

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

Lecture – 26
Scheduling (Contd.)

In our last class, we are looking into SJF policy. So, we start with an example of this SJF policy.

(Refer Slide Time: 00:32)

Example of SJF

■ Consider the following four processes and their burst time

Process	Arrival Time	Burst Time
P ₁	0.0	6
P ₂	2.0	8
P ₃	4.0	7
P ₄	5.0	3

Handwritten notes on the slide:
 - For P₁: $0 + 6 + 14 + 21 = 41$, Avg. Waiting = $\frac{41}{4}$
 - For P₂: $\frac{4}{4} = 1$
 - For P₃: $\frac{10}{1000}$
 - For P₄: $\frac{3}{24} = 0.125$
 - Overall Th: $\frac{4}{24} = \frac{1}{6} = 0.166$

■ SJF scheduling chart

■ Average waiting time = $(3 + 16 + 9 + 0) / 4 = 7$

Suppose, we have got four processes in a system at some point of time when the scheduler run is running, so it has to take a decision like which process will be executed next. And that their arrival times are like this 0, 2, 4 and 5 and their CPU burst times are 6, 8, 7 and 3. Now, SJF policy, so how do you know this burst time value? So, that is an issue, but definitely we are not considering that.

So, suppose you know these values, then as per as the SJF policy the whichever job has got the shortest burst time, so that will be selected. So, the jobs this is the smallest value. So, P 4 is schedule. So, P 4 runs from 0 to 3 time unit and from third time unit P 1 is the next job that we have whose burst time is just larger than P 4 and smaller than others. So, P 1 executives for 6 time units followed by P 3 for 7 time units and then P 2 for 8 time units. So, overall completion time is 24 for the set of jobs. So, throughput if you

compute, so we are completed 4 jobs in 4, 24 time units. So, throughput is equal to 4 by 24, so that is 1 upon 6, so 0.166, 0.166 that is the throughput; so, that is it.

Now, this average waiting time. So, this is basically this is average waiting time is equal to the this first job does job P 4, does not wait. So, the if waiting time is 0, P 1 is waiting for 3 time unit, so this is 3. P 2 is waiting for 16 time units, so that is 16 and P 4 is waiting for sorry P 3 is waiting for 9 time units, so that is 9. So, average waiting time is 3 plus 16 plus 9 plus 0 divided by 4, so that is equal to 7.

Now, if you try to see the effect of this set of job when they are scheduled on a FCFS first come first serve basis, then P 1 will be scheduled at the beginning, P 1 came first. So, P 1 gets scheduled from time 0 to time 6. So, here I have got P 1. Then from 6 then came P 2 and that requires 8 time units. So, this is 8, so, this is our power P 2. So, 6 to 8 that is 14. Then comes P 3 for 7, so this is P 3 for 7, so this is 21 and then P 4 for 3 time unit. So, P 4 came after that and P 4 for 3 time unit. So, this is 24, so that is P 4. So, you see that the set of tasks. So, this is completed at time 24 only. But under this FCFS policy so, if you compute this throughput. So, throughput remains same. So, throughput is equal to 4 by 24 that is 0.166.

However, if you try to see what is the average waiting time. So, for P 1 the a waiting time is 0 for P 2 the waiting time is 6, for P 3 waiting time is 14 and for P 4 waiting time is 21. So, 6, 14 plus 21, so 41, so, this is the total waiting time. So, average waiting time is 41 by average waiting time is 41 divided by 4, so that is 10 point 10.25. So, you see that the average waiting time for SJF policy was equal to 7 and for FCFS policy, so this is going to 10.25. So, this way if we are going for SJF policy we can see that this waiting time can reduce significantly and as there is a theoretical proof that says that you cannot beat this time 7, that is you cannot come up with any other scheduling of this P 1, P 2, P 3, P 4 such that the average waiting time become less than 7 ok. So, that is the idea.

So, we can so, but the difficulty with this SJF policy is that how do we know what is the next CPU burst time. So, the FCFS we did not have any problem because arrival time of the jobs are known, like when the job came to the system or if it is it if it has gone for an IO burst, so next when it joins the ready queue so that time is the that time can be found out from the system clock. So, we can find out which job join the queue at what time.

But this how much time the job will execute in the next CPU burst before going to an IO operation. So, this is not possible to tell accurately and there would be it also depends on the execution trace that we have in the program. So, for the programmer also it is not possible to tell the size of the CPU burst because that will depend on the actual execution sequence. Maybe for some data values it will be there doing a lot of computation and some other data values maybe it will be just bypass many of those computations.

So, as a result of this is it will depend on the value. Like say a typical example is the loop statements like if I have got a loop like for i equal to 1 to n, do. So, this, so how many times this loop will execute, so that depends on the value of n. So, if in one CPU burst may be the n value is was equal to 10, so it executed for sometime without doing IO operation. In the CPU burst may be n value is equal to 1000. So, naturally the CPU burst sizes are not comparable.

That is, so if the for the user also it is not possible to tell. So, we have to, so we cannot implement this SJF directly, but SJF being a very good policy we can try to do some approximation of that. So, next we will see and approximate technique that will try to be close to SJF until that will try to predict the behaviour of the process ok.

(Refer Slide Time: 07:16)

Determining Length of Next CPU Burst

- Can only estimate (predict) the length – in most cases should be similar to the previous CPU burst
 - Pick the process with shortest predicted next CPU burst
- Can be done by using the length of previous CPU bursts, using exponential averaging
 1. t_n = actual length of n^{th} CPU burst
 2. r_{n+1} = predicted value for the next CPU burst
 3. $\alpha, 0 \leq \alpha \leq 1$
 4. Define : $r_{n+1} = \alpha t_n + (1 - \alpha) r_n$
- Commonly, α set to $\frac{1}{2}$

Scientific

not known -> Dnni

So, that it is known as the shortest, it will try to predict the next CPU burst size. So, what is done here is that we try to estimate or predict the length ok. So, it can be observed that if I have got a program which is doing some computation, some of some scientific

computation, then it is very much likely that if in one time if in one CPU burst, so it has used say 20 time units then in the next CPU burst after that it has done a small amount of IO and after that it is going to another CPU burst. So, this CPU burst time cannot be largely different from 20. So, it cannot be that the next CPU burst is very small or the next burst is la much much larger than the original than the previous CPU burst.

This happens because the whole the CPU bursts that we are considering. So, they are part of the same process and they are doing similar computations. For example, maybe it is getting some data and based on that it is doing set of computations. So, it is very much unlikely that this computation duration will vary significantly. Of course, there may be some data values for which it will change, but in general it will not happen.

So, this how to predict? So, we do a prediction ok. So, based on this 20 value we try to predict what is the next value or to be in a to be on a safer side what we do is that we considered last few CPU burst. So, you take, so this is from the beginning of the program maybe this is the size of the CPU burst one after that we had some IO burst then we have got another CPU burst c 2 of some duration then we have got some IO burst. So, this way it goes on.

So, previously, so in this way if you have got up to this CPU burst size c n than the IO burst and then if you are trying to predict like what is going to be the size of this CPU burst. So, we take the history into consideration that in recent time. So, it has generated the CPU burst of size c 1, c 2 and up to c n. So, based on those values we try to predict the CPU burst size for the c n plus 1. So, that is the prediction.

So, what we do is suppose t_n is the actual length of the nth CPU burst. So, this nth CPU burst if this duration was equal to t_n , the size of this burst was equal to t_n . Then τ_{n+1} is the predicted value of the next CPU burst. So, this is our prediction. So, this is not actual 1. So, until unless this CPU, the CPU is given to the process for execution. So, we cannot know what the value of t_{n+1} is plus 1, so, this is not known.

This is not known, but the previous value t_n , t_n is known because when the CPU was given to the process last time how long did it use it. So, that value is the t_n value. So, that value is known, but t_{n+1} is not known. So, you what we try to do it is that we try to predict the value of τ_{n+1} . So, we try to predict the size of the next CPU burst.

So, we say that τ_{n+1} is basically $\alpha t_{n+1} + (1-\alpha)\tau_n$, but τ_n is basically the prediction that we had at this point. So, for this burst, so, what was supposed this burst was predicted like this that was the value of τ_n and it actually executed for this t_n . So, for this n th CPU burst the prediction was τ_n and its actual execution was t_n . So, we try to take an average of these two. So, basically, so this is $\alpha t_{n+1} + (1-\alpha)\tau_n$.

(Refer Slide Time: 11:40)

Determining Length of Next CPU Burst

- Can only estimate (predict) the length – in most cases should be similar to the previous CPU burst
 - Pick the process with shortest predicted next CPU burst
- Can be done by using the length of previous CPU bursts, using exponential averaging
 1. t_n = actual length of n^{th} CPU burst
 2. τ_{n+1} = predicted value for the next CPU burst
 3. $\alpha, 0 \leq \alpha \leq 1$
 4. Define : $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$
- Commonly, α set to $\frac{1}{2}$

Handwritten notes on the slide include:

- Diagram showing three boxes representing CPU bursts: t_{n-1} , t_n , and t_{n+1} .
- Equation: $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n = \alpha t_n + (1-\alpha)[\alpha t_{n-1} + (1-\alpha)\tau_{n-1}]$
- Equation: $= \alpha t_n + \alpha(1-\alpha)t_{n-1} + (1-\alpha)^2\tau_{n-1}$
- Equation: $= \alpha t_n + \alpha(1-\alpha)t_{n-1} + \alpha(1-\alpha)^2 t_{n-2} + \dots + \tau_1$

So, if you expand this expression then if you expand this expression then you can see, so this is $\alpha t_{n+1} + (1-\alpha)\tau_n$ into $\alpha t_{n+1} + (1-\alpha)[\alpha t_n + (1-\alpha)\tau_{n-1}]$. So, this is $\alpha t_{n+1} + \alpha(1-\alpha)t_n + (1-\alpha)^2\tau_{n-1}$. So, again this τ_{n-1} can be expanded.

So, essentially what I am getting is an expression like this $\alpha t_{n+1} + \alpha(1-\alpha)t_n + \alpha(1-\alpha)^2 t_{n-1} + \dots + \tau_1$. So, that way it goes on going up to t_1 .

So, you see in this expression of τ_{n+1} . So, we have considering the actual CPU burst over last n cases and based on that we are trying to see like what is going to be the next CPU burst. So, but their weights are decreasing. So, the recent burst, so it has got the maximum effect. So, this α is a value between 0 and 1, so this $1-\alpha$ is

also a value between 0 and 1. So, as you are going away from the current CPU burst, so the weightage of this weight age of that CPU burst is decreasing.

But so, whenever you are trying to predict the size of this n plus 1th CPU burst then we are taking we are giving this t n the maximum weighted that is alpha time t n, but the previous to that the t n minus 1, so that is given less weightage than t n because it is alpha into 1 minus alpha into t n minus 1. So, this weightage is less. So, as you are going back and back, so this weightage is reducing weightage on that burst size is reducing.

So, this way we try to make a prediction. And this whole part, so this whole part is captured by this term 1 minus alpha into tau n. So, what was the prediction at the nth burst. So, this way we can have some prediction for the n plus 1th burst of the CPU and then n plus 1th CPU burst for the process and accordingly we can take a decision.

(Refer Slide Time: 14:40)

Determining Length of Next CPU Burst

- Can only estimate (predict) the length – in most cases should be similar to the previous CPU burst
 - Pick the process with shortest predicted next CPU burst
- Can be done by using the length of previous CPU bursts, using exponential averaging
 1. t_n = actual length of n^{th} CPU burst
 2. τ_{n+1} = predicted value for the next CPU burst
 3. $\alpha, 0 \leq \alpha \leq 1$
 4. Define : $\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n$.
- Commonly, α set to $\frac{1}{2}$

Handwritten notes on the right side of the slide:

- $\alpha = 0$
- $\tau_{n+1} = \tau_n$
- $\alpha = 1$
- $\tau_{n+1} = t_n$

Handwritten mapping on the right side of the slide:

- $P_1 \rightarrow \tau_{1,n+1}$
- $P_2 \rightarrow \tau_{2,n+1}$
- $P_3 \rightarrow \tau_{3,n+1}$

So, if we have got up if you have got say processes say P 1, P 2 and P 3 in the ready queue, so for each of them we know the their tau value. So, tau 1 n plus 1. So, this is we have got the prediction for this one also tau 2 n plus 1 and this one also tau 3 n plus 1. And we use this tau values to take a decision like which process should be scheduled next.

Now, this value alpha that we have is between 0 and 1 and you can understand that this alpha value. So, if alpha is said to be equal to 0, if alpha is set equal to 0 then tau n plus 1

is equal to τ_n ok. So, whatever you have as the as the initial prediction, so that continuous. So, this history does not have any effect on the CPU, on the prediction of the next CPU burst.


Whereas, if you take α to be equal to 1 then this part does not have any effect and we have got this τ_{n+1} is equal to t_n . So, the next prediction is made to be equal to the actual CPU burst time that we had in the previous burst. So, that is t_n . So, it is just considering history of one previous burst. So, it is not considering all of them, it is just taking one burst and taking it as the prediction. So, τ_{n+1} equal to t_n . So, we have got; so these are the two extremes. So, ideally, we should be somewhere in between, so α is normally set to half or 0.5 and that have puts equal weightage on the previous frames previous CPU burst size and the pervious CPU burst prediction.

And the previous CPU burst prediction in its expansion we have seen that it has got effect of all the previous CPU burst times in actual. So, this way this the shortest job first algorithm, so it can be of approximated and this is very common. So, whenever a job is executing, so we note down how much was the CPU burst, how much time it executed in the CPU. So, as to have seen that, with the CPU maybe after the executive for two second it goes into an IO operation in that case t_n becomes equal 2 or in some other case maybe after executive for 5 seconds the job into IO burst. So, t_n value becomes equal to 5. So, this way based on those t_n value this τ_{n+1} can be predicted and based on the prediction the scheduler can take a decision like which job to be schedule next.

(Refer Slide Time: 17:30)

Examples of Exponential Averaging

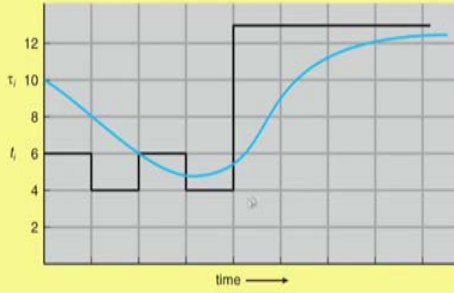
- $\alpha = 0$
 - $\tau_{n+1} = \tau_n$
 - Recent history does not count
- $\alpha = 1$
 - $\tau_{n+1} = \alpha t_n$
 - Only the actual last CPU burst counts
- If we expand the formula, we get:
$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\alpha t_{n-1} + \dots + (1 - \alpha)^j \alpha t_{n-j} + \dots + (1 - \alpha)^{n+1} \tau_0$$
- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor




So, that is the approximation to the shortest job first. This is alpha equal to 0. So, tau n plus 1 equal to tau n. So, recent history does not count. The other hand we have got alpha equal to 1, so tau n plus 1 equal to alpha times t n. So, only the actual CPU burst accounts. And as we expand it, so this is this expansion we have already seen and successive terms they will have less weight than its predecessor. So, this slide actually summarizes to what we have discussed previously in terms of this weight of these individual CPU burst.

(Refer Slide Time: 18:03)

Prediction of the Length of the Next CPU Burst



CPU burst (t_i)	6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12



swayam

So, suppose this is the actual CPU burst sizes. So, this blue line is actual CPU burst sizes and this these are the predictions. So, this is at time; so, (Refer Time: 18:20), the first CPU burst guess is. So, this is sorry. This the blue line is the tau values the predicted values and t value is the actual CPU burst. So, this is the sizes. So, initial predictions that is arbitrary set to be equal to 10 and you see the actual time it to is 6. So, the actual time it to is 6. So, this as a result the next prediction comes down to 8 ok. So, it is predicted to be 8 and so it comes here, but that time the CPU; the CPU burst was actually equal to 4.

So, next time the prediction comes down to 6 and this actual execution is also 6, so it was fine. So, prediction remains at 6 only, but then the execution is 4. So, again the prediction comes down to 5. Now, the actual execution is jumped, so it to the prediction was 5 that is here, but it is used 13 time units in a CPU burst was 13 time unit then a prediction will go up. And then the next the prediction comes to be 9 ok. So, it increases to this value, so, these becomes equal to 9.

Now, for the next part also the CPU burst is the high is 13 only, so these it continues to be 13 and the prediction again goes up using that formula. So, this is equal to um eleven it is equal to 11. Then it goes to a prediction of 12 and if it goes like this then you can see that slowly it will catch up these value 13 ok. So, this prediction that we are doing, so prediction is trying to follow the trained of the process. So, if the trained, so here it perform miserably because there was a certain change sudden change in the behaviour of the process.

So, previously the CPU burst values were equal to 6, 4, 6 and 4 and all on the sudden it has become 13. So, this is the change in the behavior of the process. So, our prediction could not follow it. But after that the process has continued to behaving like this CPU burst size to be equal to 13 and this prediction values is slowly catching up that value. So, again, so now, the next prediction happens to be equal to 12 now if all on the sudden the process drops its few burst size.

So, process starts conservative becoming more of IO bound job, then again, the prediction has to come down and it will take quite a few times steps to again catch up with the actual time. So, that is there. So, but it is a good prediction. So, we can see that the good approach for doing the approximation via the prediction.

(Refer Slide Time: 21:12)

Shortest-remaining-time-first

- Preemptive version of SJF is called **shortest-remaining-time-first**
- Example illustrating the concepts of varying arrival times and preemption.

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

• Preemptive SJF Gantt Chart

P_1	P_2	P_4	P_1	P_3
-------	-------	-------	-------	-------

• Average waiting time = $\frac{[(10-1)+(1-1)+(17-2)+5-3]}{4}$
 $= 26/4 = 6.5 \text{ msec}$

Diagram illustrating preemption: $P_i \rightarrow 15$ and $P_j \rightarrow 5$. A circle around 15 has an arrow pointing to 5, and another arrow points from 5 to 10.

Small video inset of a speaker in the bottom right corner.

So, another algorithm which is known as shortest remaining time first or SRTN or SRT or sometime it is called SRTN. So, instead of first we call it shortest remaining time next also. So, then the algorithm will be called SRTN algorithm or SRTF algorithm.

So, this is a preemptive version of the SJF algorithm. So, shortest job first it tells that whichever job is has got shortest CPU burst. So, that will be getting first. So, if now a job comes and whose burst time is less than whichever job is currently running. Then naturally suppose a some job P_i was some job P_i was running and P_i 's CPU burst predicted CPU burst was say 15 units, so this tau value is equal to 15. So, it has been selected and it is running in the CPU.

Now, a new job P_j comes whose CPU burst who predicted CPU burst is less, suppose this is equal to 5. Then if this P_i is executing and P_i might out of this 15 maybe it has run for say 5 time unit. So, it needs to run for another time unit to finish this 15, but these job P_j has come, so which is definitely shorter than this value 10. So, it is advisable that we take out P_i and give P_j the CPU. So, that is exactly the concept that we have. So, we have got if we vary the arrival times then apply this preemption policy then we can have this things. Suppose, you have got four processes P_1, P_2, P_3, P_4 their arrival times are 0, 1, 2, 3 and the burst times are 8, 4, 9 and 5.

Now, at time, so at initially this P_1 came. So, when this scheduler was there. So, at time we had only P_1 , so P_1 came, so P_1 got scheduled. So, P_1 executed, so after that P_2

came at time 1. So, when, so at time 0 only P 1 was there in the system, so scheduler could not have any choice. So, it scheduled P 1 then after sometime after 1 time unit P 2 has come, when P 2 has come, so it to the system sees that, the job that is running now is going to take another 7 time units and P 2 is of burst time 4. So, P 1 is preempted and P 2 gets the chance for execution. So, P 1 has executed for 1 time units, so it has got 7 units less left.

(Refer Slide Time: 23:53)

Shortest-remaining-time-first

- Preemptive version of SJF is called **shortest-remaining-time-first**
- Example illustrating the concepts of varying arrival times and preemption.

Process	Arrival Time	Burst Time
P ₁	0	8 (7)
P ₂	1	4 (2)
P ₃	2	9 (1)
P ₄	3	5 (1)

- Preemptive SJF Gantt Chart

A Gantt chart illustrating the execution of processes P₁, P₂, P₃, and P₄ under a preemptive Shortest Remaining Time First (SRTF) scheduling policy. The time axis ranges from 0 to 26. P₁ starts at time 0 and runs until time 1. At time 1, P₂ arrives and preempts P₁. P₂ runs until time 5. At time 2, P₃ arrives but does not preempt P₂ because its remaining time (9) is greater than P₂'s remaining time (3). At time 3, P₄ arrives and preempts P₂ because its remaining time (5) is less than P₂'s remaining time (3). P₄ runs until time 10. At time 10, P₁ resumes execution and runs until time 17. At time 17, P₃ resumes execution and runs until time 26.

- Average waiting time = $\frac{[(10-1)+(1-1)+(17-2)+5-3]}{4}$
- = $26/4 = 6.5$ msec

Now, P 2 comes, so P 2 needs time for 4 time unit, but a time 2, at time 2, P 3 comes. So, P 3 when it comes, so P 3 its burst time is 9, but this is this has already executed for 1 time unit, so the remaining time is 3, but it cannot. So, this this P 2 continues to be the shortest job. So, this P 2 continuous. Then after sometime after 1 time unit P 4 comes, but here also the P 2. So, this P 3 is waiting and P 2 has executed for 1 more time unit. So, remaining time is 2. But P 5, P 4 is of requiring 5 time unit, so they are not schedule. So, P 2 continuous what 4 time units and it completes the time 5. So, this is time 0, this is time 1, so this of to time 5 it executes.

Now, we have got two processes 3 process in the system P 2 sorry P 1, P 3 and P 4 and out of that P 4 has the smallest remaining; smallest remaining time. So, P 4 gets scheduled, so P 4 executes. And no further job comes. So, form this point onwards it is behavior is similar to SJF. So, P 4 executes for 5 time unit, so it completes at time 10. Then between P 1 and P in the P 3, so P 1 had 7 time units left and P 3 had 9 time units

left. So, P 1 got scheduled, so it executed till time 17 and then P 3 got schedule. So, P 3 executed till time 26.

Now, the point is that this is a preemptive version. So, this whenever a new job comes the scheduler will be invoked to check whether it needs to that preempt the job from the CPU, the currently running job from the CPU and schedule the new one. Now, this is good, but at the same time you see that it is a bit overhead is slightly more because every time in new job comes to the ready queue the schedule has to be involved and though we is we tell that at this time, so P 1 executes from time 0 to 1 and then P 2 executes form 1 to 5. But what will happen is that in between the scheduler will be invoked.

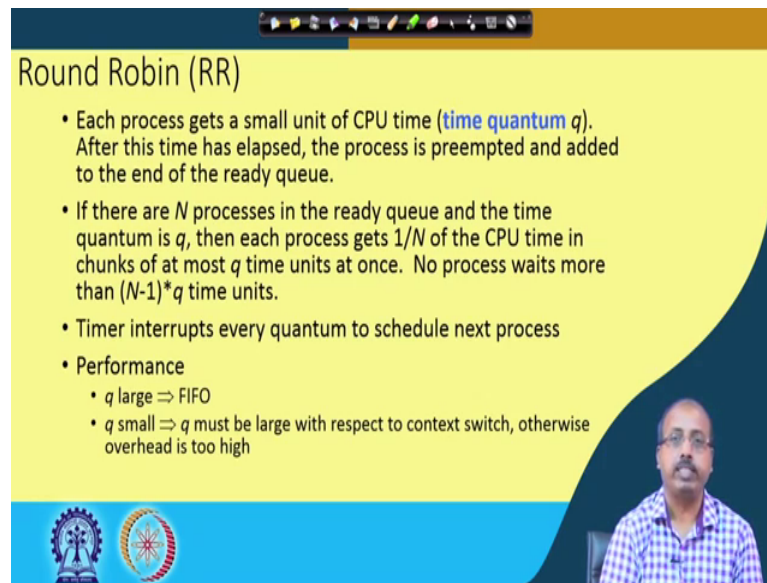
So, there is there sometime will be spent in invoking the schedule burst. Then again at time 2; at time 2 P 3 has arrived. So, at time two again the scheduler will be invoked to see whether it needs to do some switchover. Similarly, at time 3 the scheduler will be invoked to see if it needs to do any switchover. So, those overheads are there, so though it is not shown in this diagram, so those overheads are there. So, as a result; so, this is the theoretical behavior, but in the actual behavior, so those are timing of are there are execution time for the schedulers are to be taken into consideration and as a result the duration will be more.

So, this shortest remaining time first. So, this is a good a algorithm, because it is also taking care of the job which is running in the system, but of course, with the catch that the it is quite complex strategy as for as execution implementation is concerned.

So, this out of the algorithm that we have seen. So, for FCFS implementation is very easy because I can have a FIFO data structure like q type of data structure and then in that q whenever a new job comes it joins at the end that tale end and whenever a job has to be taken for execution, so it is taken from the front end. So, we do not really need to remember the arrival times also with the jobs, so if we follow the policy it is very simple. But the remaining policy, so it is a slightly difficult because we have to take care of this other information like their burst time their priorities and all. So, that makes it difficult.

So, even for this SRTN algorithm so, we have to take care of this policies. So, in this way you; so, we can have different scheduling policies. Now, we can; we can; we can have some other policies which is one of them is a round robin policy.

(Refer Slide Time: 28:17)



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 \Rightarrow FIFO
 - q small \Rightarrow q must be large with respect to context switch, otherwise overhead is too high

The slide features a yellow background with a dark blue curved shape on the right side. At the bottom left, there are two circular logos. A small video inset in the bottom right corner shows a man with glasses and a mustache, wearing a blue and white checkered shirt, speaking.

So, this round robin policy we have got some small time quantum and the job is executed for some time quantum q and when the time quantum expires then the job is taken back from the CPU when the next job gets a time for execution.

So, will see that this type of policy. So, this is if you have got an interactive system or a multi-user system where each job should get attention for execution for sometime. So, this is good for an interactive system, but at the same time will see that the amount of time wasted is the maximum here and the weight time and everything will go of for the process.

So, will see that in the next class.