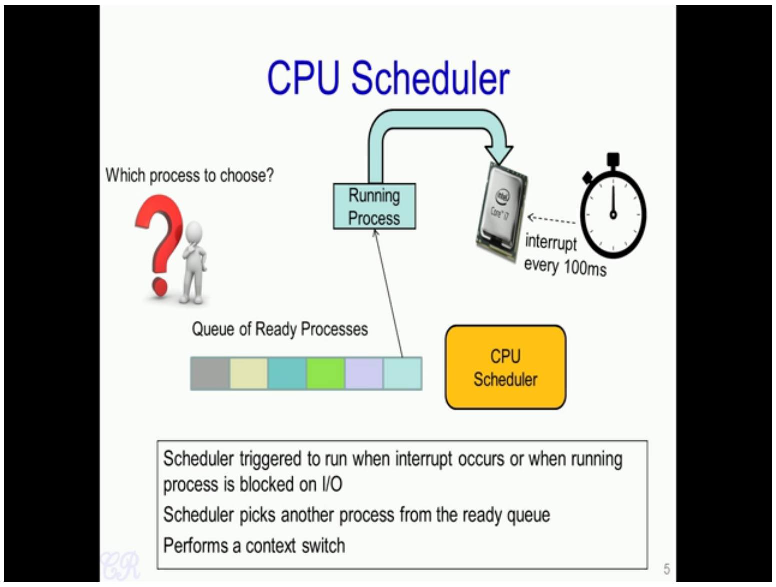
Suppose we are using a word processor such as note pad or vim and we are giving this IO bound process a very low priority. Now, suppose a user presses a key, because it as a low priority it does not get the CPU to very often and therefore, it would take some time before that key pressed by the user appears onto the screen. So, this may be quite uncomfortable for the user, therefore we would like to give the IO bound process such as the word processor higher priority with which it will get the CPU, so that the user interaction with the CPU comes more comfortable.

On the other hand, if you look at CPU bound processes, we could give a lower priority, now for instance if you take one of this applications the CPU bound applications like for instance say let us say gcc that is compiling a program and let us say you compiling a large program which takes 5 minutes. Now it will not effect this user much if the time taken to compile that particular program increases from 5 minutes to say 5.5 minutes.

Thus, a CPU bound processes could work with the lower priority. This classification between IO bound and CPU bound is not a rigid classification that is a process could be an IO bound process at one time, and after some time it could be behave like a CPU bound process.

So, to take an example of a process which behaves both like an IO bound as well as CPU bound, you could take for instance Microsoft excel. When we are actually entering data into the various cells in excel, it acts as IO bound process. So, it behaves like an IO bound process with small CPU burst and large times of IO cycles. While on the other hand when you are actually computing some statistic on the data entered, Excel will behave like a CPU bound process, where there is a large portion of CPU activity or the time taken to actually operate on that particular data.

(Refer Slide Time: 04:38)

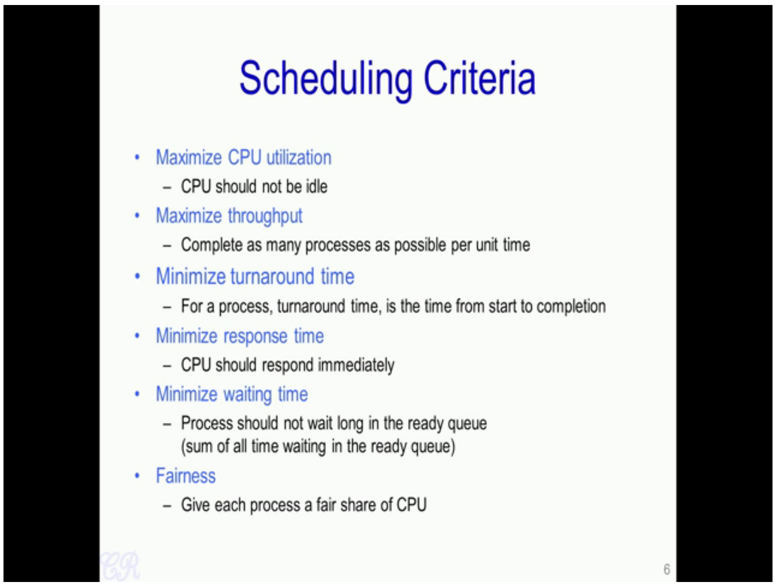


CPU Scheduler

Scheduler triggered to run when interrupt occurs or when running process is blocked on I/O
Scheduler picks another process from the ready queue
Performs a context switch

Now, let us come back to the question about the how the CPU scheduler should choose from the queue of ready processes, the next process to execute in the CPU. There could be several ways in which the scheduler could makes this choice; essentially, they could be several CPU scheduling algorithms which would looking into the queue and make a particular decision.

(Refer Slide Time: 05:05)



Scheduling Criteria

- Maximize CPU utilization
  - CPU should not be idle
- Maximize throughput
  - Complete as many processes as possible per unit time
- Minimize turnaround time
  - For a process, turnaround time, is the time from start to completion
- Minimize response time
  - CPU should respond immediately
- Minimize waiting time
  - Process should not wait long in the ready queue (sum of all time waiting in the ready queue)
- Fairness
  - Give each process a fair share of CPU

6

In order to compare these various scheduling algorithms, operating systems text books or operating system researches defines several scheduling criteria. So, these criteria could be used to actually to compare various scheduling algorithms to see the advantages and disadvantages of each of them. Let us go through each of these scheduling criteria one by one. The first scheduling criteria are the CPU utilization. The scheduling algorithms are designed in such a way so as to maximize CPU utilization. In other words, the CPU should be idle as minimum time as possible.

The next criteria, we will look at is the throughput; essentially, scheduling algorithms would try to complete as many processes as possible per unit time. The third criteria are the turnaround time; and these criteria looked at from a single process perspective. So, turnaround time is defined as the time taken for a single process from start to completion.

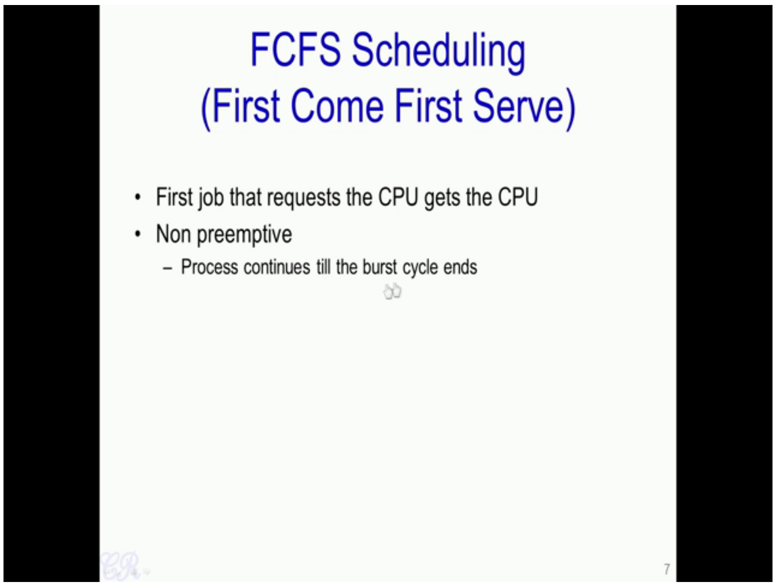
The fourth criteria are response time. So, this defined as the time taken from the point that when the process enters into the ready queue to the point when the process goes into the running state that is the time taken from the instant the process enters the ready queue to the time the CPU begins to execute instruction corresponding to that process. Another criteria, is the waiting time. Now this criteria, is based on the time taken by a process in the ready queue. Now as we know processes ready to run are present in the ready queue and it is required that they do not wait too long in the ready queue. So, scheduling algorithms could be designed in such a way that the waiting time of the average waiting time in the ready queue is minimized.

The final criteria we will see now is fairness. The scheduler should ensure that each process is given a fair share of the CPU based on some particular policy. So, it should not be the case that some processes, for instance, takes say 90 percent of the CPU while all other processes just get round 10 percent of the CPU. So, all these criteria need to be considered while designing a scheduling algorithm for an operating system.

A single scheduling algorithm will not be efficiently being able to cater all this criteria simultaneously. So, therefore, scheduling algorithms are therefore designed for to meet a subset of these criteria. For instance, if you consider real time operating system the scheduling algorithm for that system would for instance be designed to have minimum response time; other factors such as CPU utilization and throughput may be of secondary importance.

On the other hand, desktop operating system like a Linux will be designed for fairness, so that all applications running in the CPU or in the system are given a fair share of the CPU; criteria such as response time may be less important from that perspective. So, we will now look at several scheduling algorithms starting from the simplest one that is the first come first serve scheduling algorithm and go to more complex schedule in algorithm as we proceed.

(Refer Slide Time: 08:42)



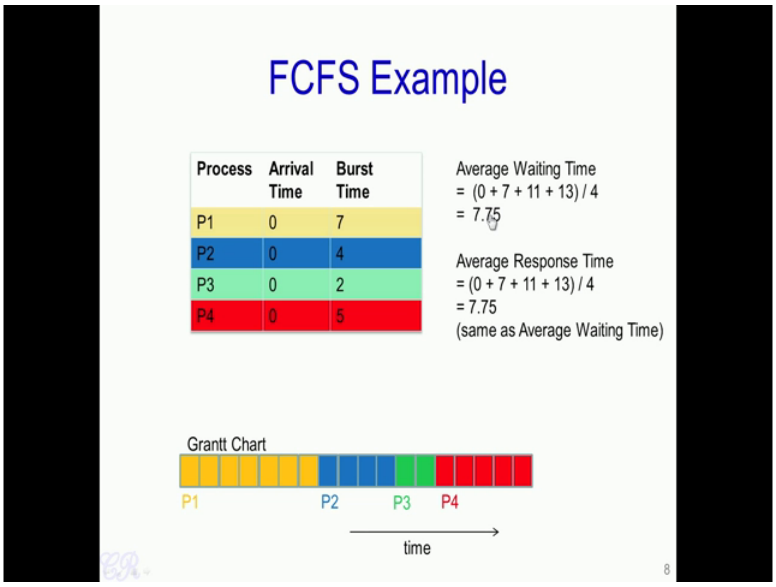
FCFS Scheduling (First Come First Serve)

- First job that requests the CPU gets the CPU
- Non preemptive
  - Process continues till the burst cycle ends

7

Let us look at the first come first serve of the FCFS scheduling algorithm. The basic scheme in this case is that the first process that request the CPU would be allocate the CPU; or in other words, the process, which enters into the ready queue would be allocated the CPU. So, this is a non preemptive scheduling algorithm which means that the process once allocated the CPU will continue to executive in the CPU until it burst cycle completes.

(Refer Slide Time: 09:19)



FCFS Example

Process	Arrival Time	Burst Time
P1	0	7
P2	0	4
P3	0	2
P4	0	5

Average Waiting Time
= (0 + 7 + 11 + 13) / 4
= 7.75

Average Response Time
= (0 + 7 + 11 + 13) / 4
= 7.75
(same as Average Waiting Time)

Let us see this with an example. Let us say we have system with four process are running; the processes are label P 1, P 2, P 3, P 4. And they have an arrival time, so the arrival time is present in the second column. The arrival time is defined as the time when these processes enter into the ready queue. So, for this particular very simple example, so we will consider that all processes enter the ready queue at the same time that is at the 0th time instant. The third column is the CPU burst time, so it gives the amount of a CPU burst for each process.

For instance, P 1 has a CPU burst of 7 cycles; P 2 has a burst of 4 cycles. Thus, this particular table we have like four processes, which all enters simultaneously into the ready queue at the time instance 0; and they have different CPU burst time for instance P 1 has 7 cycles; P 2 - 4 cycles; P 3 - 2 cycles; and P 4 - 5 cycles.

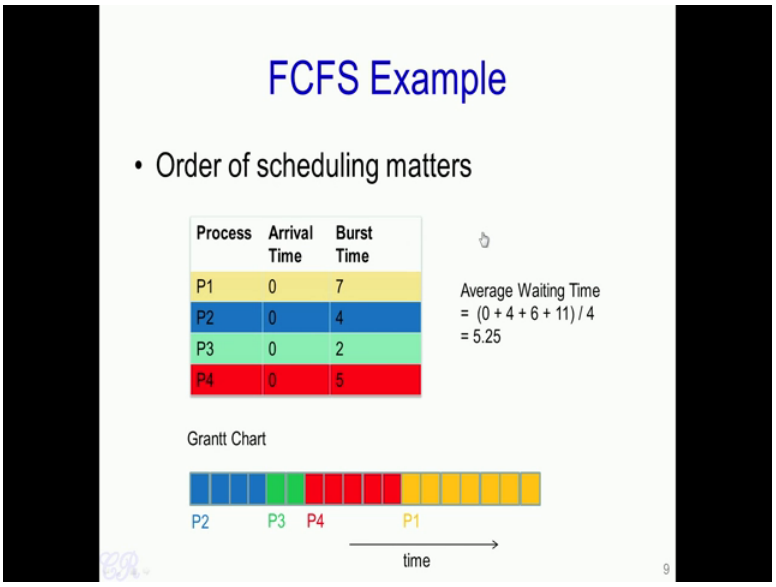
Now, we will see how these four processes get scheduled into the CPU or how these four processes get allocated the CPU. Since, all of these processes arrive at the same time the scheduler does not actually have a choice to make. So, he would pick randomly a particular ordering. For instance, let us say the scheduler picks P 1 to run, so P 1 runs for 7 cycles; and when it completes, the scheduler picks P 2; and P 2 runs for 4 cycles. After P 2 completes its burst, and P 3 executes in the CPU for 2 cycles; and then finally, P 4 is scheduled into execute for 5 CPU cycles. So, this is represented by a Grantt chart.

So, a Grantt chart is a horizontal bar chart developed as a production tool in 1917 by Henry L Grantt who was an American engineer and a Social scientist. So, essentially in a Grantt chart we have like several blocks over here; and each block represents a cycles of execution. So, for instance, P 1 executes for 7 cycles, so it has likes 7 blocks – 1, 2, 3, 4, 5, 6, 7. P 2 then executes for 4 cycles, so it given like 4 blocks. Then P 3 executes for 2 cycles, so it is given 2 blocks. And finally, P 4 executes for 5 cycles, so it is given 5 blocks.

So, we would compute the average waiting time for this particular case. We see that process P 1 enters into the ready queue at the instance 0, and immediately gets to execute in the CPU. Thus, it does not have to wait at all. The second process P 2 arrives also a 0 that is also arrives at this point, but gets to execute only after P 1 executes that is only after 7 cycles; thus, it is wait time is for process P 2 is 7. Similarly, the process P 3 which also enters in the 0th cycle gets to execute only after process P 1 and P 2 completes. In other words, it has to wait 11 cycles. And the fourth process in a similar way needs to waits 13 cycles. Therefore, the average waiting time in this particular case is 7.75 cycles.

Now, we can look at other scheduling criteria, which is the average response time. So, average response time in this case is the same as the average waiting time. The average response time is the time taken for a particular process to begin executing in the CPU minus the time it actually enters into the ready queue. So, for instance, P 2 enters into the ready queue at this instant, but begins to execute in the CPU only after 7 cycles. So, therefore, the response time for process P 2 is 7. Similarly, process P 3 has a response time of 11 because it has waited for 11 cycles to actually begin executing in the CPU. So, on average, the average response time is 7.7 5 just like the average waiting time.

(Refer Slide Time: 14:07)



FCFS Example

- Order of scheduling matters

Process	Arrival Time	Burst Time
P1	0	7
P2	0	4
P3	0	2
P4	0	5

Average Waiting Time
= (0 + 4 + 6 + 11) / 4
= 5.25

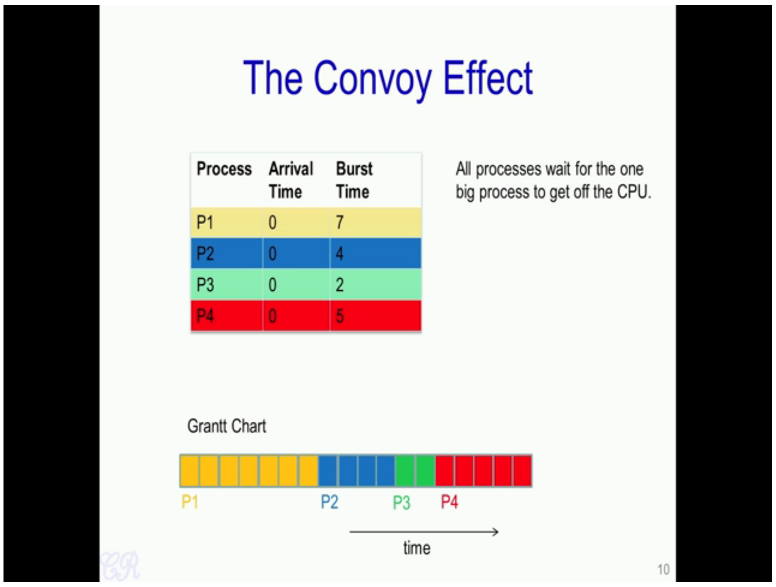
Grantt Chart

P2
P3
P4
P1
time
9

Now, one characteristic of FCFS scheduling algorithm is that the order of scheduling matters. In the previous slide that is in this slide we had assume that P 1 executes then P 2 executes then P 3 then P 4, and we have got average waiting time and average response time 7.75 cycles. Now, suppose we just change the ordering and let us say the ordering is now as follows that P 2 executes, then P 3 executes then P 4 finally, P 1.

In such a case, if we compute the average waiting time it we see that it gets reduce from 7.75 cycles to 5.25 cycles. Similarly, if you compute the average response time, you will see that the response time is also 5.25 cycles. In a similar way, you could compute the other criteria, which we are mentioned over here. And you will able to actually see the difference with the two ordering schemes.

(Refer Slide Time: 15:13)



The Convoy Effect

Process	Arrival Time	Burst Time
P1	0	7
P2	0	4
P3	0	2
P4	0	5

All processes wait for the one big process to get off the CPU.

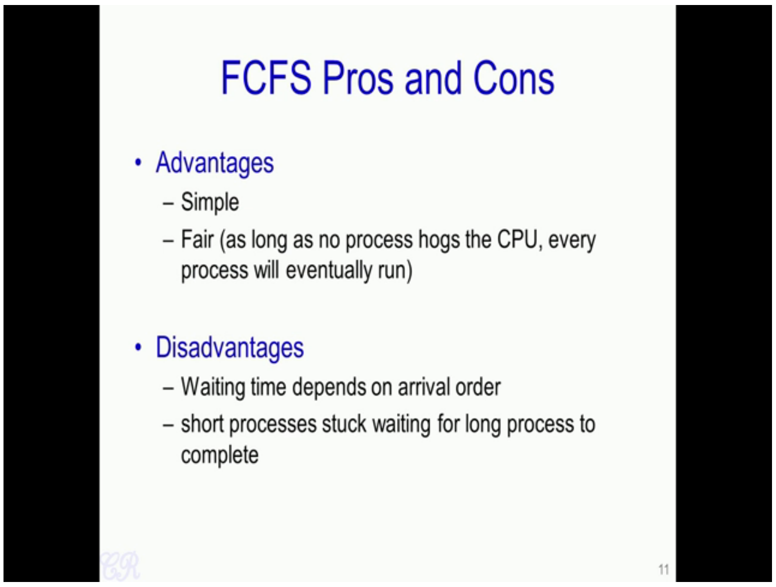
Grantt Chart

P1
P2
P3
P4
time

10

Another characteristic of the FCFS scheduling algorithm is the convoy effect. Essentially, in this particular case we see that all processes wait for one big process to get off the CPU. So, for instance, in this case, until or other even though all processes enter the CPU at the same time, all processes have to wait for P 1 to complete only then that they could be scheduled. So, if we have a large processes over here instance, we have a process P 1 instead of burst time 7 takes a burst time of say 100, all other processes P 2, P 3 and P 4 would wait for 100 block cycles before they actually be able to get to execute in the CPU. So, this is huge drawback of FCFS scheduling scheme.

(Refer Slide Time: 16:10)



FCFS Pros and Cons

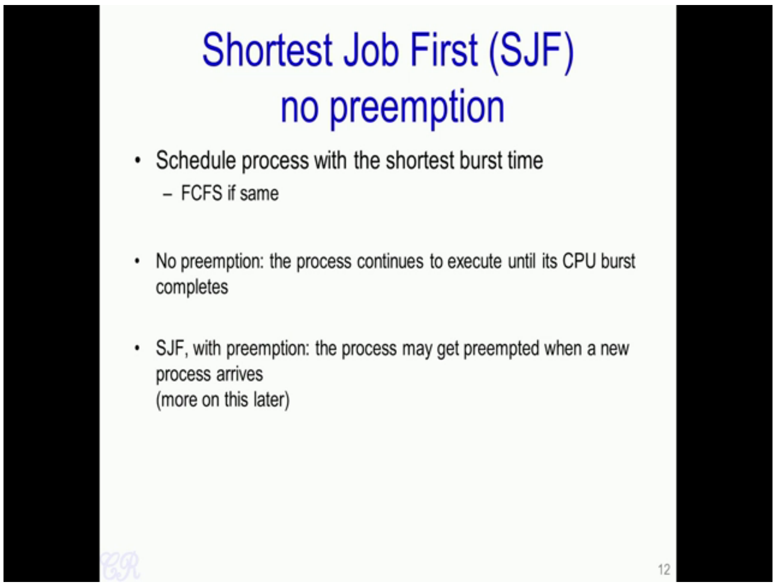
- Advantages
  - Simple
  - Fair (as long as no process hogs the CPU, every process will eventually run)
- Disadvantages
  - Waiting time depends on arrival order
  - short processes stuck waiting for long process to complete

11

However, FCFS scheduling algorithm has several advantages. The first thing is it is extremely simple. The scheduling algorithm could complete very quickly and therefore, the time taken by the scheduling algorithm will be very less, and you would end up with very less contexts delays while changing the contexts. Another advantage of the FCFS scheduling algorithm is that it is fair. As long as no process hogs the CPU every process will eventually run; or in other words, as long as every process terminates at some point every other process in the ready queue will eventually get to execute in the CPU.

Now the drawback or the disadvantage of the FCFS schedulers as we have seen is that the waiting time depends on the arrival order. So, we have seen the example in the previous slides. Another disadvantage is that short processes are stuck in ready queue waiting for long processes to complete; or rather, this is the convoy effect that we have just looked at.

(Refer Slide Time: 17:19)



Shortest Job First (SJF) no preemption

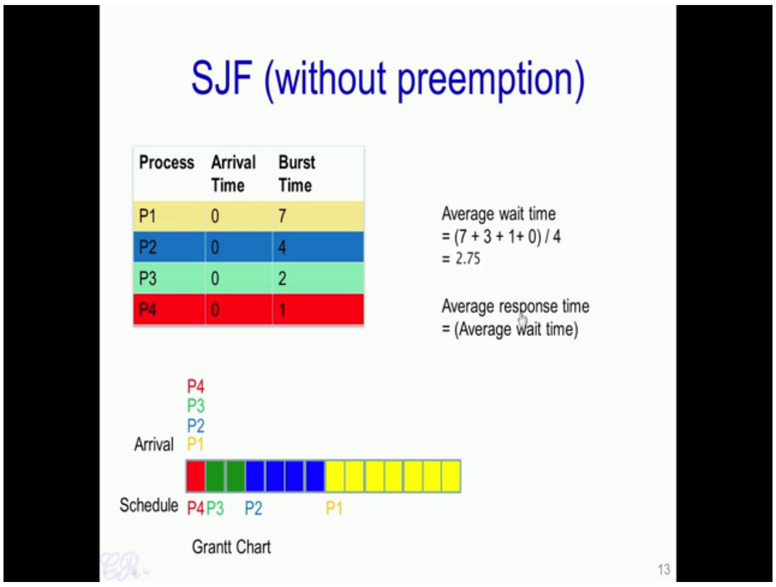
- Schedule process with the shortest burst time
  - FCFS if same
- No preemption: the process continues to execute until its CPU burst completes
- SJF, with preemption: the process may get preempted when a new process arrives
  (more on this later)

12

Now, let us look at another scheduling algorithm know as the shortest job first scheduling algorithm. In this particular scheduling algorithm, the job or the process with the shortest CPU burst time is scheduled before the others. Now if you have more than one process with the same CPU burst time then standard FCFS scheduling is used. There are two variants of the shortest job for scheduling algorithm.

The first is the no preemption of variant, while the second one is the shortest job first with preemption. Now in the SJF with no preemption, the process will continue to execute in the CPU until its CPU burst completes. In the second variant with preemption, it may be possible that the process which may get preempted when a new process arrives to the ready queue. We will see more on this later, but first we will start with the shortest job first variant with no preemption.

(Refer Slide Time: 18:21)



SJF (without preemption)

Process	Arrival Time	Burst Time
P1	0	7
P2	0	4
P3	0	2
P4	0	1

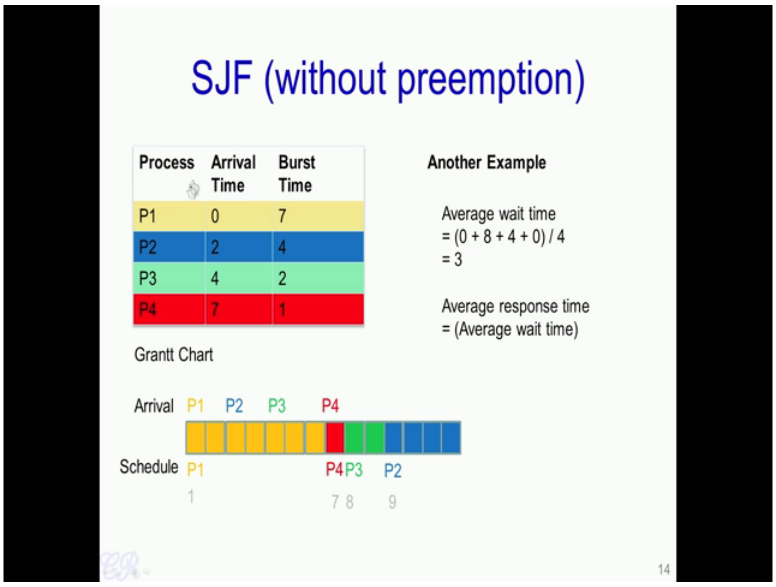
Average wait time
= (7 + 3 + 1+ 0) / 4
= 2.75

Average response time
= (Average wait time)

Let us take the same example that we have seen in the previous case we had the four processes P 1 to P 4m and all of them arriving at the instant 0 that is at instant 0 these four processes P 1 to P 4 get into the ready queue. And each of this processes have a different CPU burst time that is 7, 4, 2 and 1 respectively. Now in the first instant, the CPU scheduler will look at the various burst times and find the one, which is minimum.

So, in this case we see that P 4 has the minimum CPU burst time, so that were scheduled first. So, first the process P 4 gets scheduled until it completes, in this particular case it completes in 1 cycle. Then among the remaining three, we see that P 3 has the lowest CPU burst time. So, process P 3 gets scheduled and completes till and executes till it completes its burst. Then P 2 gets scheduled because it has a burst time of 4, while P 1 has a higher burst time of 7. And finally, P 1 gets to executes till completion. Now, the average wait time, if we compute this is 2.75 while the average response time is also 2.75 as in the wait time case.

(Refer Slide Time: 19:43)



SJF (without preemption)

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	2
P4	7	1

Another Example

Average wait time
= (0 + 8 + 4 + 0) / 4
= 3

Average response time
= (Average wait time)

Grantt Chart

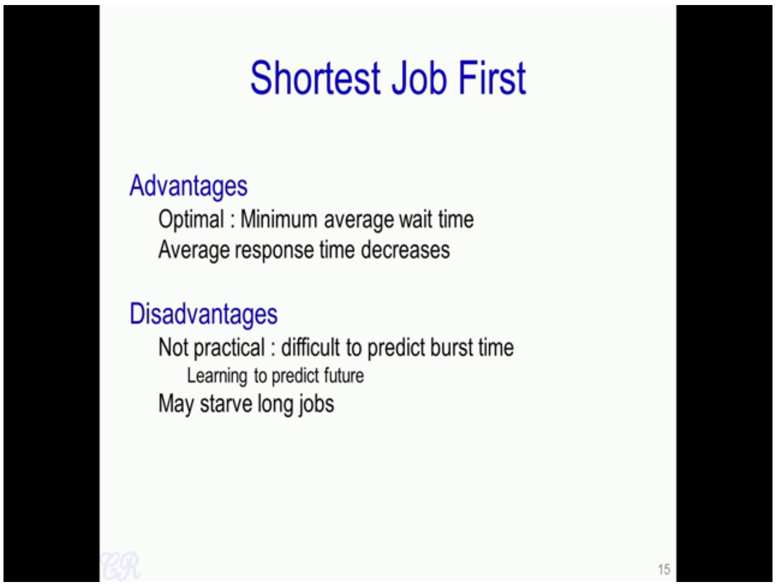
Arrival
P1
P2
P3
P4
Schedule
P1
P4
P3
P2
1
7
8
9
14

Now let us look at another example of shortest job first without preemption. So, we will take the same four processes P 1 to P 4 and each of these process have the same burst time as before that is 7, 4, 2 and 1 respectively. However, they arrive at different instance that is the moment the instant in which they enter into the ready queue would be different. So, P 1 enters at the 0th instant, P 2 in the second instant, P 3 at the fourth instant and P 4 in the 7th instant. Now, this is a slight modification in the Grantt chart, where in addition to showing which process is executing in the CPU, it also show the order in which process arrive it shows that the P 1 arrives first, then P 2 in the second instant, then P 3, finally P 4 in the seventh instant.

Now, when the scheduler begins to execute at this particular instant, the only process that has arrived at this particular point is P 1 therefore, it schedules P 1 to execute. So, P 1 executes for its entire burst that is of 7 cycles, and then at this particular cycle, the scheduler enters again or scheduler executes again and this time it has got three processes to choose from all P 2, P 3 as well as P 4 have arrived in the ready queue.

And out of them P 4 has the shortest burst time therefore, it is chosen for execution. Therefore, P 4 executes in the CPU, and then P 3 because P 3 has a shortest burst time than P 2 and finally, P 2 get executed. So, if we compute the average wait time, we see that it is 3 cycles.

(Refer Slide Time: 21:37)



Shortest Job First

Advantages

Optimal : Minimum average wait time

Average response time decreases

Disadvantages

Not practical : difficult to predict burst time

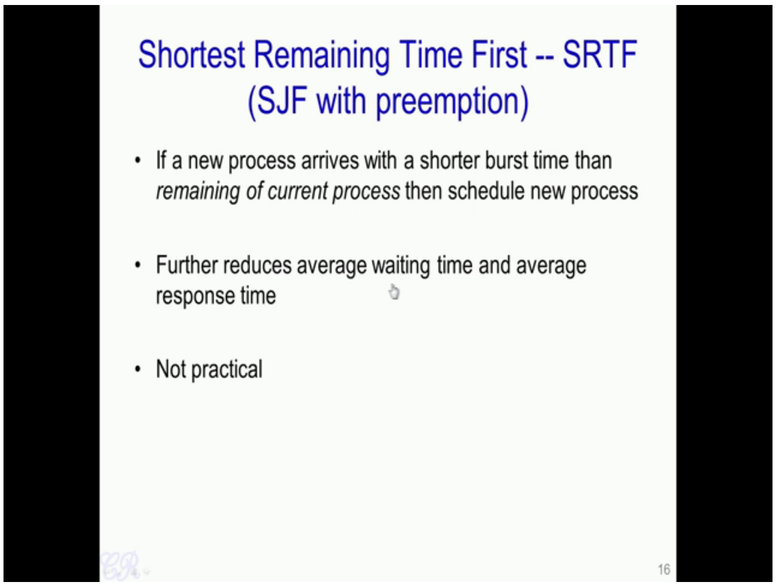
Learning to predict future

May starve long jobs

15

The advantage of shortest job first scheduling algorithm is that is optimal. It will always give you the minimum average waiting time. And as a result of this, the average response time also decreases. The main disadvantage of the SJF scheduling algorithm is that is not practical; essentially, it is very difficult to predict what the burst time would be. The another drawback of the SJF scheduling algorithm is that some jobs may gets starve; essentially if you have a process which has an extremely long CPU burst time, then it may never get a chance to actually to execute in the CPU.

(Refer Slide Time: 22:19)



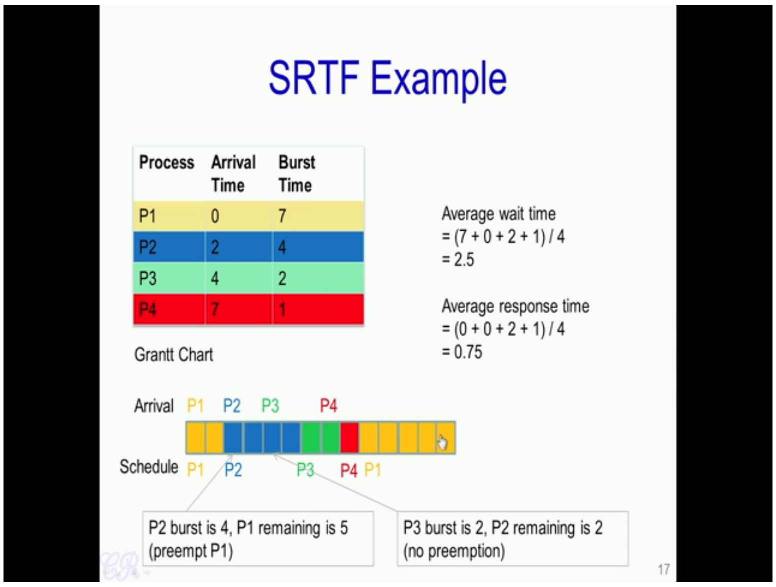
Shortest Remaining Time First -- SRTF (SJF with preemption)

- If a new process arrives with a shorter burst time than *remaining of current process* then schedule new process
- Further reduces average waiting time and average response time
- Not practical

16

Now, we will look at shortest job first scheduling algorithm with preemption. So, this is also the shortest remaining time first. The basic idea in this algorithm is that if a new process arrives into the ready queue, and this process has a shorter burst time than the remaining of the current process then there is a context switch and the new process gets scheduled into the CPU. This further reduces the average waiting time as well as the average response time. However, as in the previous case that is the shortest job first with no preemption here also it is not practical. Let us understand more on this with an example.

(Refer Slide Time: 23:02)



SRTF Example

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	2
P4	7	1

Average wait time
= (7 + 0 + 2 + 1) / 4
= 2.5

Average response time
= (0 + 0 + 2 + 1) / 4
= 0.75

Grantt Chart

P2 burst is 4, P1 remaining is 5 (preempt P1)

P3 burst is 2, P2 remaining is 2 (no preemption)

Let us take the same example of 4 processes with burst time 7, 4, 2 and 1; and arrival times at 0, 2, 4 and 7 will develop the Grantt chart as the time processes. So, at the instant 0, the only process which is present is P 1; and therefore, the scheduler has no choice, but to schedule P 1 onto the CPU. Thus P 1 would execute for 2 clock cycles.

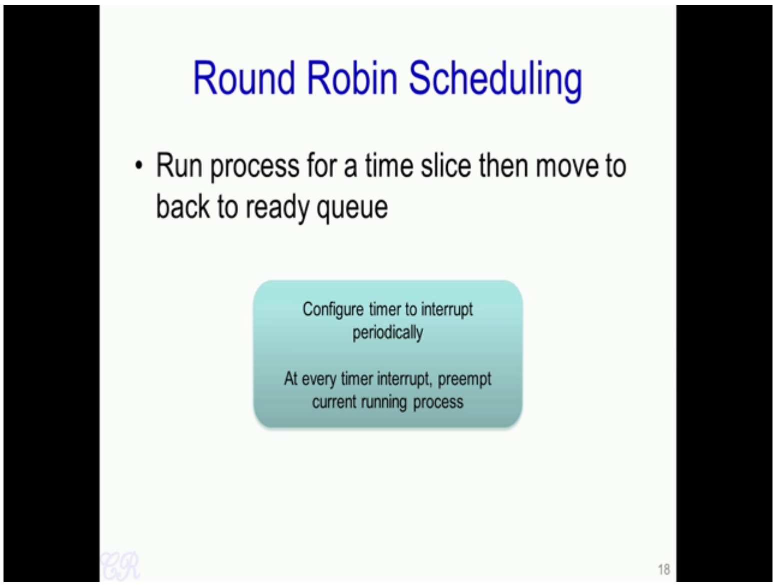
Now after the second clock cycle the process P 2 has entered into the ready queue. Now you see that P 2 has a burst of 4 cycles. However, P 1 has a remaining burst of 5 cycles. So, what we mean by this is that out of the CPU burst time of 7 cycles, P 1 has completed 2 cycles. So, what is remains is 5. Now the scheduler will see that P 1 has 5 cycles, which is greater than P 2, which has burst of 4 cycles. Therefore, it will do a context switch and a schedule P 2 to run. So, P 2 will run for 2 clock cycles and then P 3 arrives at the fourth clock instant.

So, at this particular instant, the scheduler will find out that P 3 has burst time of 2 cycles, while P 2 has a remaining a burst time of 2 cycles, we achieve 2 because out of the 4 cycle burst time for P 2, it has completed 2. So, what remain are 2 more cycles. Since P 2 the old process which is running on the CPU has a 2 cycle remaining burst time, and P 3 the new process also has 2 cycles burst time therefore, there is no preemption and P 2 will continue to execute.

Now after P 2 completes, in this particular case, P 3 executes for 2 cycles, and then P 4 enters now after which if you will verify will not cause any preemption. So, after P 3 completes, the scheduler decides to run either P 4 or P 1. So, we see that P 4 has a burst cycle of 1 while P 1 has a remaining burst cycle of 5, therefore the scheduler will decide to choose P 4 over p 1. So, P 4 runs on the CPU and after it completes P 1 will executes for its remaining burst time.

So, if you compute the average wait time, you see that it reduces to 2.5, while the average response time reduces considerably to 0.75. However, as we mentioned before just like the shortest job first this scheduling algorithm is also not feasible to implement in practice, because it is very difficult to actually identify what the burst time of a process is and even more difficult to identify what the remaining burst time of the process would be.

(Refer Slide Time: 26:08)



Round Robin Scheduling

- Run process for a time slice then move to back to ready queue

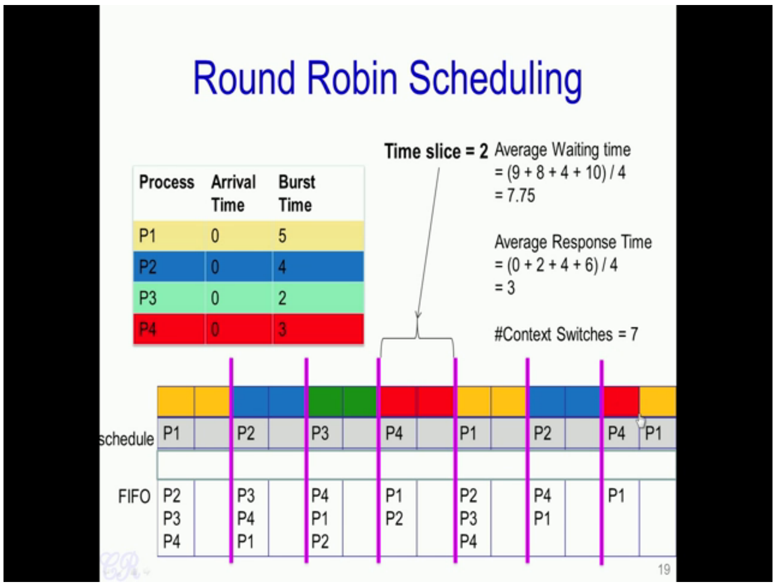
Configure timer to interrupt periodically

At every timer interrupt, preempt current running process

18

Now we will look at another scheduling algorithm known as the round robin scheduling algorithm. So, essentially with the round robin scheduling algorithm, a process runs for a time slice that is the process executes for a time slice and when the time slice completes it is moved on the ready queue. So, in order to achieve this round robin scheduling algorithm which is also a preemptive scheduling algorithm; now in order to achieve round robin scheduling, we need to configure the timer in the system to interrupt the CPU periodically; at every timer interrupt, the kernel would preempt the current process and choose another process to execute in the CPU.

(Refer Slide Time: 26:50)



Round Robin Scheduling

Process	Arrival Time	Burst Time
P1	0	5
P2	0	4
P3	0	2
P4	0	3

Time slice = 2

Average Waiting time
= (9 + 8 + 4 + 10) / 4
= 7.75

Average Response Time
= (0 + 2 + 4 + 6) / 4
= 3

#Context Switches = 7

Let us discuss the round robin scheduling algorithm with an example. So, one difference with respect to the other scheduling algorithm that we have seen so far is the notion of time slice. So, this is the Grantt chart. So, especially see that periodically in this case with a period equal to 2 that is we have keeping a time slice equal to 2, there is a timer interrupt that occurs and the timer interrupt would result in the scheduler being run and potentially another process being scheduled into the CPU. So, a data structure which is very useful in implementing the round robin scheduling algorithm is the FIFO.

This particular FIFO shows the processes that need to be executed next into the CPU. For example, in this particular case, P 2 is at the top at the FIFO, so it is a next process which gets executed in the FIFO. So, P 2 gets executed over here. So, for example, we will still consider the 4 processes as we have done before that is P 1 to P 4 and we will assume that all of them arrive at the instant 0 and go into the FIFO of the ready queue and they have burst times of 5, 4, 2 and 3 respectively.

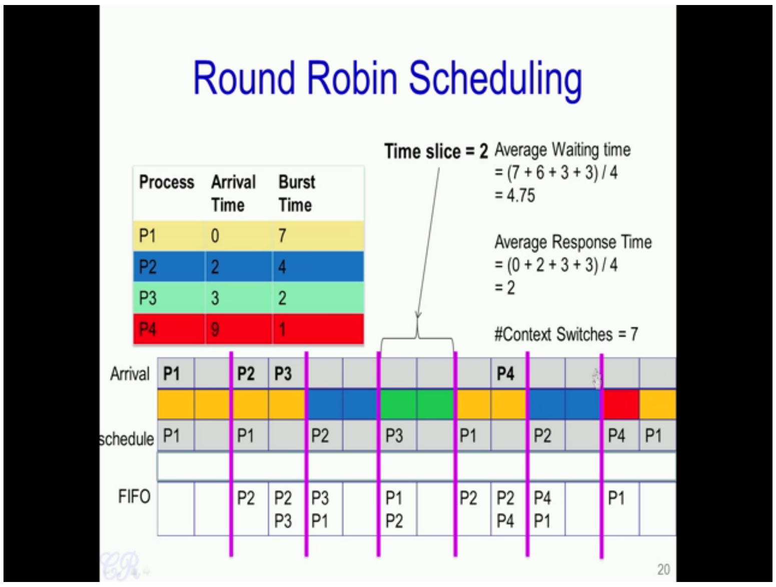
Let us say for discussion that the scheduler starts off with this particular order P 1 P 2 P 3 and P 4, so and it first chooses to execute p 1. So, P 1 executes for 2 cycles and then there is interrupt which occurs leading to a context switch and then the top of the FIFO in this particular case P 2 gets scheduled into the CPU, while P 1 gets pushed into the FIFO. So, P 2 then executes for 2 cycles until the next timer interrupts in which case the time slice of 2 cycles completes and then it gets pushed into the FIFO. So, P 2 is at the bottom of the FIFO, while P 3 which is at the top of the FIFO gets scheduled to run So, in this way, every two cycles a new process may get scheduled into the CPU and execute.

So, if we compute the average waiting time. So, in this particular case, we will see that the average waiting time and the average response time is different. So, what is the average waiting time for P 1? So, P 1 executes 2 cycles here, 2 cycles here and completes execution over here. So, it waits in the in the ready queue in the remaining of the cycles. The number of cycles it waits is 1, 2, 3, 4, 5, 6, 7, 8, 9. So, P 1 waits for 9 cycles; now P 2 waits for 8 cycles – 1, 2, 3, 4, 5, 6, 7, 8,; P 3 for 4 cycles, and P 4 for 10 cycles. The average waiting time is 7.75 cycles.

Now, to compute the average response times as we defined it before, the response time is the time the process enters into the ready queue to the time it begins to execute in the CPU that latency would be the response time. So, P 1 for instance has a response time of 0 because it enters into the ready queue or enters into the FIFO and gets executed immediately. P 2 on the other hand enters in the 0th cycle, but gets to execute only after 2 cycles, so it has a response time of 2.

Similarly, P 3 enters at 0 and executes at only this point. So, at this instant therefore, it has a response time of 4; and P 4 has a response time of 6 therefore, the average response time is 3. Now the number of context switch is that occur are 7. So, in this case, 1, 2, 3, 4, 5, 6 and a context switch occurs over here because the process P 4 is existing out and P 1 gets continues to execute. The numbers of context switches over here are 7 that are 1, 2, 3, 4, 5, 6 and the seventh context switch occurs over here then P 4 exits and P 1 gets switched into the CPU.

(Refer Slide Time: 31:42)



Round Robin Scheduling

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	3	2
P4	9	1

Time slice = 2

Average Waiting time
= (7 + 6 + 3 + 3) / 4
= 4.75

Average Response Time
= (0 + 2 + 3 + 3) / 4
= 2

#Context Switches = 7

Arrival	**P1**		**P2**	**P3**						**P4**				
	🟧	🟧	🟧	🟧	🟦	🟦	🟩	🟩	🟧	🟧	🟦	🟦	🟥	🟧
schedule	P1		P1		P2		P3		P1		P2		P4	P1
FIFO			P2	P2 P3	P3 P1		P1 P2		P2	P2 P4	P4 P1		P1	

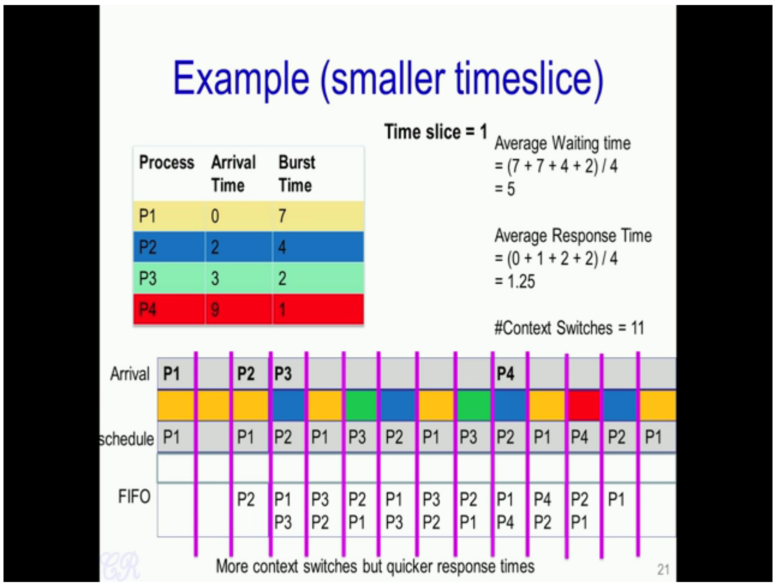
20

Now let us take another mode complex example of round robin scheduling, where we also have arrival times, which are not the same. So, P 1 is arriving at the 0th instant; P 2 at the second P 3 and P 4 at the third and ninth respectively. So, we can similarly draw the Grantt chart and the states of the FIFO for this case. So, to start with in the 0 instant, the only process which has arrived is P 1, and therefore, P 1 executes for 2 cycles. And at this particular point, when the timer interrupt occurs, no other process is present as yet, therefore P 1 will continue to execute for another 2 cycles for another time slice.

However, in this time slice, we have two processes which are entered into the ready queue; these are the process P 2 and P 3. So, P 2 arrives at this interval while P 3 arrives in this interval and they get added into the FIFO. So, at the second time slice completion, there is a context switch, and P 2 gets scheduled into the CPU to execute, while P 1 which was executing will then go into the FIFO. So, P 2 executes for 2 cycles, then P 3 executes for 2 cycles, then P 1 executes for 2 cycles and at that time P 4 has arrived and gets added into the FIFO.

Now, we have three processes P 1, P 2 and P 4 and these gets to schedule to run for a period of time. The average waiting time in this case is 4.75, while the average response time is 2. So, how is the average waiting time 4.75, it means that process P 1 has waited for 7 cycles. So, before it completes, so that is 1, 2, 3, 4, 5, 6 and 7. While process P 2 has waited for 6 cycles, so it is 1, 2, 3, 4, 5, 6; process 3 has waited for 3 cycles, and process 4 has waited for 3 cycles. The average response time can be verified to be equal to 2, and the number of context switches was as before 7.

(Refer Slide Time: 34:15)



Example (smaller timeslice)

Time slice = 1

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	3	2
P4	9	1

Average Waiting time
= (7 + 7 + 4 + 2) / 4
= 5

Average Response Time
= (0 + 1 + 2 + 2) / 4
= 1.25

#Context Switches = 11

Arrival	P1		P2	P3						P4				
	🟧	🟧	🟧	🟦	🟧	🟩	🟦	🟧	🟩	🟦	🟧	🟥	🟦	🟧
schedule	P1		P1	P2	P1	P3	P2	P1	P3	P2	P1	P4	P2	P1
FIFO			P2	P1 P3	P3 P2	P2 P1	P1 P3	P3 P2	P2 P1	P1 P4	P4 P2	P2 P1	P1	

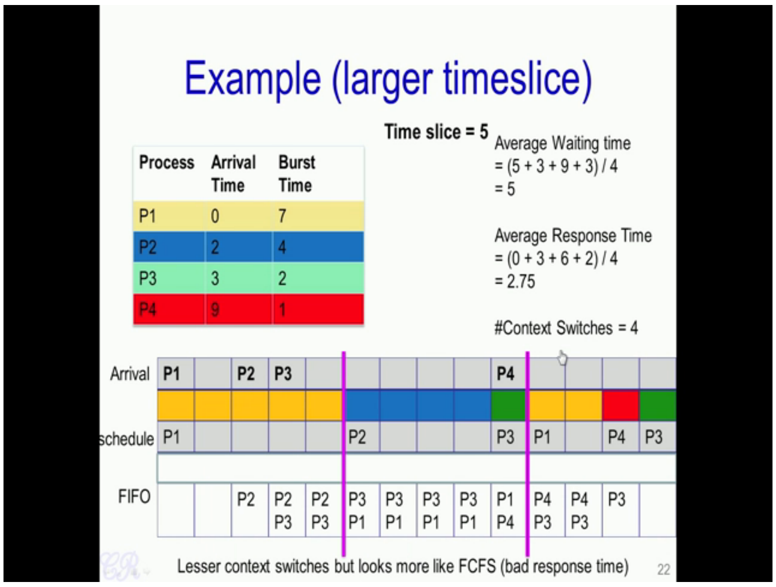
More context switches but quicker response times

21

Now let us take the same example as we have done before, but with a time slice of one that is we have reduced the time slice from 2 and we have made it 1. So, we see that in every instant we have a timer interrupt and potentially context switch that can occur. If we compute the average waiting time and average response time for this particular case we see that the average response time in particular has reduced. Why has this occurred is because once a process has entered into the queue it has to wait for lesser cycle before it gets scheduled into the CPU.

On the other hand, the number of context switch has increased from 7 to 11, since we are having timer interrupts which are more frequent therefore, there is more lightly that a context switch will occur.

(Refer Slide Time: 35:10)



Example (larger timeslice)

Time slice = 5

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	3	2
P4	9	1

Average Waiting time
= (5 + 3 + 9 + 3) / 4
= 5

Average Response Time
= (0 + 3 + 6 + 2) / 4
= 2.75

#Context Switches = 4

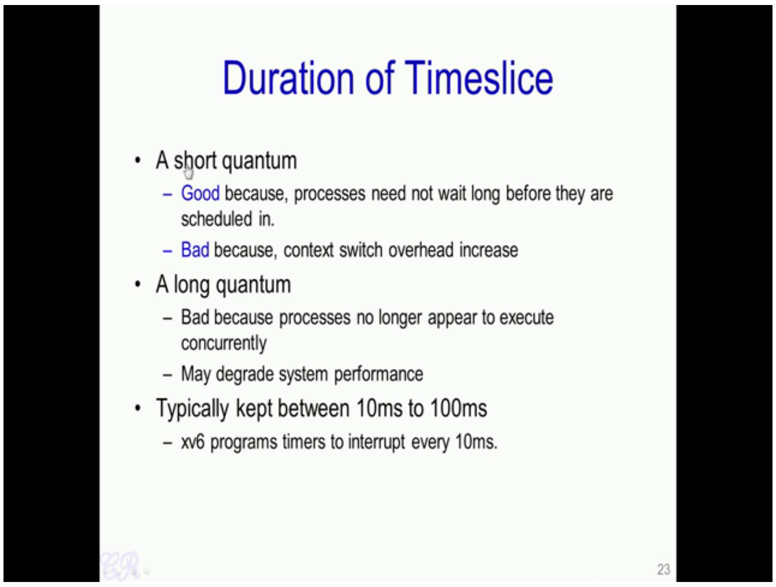
Arrival	P1		P2	P3						P4				
	🟨	🟨	🟨	🟨	🟨	🟦	🟦	🟦	🟦	🟩	🟨	🟨	🟥	🟩
schedule	P1					P2				P3	P1		P4	P3
FIFO			P2	P2 P3	P2 P3	P3 P1	P3 P1	P3 P1	P3 P1	P1 P4	P4 P3	P4 P3	P3	

Lesser context switches but looks more like FCFS (bad response time)

22

Now, if we take the same example, but with the time slice of 5 instead of 1, and if we compute the average waiting time and average response time, we see that the response time increases considerably 2.75 while the number of context switches reduces quite a bit to 4. On the other hand, we see that the scheduling begins to behave more and more like the first come first serve. The response time is bad because due to the large time slice the scheduling behaves more and more like the first come first serve which we know has a bad response time. So, from all this examples that we have seen so far we seen we can conclude that the duration of a time slice is very critical. So, it effects both the response time as well as the number of context switches.

(Refer Slide Time: 36:07)



Duration of Timeslice

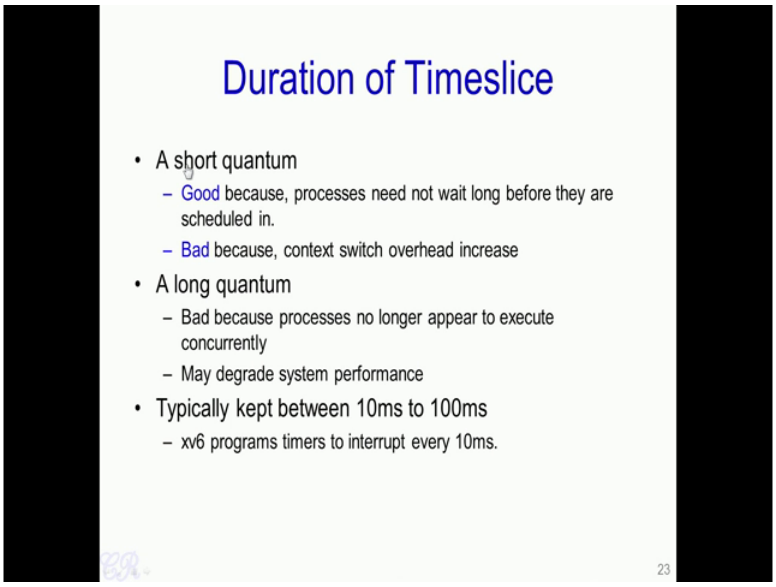
- A short quantum
  - Good because, processes need not wait long before they are scheduled in.
  - Bad because, context switch overhead increase
- A long quantum
  - Bad because processes no longer appear to execute concurrently
  - May degrade system performance
- Typically kept between 10ms to 100ms
  - xv6 programs timers to interrupt every 10ms.

23

So, essentially, if we have a time slice which is of a very short quantum, the advantage is that processes need not wait too long in the ready queue before they get schedule into the CPU. Essentially this means that the response time of the process would be very good or it would have a less response time.

On the other hand, having a short time slice is bad, because we would have very frequent context switches. And as we seen before context switches could have considerable over heads. Therefore, it degrades the performance of the system. A long time slice or a long quantum has a drawback that processes no longer appear to execute concurrently, it appears more like a first come first serve type of scheduling algorithm and so this again in turn may degrade system performance. So, typically in a modern day operating systems the time slice duration is kept anything from 10 milliseconds to 100 milliseconds. So, xv6 programs, programs timer to interrupt every 10 milliseconds.

(Refer Slide Time: 37:17)



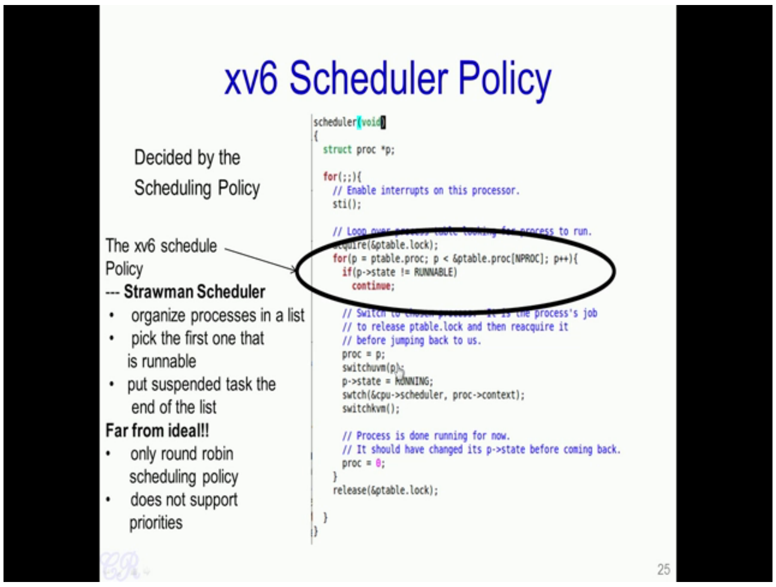
Duration of Timeslice

- A short quantum
  - Good because, processes need not wait long before they are scheduled in.
  - Bad because, context switch overhead increase
- A long quantum
  - Bad because processes no longer appear to execute concurrently
  - May degrade system performance
- Typically kept between 10ms to 100ms
  - xv6 programs timers to interrupt every 10ms.

23

The advantage of the round robin scheduling algorithm is as follows. The algorithm is fair because each process gets a fair chance to run on the CPU. The average wait time is low especially when the burst times vary. And the response time is very good. On the other hand, the drawbacks of the round robin scheduling algorithm are as follows; there is an increase number of context switching that occurs and as we have seen before context switching has considerable over heads. And the second drawback is that the average wait time is high especially when the burst times have equal lengths.

(Refer Slide Time: 37:56)



xv6 Scheduler Policy

Decided by the Scheduling Policy

The xv6 schedule Policy
--- Strawman Scheduler
- organize processes in a list
- pick the first one that is runnable
- put suspended task the end of the list

Far from ideal!!
- only round robin scheduling policy
- does not support priorities

```
scheduler(void)
{
  struct proc *p;

  for(;;){
    // Enable interrupts on this processor.
    sti();

    // Loop over process table looking for process to run.
    acquire(&ptable.lock);
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      if(p->state != RUNNABLE)
        continue;

      // Switch to chosen process.  It is the process's job
      // to release ptable.lock and then reacquire it
      // before jumping back to us.
      proc = p;
      switchuvm(p);
      p->state = RUNNING;
      swtch(&cpu->scheduler, proc->context);
      switchkvm();

      // Process is done running for now.
      // It should have changed its p->state before coming back.
      proc = 0;
    }
    release(&ptable.lock);

  }
}
```

25

The xv6 scheduling policy is a variant of the round robin scheduling policy; the source code is shown over here. So, essentially what the xv6 scheduler does is that it passes through the p table array. So, we have seen p table before which is an array of proc and the scheduler passes through this particular array and finds the next process that is runnable and invokes the switch. So, this is where you invoke the context switch. So, every time the scheduler executes, the next process in this array that is runnable gets scheduled to execute.

Next, we will see scheduling algorithms which are based on priority. So, these are the priority based scheduling algorithms.

Thank you.