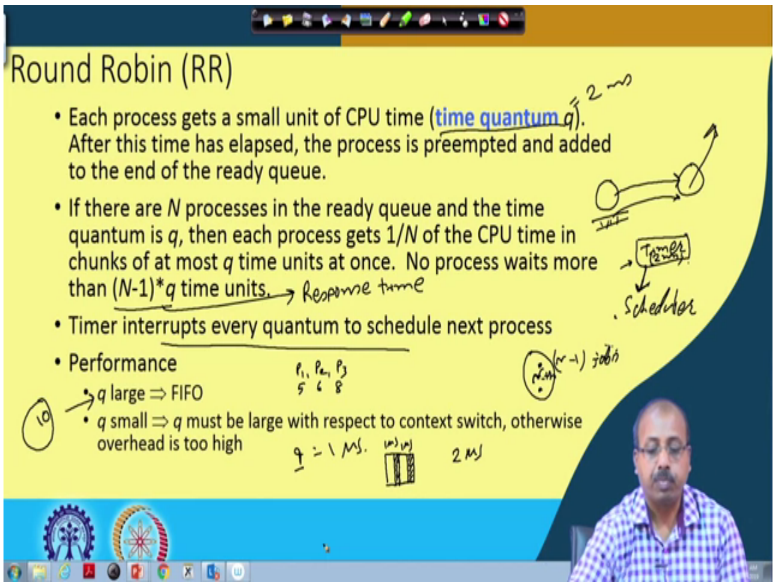
Operating System Fundamentals
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 27
Scheduling (Contd.)

So, after looking into this FCFS, SJF and the shortest remaining time first algorithms, so next algorithm that we have in the Scheduling policy that you can find in most of the computer systems is the round robin policy.

(Refer Slide Time: 00:37)



Round Robin (RR)

- Each process gets a small unit of CPU time (time quantum *q*). After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are *N* processes in the ready queue and the time quantum is *q*, then each process gets 1/*N* of the CPU time in chunks of at most *q* time units at once. No process waits more than (*N*-1)**q* time units.
- Timer interrupts every quantum to schedule next process
- Performance
  - *q* large ⇒ FIFO
  - *q* small ⇒ *q* must be large with respect to context switch, otherwise overhead is too high

So, as the name suggest, so there are a number of processes that we have in the system and each process is given some small unit of CPU time. So, this is known as the time quantum or this q. So, this is given for small time quantum. And so, this quantum value may be very small, so that is we will come to those value related issues later.

But suppose this time quantum value is equal to say 2 millisecond. So, what has what happens is that a job is given 2 millisecond for execution, so, after, so as soon as a job is put into the CPU. So, from the ready queue it is taken and it goes to the running state. So, simultaneously we start a timer. So, this timer when it expires, this timer is for 2 millisecond and when this timer expires. So, at the expiry of this timer the scheduler is invoked again. The scheduler is invoked. So, what the scheduler does is that it takes the job out of the CPU and it from the ready queue it picks of the next job and puts it into CPU.

So, that is how this whole thing operates. So, if there are N processes in the ready queue and the time quantum is q then each process gets 1 by N of CPU time in chance of at most q units at once. So, 1 by N of the CPU time, so that is a given to every process and once a process gets the time it is for q time units. And it can be shown that no process waits more than N minus 1 into q time units because if a job has to come if a job has arrived, so and there are N processes in the system then what can happen is that this is the N th process. So, before this fellow gets a chance for execution, so before this there are N minus 1 jobs, there are N minus 1 jobs before this each of them will get a q time units after that only this fellow will get a chance for execution.

So, the wait time for this N th job is equal to N minus 1 into q. So, after that it will get a chance for execution. Now, whether it will complete in the q time unit that it has got that is the question, but if the first time the response time that we have, so response time can be less because you can predict what is the response time. So, in the previous algorithms FCFS, SJF etcetera predicting response time was not possible because it depends on the CPU parts sizes of other jobs, but here whatever be the CPU, but size of the previous jobs so they all the previous jobs they are given only q time units.

So, total N minus 1 jobs at most, so that is N minus 1 into q that is the total time. So, this bound of N minus 1 into q, so this is a bound on the response time. So, it is so this response time is bounded by N minus 1 into q. So, timer quantum timer interrupts, so timer interrupts every quantum to schedule next process. As I was telling here that as soon as a job is put into the CPU the timer starts and the timer when it expires then it the scheduler is invoked and scheduler takes out; takes out the running process puts back puts it back into the ready q and it is the CPU is given to the next process.

Now, this time quantum follows the process gets the CPU it is not mandatory that the process will use its entire time quantum. So, it may so happen that the process is a highly IO bound job. So, though it is given 2 millisecond after executing for 1 millisecond it finds that I need to do some IO operation, so it just comes out of the CPU. So, that case also scheduler is invoked. So, before the timer expires the job gives control back to the scheduler, so that way also it may be possible that the scheduler is invoked and then again, the next job gets the chance to execute. So, whenever a job is selected by the scheduler the timer is started for 2 millisecond and that way it continues.

So, performance, if q is a very large value say the time quantum is very large then what happens is that once a process gets the CPU until and unless it voluntarily releases it. So, it will be taken back only after q time unit. Now, suppose in a system we have got jobs of this processes we have got 3 processes P 1, P 2 and P 3 and their CPU burst size sizes are 5, 6 and say 8. Now, if this q is equal to say 10, then what happens is that this jobs P 1, P 2, P 3. So, they will never face this timer going out.

So, they will always whenever it is getting the CPU for 10 time unit, P 1 will use for 5 time unit and then it will release it, P 2 will get it, P 2 will execute 6 time unit and release it. So, that way this timer interrupt will never come into picture. So, the situation is exactly same as what you have in the FIFO policy so, first in first out policy. Assuming that P 1 came to the system first followed by P 2 and P 3 the behaviour will be exactly same as what you have here. So, with a large this thing.

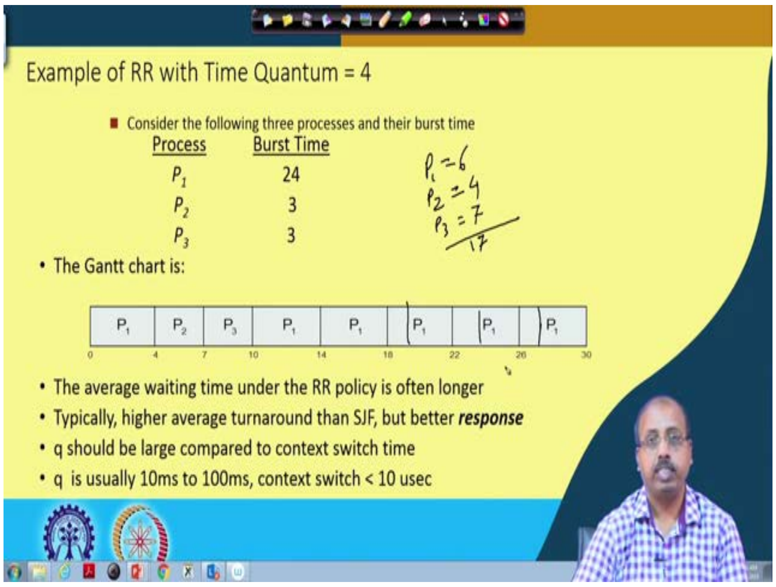
Now, if q is made very small ok. So, instead of making it say 10 millisecond, so if I make say q equal to say 1 microsecond which is very small then what happens is that the job gets the chance for execution it executes the 1 microsecond and then the scheduler has to be invoked again to select the next job. So, in this part the scheduler is running. Again, the next job is selected, so that runs for 1 micro second and then again the scheduler will come into picture.

So, whenever the scheduler comes into picture there is a context switch because the currently running process is taken out and the next process is started. So, you have to copy the contexts from PCB and all. So, that is a big thing. So, this context switching time becomes very high. So, q must be large with respect to context switch otherwise overhead is too high.

So, if the context switching time itself is around say 2 microsecond then if you keep this q to be 1 microsecond then you see for every microsecond execution of job you are wasting 2 microsecond in the context switch, so that way the CPU utilization will be this throughput and throughput will be very low, CPU utilization is also low as far as jobs are concerned and then you’re waiting time will be very high. So, this way the behaviour will become very poor for the system. So, at both ends making q very large making q very small, so both of them have got negative impact on the system performance, so, it is a it is an issue to find out what is a good q value for a system.

So, and again that is system dependent. So, there cannot be any general prediction like what should be the value of q. So, we can start with some value and after that what we can do we can look into the system behaviour and if we find that the system throughput is low or waiting time is high then we try to increase this q value a bit and see the effect.

(Refer Slide Time: 08:02)



Example of RR with Time Quantum = 4

- Consider the following three processes and their burst time

Process	Burst Time
P_1	24
P_2	3
P_3	3

$P_1 = 6$
$P_2 = 4$
$P_3 = 7$
17

- The Gantt chart is:

- The average waiting time under the RR policy is often longer
- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q is usually 10ms to 100ms, context switch < 10 usec

So, let us consider an example of this round robin policy with time quantum equal to 4. So, we have got 3 processes P 1, P 2 and P 3 and their time quantum values are 24 burst time values are 3 24, 3 and 3.

Now, if we take the time quantum equal to 4 then P 1 executes for 4 time units and since its burst time was 24, so the timer expired at this point. So, P 1 utilized full 4 time units and then the timer expired and now P 1 is taken out of CPU and P 2 gets a chance for execution. But P 2's burst time is 3, so after 3 time unit P 2 finishes, so now we have got P 3. Now, it is P 3 stands for execution, so P 3 executes for 3 time units and after that P 3 finishes now after that there is only one process in the system now P 1. So, P 1 gets successive time quantum's 10 to 14, 14 to 18, 18 to 22 like that. So, it finishes entire thing finishes in 30 time unit.

So, what is the average waiting time? So, average waiting time if you calculate then for P 1 the total waiting time is equal to the time it is waiting in the ready queue. So, for P 1 the waiting time is equal to 3 plus 3, 6 time unit, for P 2 the waiting time is equal to 4, and for P 3 waiting time equal to 7.

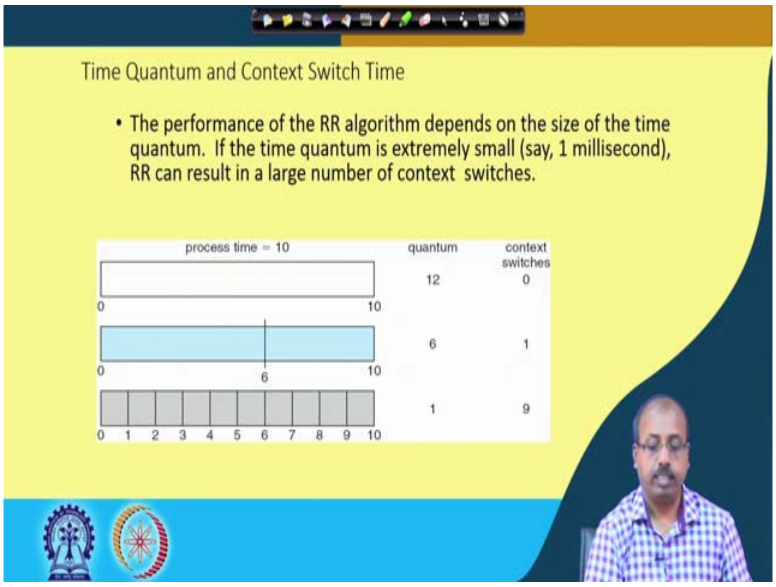
Of course, in this particular case we do not find much difference, it is a 10 plus 7, 17. So, it is like that HCFS policy only, but what can. So, if there are more processes such that, so this P 1 is not executing in successive time slots, so in between some other processes are also coming, so here in this region then P 1's time will be P 1's waiting time will increase further.

So, in general, average waiting timing of this round robin policy is larger than others, typically higher average turnaround and SJF, but better response times. So, average turnaround time compared to shortest job first year it will be more because there it has to all the jobs are getting catered too. So, it is not a given to a single job that we. Unlike, SJF the shortest job gets the immediate attention to its completion, so here that thing does not happen. So, every job is getting attention for some time equal to the time quantum and that in a round robin fashion the CPU is given to the jobs.

However, the response is better the response time is better because the job, but does not have to wait. Even if the we have been previously that if there are N jobs in the system in the response time is at most equal to N minus 1 into q, so that way it is good. Q should be large compared to context switching time, so compared to context switching time q should be large, so that we do not have the impact of this context switching time much otherwise there will be larger number of context switching if q value is small and each of those context switch, so that will take that context switching time, so that way it will be a problem.

Q is usually 10 millisecond to 100 millisecond for context switching time less than 10 microseconds. So, this is the range of values that we that in different systems you can find out. Of course, there is no hard and fast rule like it depends on the system that you have in your hand. So, normally it is a tunable parameter. So, context switching time is in the is less than 10 microsecond and this q value the time quantum value is between 10 and 100 milliseconds. So, round robin is good because this is for an interactive systems, so it is catering to all the processes, so that way it is good.

(Refer Slide Time: 12:04)


Time Quantum and Context Switch Time

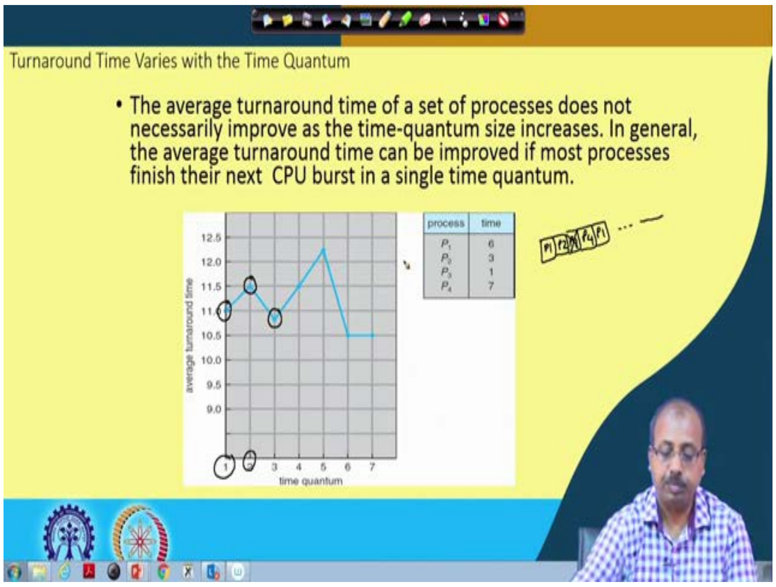
- The performance of the RR algorithm depends on the size of the time quantum. If the time quantum is extremely small (say, 1 millisecond), RR can result in a large number of context switches.

Now, this is a study on the quantum size. So, the performance of round robin algorithm depends on the size of the time quantum. If the algorithm the time quantum is extremely small say 1 millisecond then it can create a large number of context switches. Like here, so process time is say 10 and the time quantum time is time quantum size is 12 unit then it is a processor the CPU is given to the process for 12 time units, it finishes in 10 units. So, there are no context switchers, so context switch is 0.

However, if the time quantum value is equal to 6 then after 6 time unit there will be a context switch and then the process finishes at time 10. So, that way even if there is a single process then at after 6 time unit the timer will expire and the scheduler will come into picture scheduler will find that there is no other process to execute. So, it will continue with time 10, but for the execution of the scheduler itself there will be a context switch. So, as a result this context switching there is one context switching that will occur.

On the other hand, if we have got a time quantum size of 1. So, even if you have got a single process in the system. So, every one time you need that timer will expire and it will send an interrupt to the system and as a result scheduler will be invoked and this context switching will happen because of these between the scheduler and the process. So, there are 10, there are 9 such context switches that are occurring in this. So, you see that if you make this time quantum very small then the number of context switches will be very high and as a result a good amount of time will be wasted in setting this in executing the context switches.

(Refer Slide Time: 13:55)



Turnaround Time Varies with the Time Quantum

- The average turnaround time of a set of processes does not necessarily improve as the time-quantum size increases. In general, the average turnaround time can be improved if most processes finish their next CPU burst in a single time quantum.

process	time
P_1	6
P_2	3
P_3	1
P_4	7

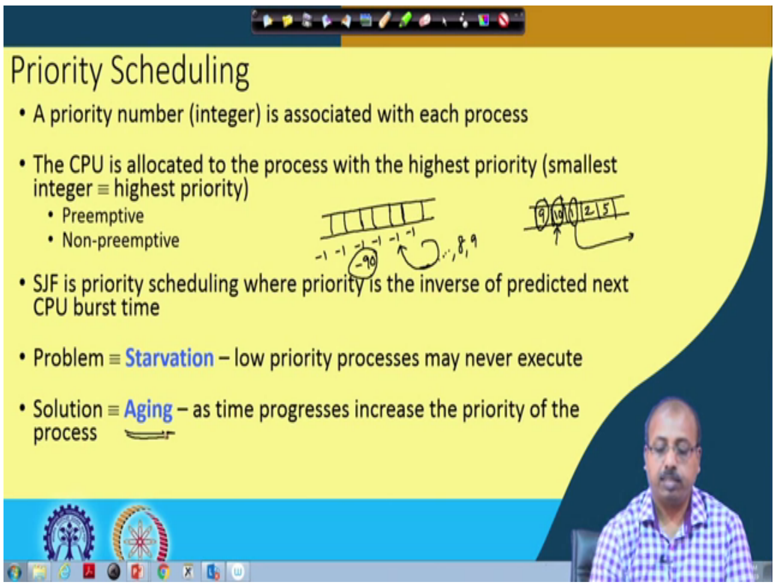
Now, how does this time quantum size turnaround time varies with the time quantum. So, this is a graph of an example. So, suppose I have got four processes P 1, P 2, P 3; P 1, P 2, P 3 and P 4 and their burst times are 6, 3, 1 and 7 and if you keep this time quantum value equal to one then the average turnaround time. So, if you, so it can be found out. So, if it is 1 then what will happen is that. So, you can calculate it like this. So, first P 1 will be executing for 1 time unit. So, 1 time unit, P 1 will execute. So, it is for. So, for 1 time unit P 1 will execute then after that P 2 will execute for 1 time unit then P 3 will execute 1 time unit ok. So, then, so P 3 will be over at this time, then after that P 4 will come for 1 time unit.

So, in this way; so, this now P 1 will execute for one more time. So, this way you can draw a Gantt chart and after that you can see that what is the response time average response time for the processes and it can be shown that if the time quantum size is said to be equal to 1, then this is the average waiting time, so this is 11.0. Whereas, if you take that this is equal to 2 then the average waiting time increases, so it becomes 11.5. Whereas, taking into 3 the average turnaround time, so that reduces. So, this is, so this turnaround time it does not necessarily improve as the time quantum size increases. So, in general the average turnaround time can be improved if most of the processes finish their next CPU burst in a single time quantum.

So, this is the thing that it says that it shows that as you are increasing the time quantum size. So, when you are going to this value of 6 for example, then you see that most of the processes they can finish within 1 time quantum and then the average waiting time starts to give good result. And if you set it to 7 then everybody can finish within 1 time quantum, so that gives us the minimum average waiting time. But that is close to FCFS type of algorithm.

So, that is, so this response time will be a problem, but turnaround time will be reduced. But it is not mandatory like when you are going from 6 to 5 this turnaround time increases significantly and again going from 5 to 4, 5 to 3 turnaround time reduces, 5 to 4 also turnaround time reduces. So, there is no generic behaviour of this graph, only thing is that if it is beyond the it if it is close to the burst size of individual processes then the average waiting time will average turnaround time will be minimized.

(Refer Slide Time: 17:15)



Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Non-preemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ Starvation – low priority processes may never execute
- Solution ≡ Aging – as time progresses increase the priority of the process

So, this way it gives us some guideline about how to select this time quantum size. So, but another type of scheduling algorithm that will look next is known as the priority scheduling.

So, priority is a very important issue because all the processes that we have in a system say they need not have the same priority. For example, if you have got a system where there are signals coming from sensors like say smoke detector and all. So, there is a fire that comes up that has to be attended immediately. So, that way that has got a larger priority or considered telephone exchange now all that calls that are going through the exchange, so they are not of same priority. So, calls which are of higher priority maybe the calls for which the tariffs are higher, so for them the priority should also be higher. So, they should not face much blocking.

So, for doing that we need to schedule the corresponding processes earlier than the processes which are of lower priority. So, normally in any process, in any operating system whenever we create a process we associate some priority with that. So, priority is generally represented by means of some integer number and the CPU is allocated to the process with the highest priority. So, as a convention we can say that the smallest integer value will correspond to the highest priority, but it is it just a convention, so, it maybe the reverse also. So, the smallest integer has got highest priority.

Now, the process when the scheduler is scanning the ready queue then whichever process has got the highest priority, so that will be scheduled next. Now, this priority scheduling policy, so it can be both preemptive and non-preemptive. So, preemptive, non-preemptive means at the time of taking a scheduling decision the scheduler looks into the priorities of the processes in the ready queue and based on that whichever process has the highest priority, so it is getting the CPU. And once it gets the CPU, so it completes its execution it completes entire CPU burst and after that only the scheduler is invoked again to see which process should be scheduled next based on priority. So, that is the non-preemptive type of a priority scheduling.

However, what can happen is that when a process is executing in the CPU, so there maybe a new process that has arrived and that newly arriving process so it may have higher priority than the process which is running currently in the CPU. So, as a result we need to give this process priorities. So, we need to pre-empt the CPU from the process which is currently running and give the CPU to the newly arrived process. So, that type of policy is known as the preemptive priority based policy. So, the CPU is allocated to the process with highest priority and it can be a preemptive scheduling or it can be a non-preemptive schedule.

So, SJF is one type of priority scheduling where priority is the inverse of predicted next CPU burst. So, priority has to be computed. So, in SJF you can say that, as if we are calculating priority by calculating their predicted next CPU burst and as you know that whichever process has got the least predicted CPU burst, so that gets chance for execution. So, here, so that is the priority is the inverse of this predicted next CPU burst.

This priority scheduling problem, so it has got difficulties one is called starvation because low priority processes may never execute. So, as I have already said is like this that if this ready queue that we have, so there I have got this processes. Now, this process they have got the priority, may be this process is a priority file. So, this is of 2, this is of 1, this is of 10, this is of 9 like that. So, as per when the scheduler will be invoked next since this has the least this has got the least value. So, this is of highest priority. So, this is getting this process will get chance for execution. After that this process with priority 2 will execute, then with 5, then with 9, then with 10 like that.

But what can happen is that there may be a large number of incoming processes there may be a continuous flow of higher priority processes that are coming to the system as a result the processes with priority values 9 or 10. So, they may not get chance for execution at all. So, over a long period of time may be the process came to the system quite early, but due to the flow of this higher priority processes. So, this process could not get a chance for execution. So, this is the starvation problem for this low priority processes.

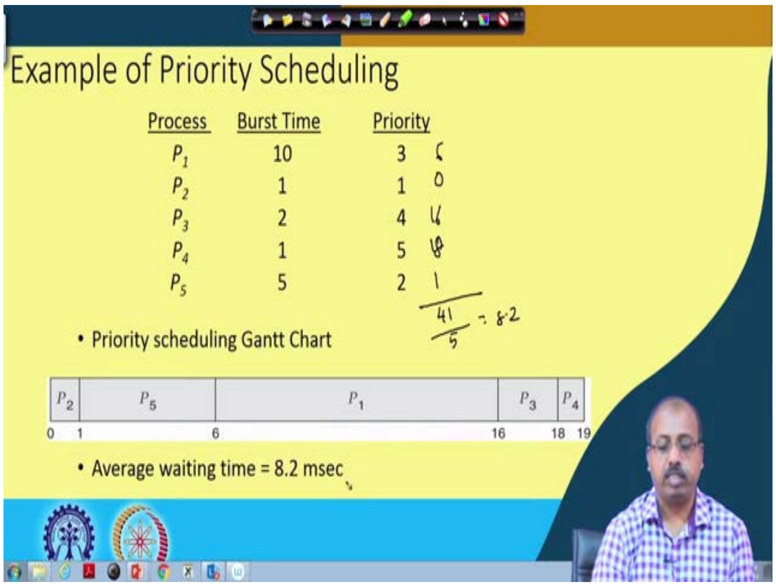
So, this is a very serious issue because one process has been submitted, but it is not getting chance for execution because of this high priority processes are continually coming. So, this is the problem of starvation. So, one solution to the starvation problem is the is aging. So, what we do is that as time progresses we increase the priority of the processes. So, every time we, so maybe I have got the ready queue, so in the ready queue we have got this processes ok. Then initial, so all this processes they have got different different priorities.

Now, what we do is that after some at some fixed intervals of time we scan this; we scan this ready list ready queue and we decrement all this priority values by 1. So, we decrement all this priority values by 1. Then what will happen? If a process is in the ready queue for a long time, so, this process like with the priority say 10. So, next time this cycle is done so this value will be reduced to 9, next time this cycle will be done so this will be reduced to 8. So, this way in successive in successive cycles of disparity check. So, this value will get reduced and after some time, so this value will become so low that even if a new process that is coming to the system they will not have that much low priority.

So, maybe after sometime after 100 iterations of this loop the value that we have here is say 9 say minus 90 for this priority, for this process. So, low process will possibly join the system with a priority value lesser than minus 90, so as a result this process will be the highest priority process now and next time the scheduler comes to select process from the ready queue this process is picked up for execution.

So, this aging, so this is known as aging. As the process resides in the ready queue for a longer time its priority is improved over the time, so that is the aging process. So, as time progresses increase the priority of the process. So, this solves the starvation problem definitely and that is often followed in many CPU scheduling algorithms that uses this priority based scheduling. Now, this is an example of this priority based scheduling.

(Refer Slide Time: 24:41)



Example of Priority Scheduling

Process	Burst Time	Priority	
P_1	10	3	6
P_2	1	1	0
P_3	2	4	16
P_4	1	5	18
P_5	5	2	1

$$\frac{41}{5} = 8.2$$

- Priority scheduling Gantt Chart

P_2	P_5	P_1	P_3	P_4
0–1	1–6	6–16	16–18	18–19

- Average waiting time = 8.2 msec

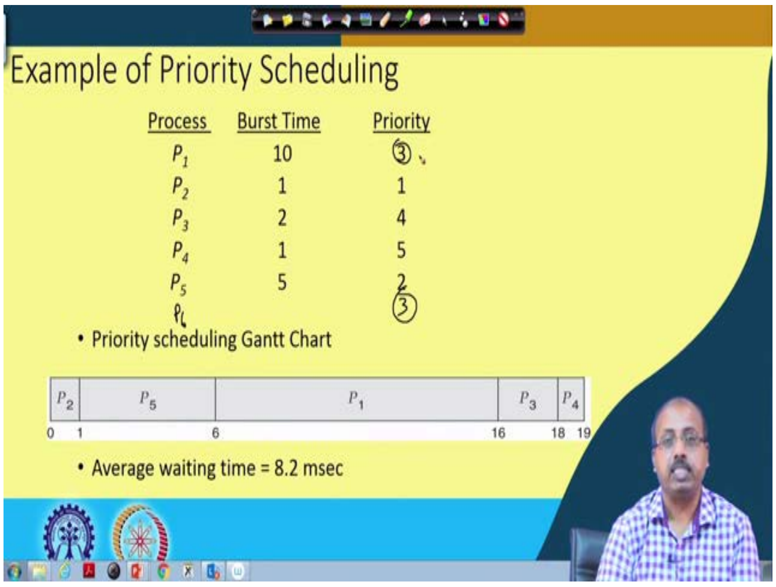
So, we have got five processes P 1, P 2, P 3, P 4, P 5 and their burst times are like this 10, 1, 2, 1, 5 and the priorities are 3, 1, 4, 5, 2. So, how this priorities were assigned, so that is not an issue here. So, that the priorities have been assigned by some technique, whatever be the strategy, they may be coming from different users, they may be due to aging and all. So, this is the current priority assignment.

Now, since P 2 has got the highest priority, so at time 0 P 2 is scheduled. So, P 2 executes for 1 time unit then next priority is to P 5. So, P 5 is a priority 2. So, P 5 gets a chance for execution, it executes for 5 time units. Then next is P 1, so P 1 executes for 10 time units then it is P 3. Then 3 has got priority 4, so it executes for 2 time units and then P 4 its priority is the last one, so that is it executes the 1 time unit.

So, average waiting time in this case is we can compute for P 2, the waiting time is for P 1 the waiting time is 6, for P 2 the waiting time is 0, for P 3 the waiting time is 16, for P 4 the waiting time is 18 and for P 5 the waiting time is 1. So, you can sum them up and then divide by number of processes. So, 16 plus 6, 22 plus 18 40, so that is 41. So, 41 divided by 5, so that is 8.2. So, this 8.2 is the average waiting time.

Now, this of course, so we cannot say anything in the sense that how this average waiting time will vary with the priority, burst time etcetera. So, that cannot be predicted because it is very much dependent on the job mix that we have ok. So, but this is a good policy because we can differentiate between the classes of processes that we have in a system, so that way it is a good one.

(Refer Slide Time: 26:54)



Example of Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2
P_6		3

- Priority scheduling Gantt Chart

P_2	P_5	P_1	P_3	P_4

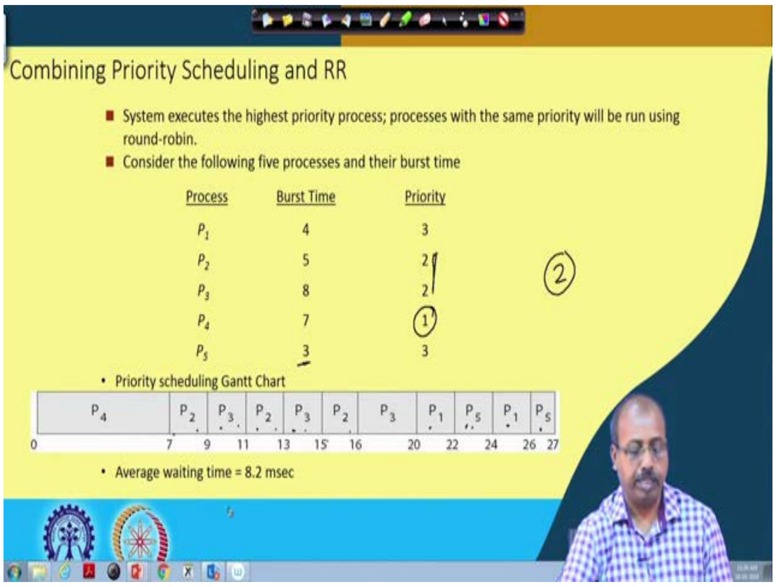
0 1 6 16 18 19

- Average waiting time = 8.2 msec

Now, you can club this priority scheduling with round robin. So, what we say is the system executes the highest priority processes and process is same priority will run using round robin. So, in the previous example that we had, so here all the priorities were different they were distinct. Now, what if there are two processes with the same priority. For example, here if there are two processes that of both of them are having priority 3. So, there is another process say P 6, whose priority is say. Suppose we have got another process P 6 whose priority is also 3. Now, between say P 1 and P 6 which one will be picked up, so that is an issue.

So, in this case it is said in the next policy that we are going to see what will happen is that P 1 and P 6 they will be picked up in a round robin fashion and that way they with both of them will have equal priority and they will execute in a round robin fashion.

(Refer Slide Time: 27:53)



Combining Priority Scheduling and RR

- System executes the highest priority process; processes with the same priority will be run using round-robin.
- Consider the following five processes and their burst time

Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

- Priority scheduling Gantt Chart

P_4	P_2	P_3	P_2	P_3	P_2	P_3	P_1	P_5	P_1	P_5
0–7	7–9	9–11	11–13	13–15	15–16	16–20	20–22	22–24	24–26	26–27

- Average waiting time = 8.2 msec

So, here we have got a situation where P 1 and P 5 they are having same priority, P 2 and P 3 they are also having same priority. So, first the; so, if for P 4 there is no confusion. So, P 4 it is priority is the highest. So, P 4 executes for 7 time units, sorry not this one. So, P 4 has the highest priority, so P 4 executes the 7 time units. After that we have got P 2, so P 2 P 2 and P 3 both of them have got same priority 2. So, between P 2 and P 3, so there will be round robin.

So, round robin size is the bar time quantum size is to taken to be equal to 2, so P 2 executes for 2 time units, then P 3 for 2 time units, then P 2 again for 2 time units P 3, then 2 time units. Then P 2, so P 2 has already executed 2 plus 2, 4 time units. So, P 2 will execute for 1 time unit now, so that is there is 1 time unit that P 2 executes. After that P 3, so P 3 is now is the only, P 3 is now the highest priority process and it has already executed for 4 time units, so now, it executes the remaining 4 time units.

After that there is a tie up priority between P 1 and P 5. So, between P 1 and P 5 again we have got this round robin. So, P 1 executes for 2 time units followed by P 5 for 2 time units and again P 1 for 2 time units. So, P 1 finishes at with 4 burst time. Now, P 5, P 5's burst time was 3, so it has executed for 2 time units here, so it executes for 1 time unit more at this point.

So, this way we can have this combination of this priority scheduling and round robin. So, they are taken together and then we get this type of scheduling. So, this is actually a compromise because it is difficult to maintain a large number of priority levels in a system. Like any operating system you look into, so there is a limit on the number of different priority levels that can be supported.

So, many times so even if we want that our the priority levels will be different. So, it is not possible to create processes in all different priority levels. So, we group the processes together into some priority levels, so that it is suppose there are only a 4 priority levels available in a system. So, if there are 100 jobs, so they will be distributed into this 4 priority levels only. So, within a priority level, so we follow round robin and between the priority level, so we have got this priority based scheduling.

And in this particular example you can find out that the average wait time is 8.2 millisecond and that is the result of this. So, you can, so depending upon the priorities and this burst time values, so this time will vary, so that is a compromise between priority and round robin ok.

So, continue with this in the next class.