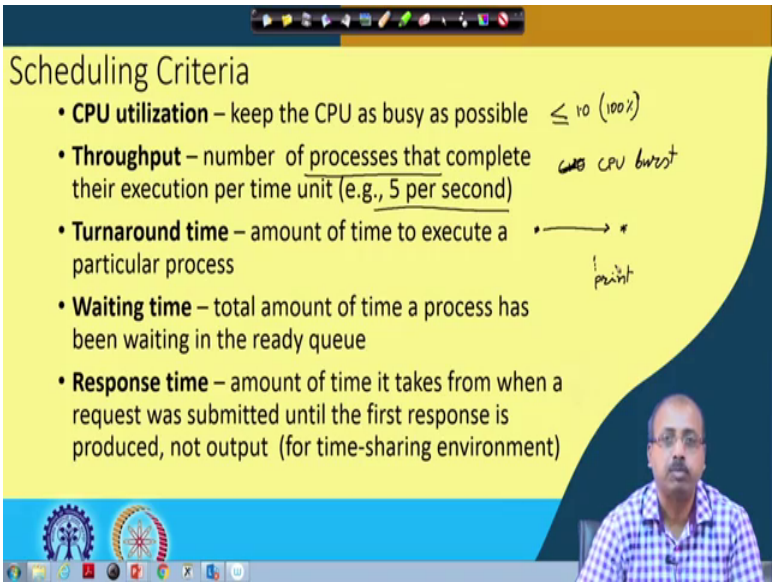
Operating System Fundamentals
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Science, Kharagpur

Lecture – 25
Scheduling (Contd.)

In the last class we were discussing about Scheduling algorithms and there we have seen that there are certain criteria that any scheduling policy will try to optimize.

(Refer Slide Time: 00:35)



And those criteria that we have a consists of a few number of them and it is not mandatory that all of them can be optimized by a single scheduling policies so, different scheduling policies may target different criteria to be optimized. So, to summarize the scheduling criteria so, one is the CPU utilization. So, naturally objective is to have the CPU utilization equal to 100 percent; definitely we cannot have this CPU utilization more than 100 percent. Because, in that case it means that CPU is we are trying to keep it busy for more than 100 percent of the time.

But at one point of time CPU can do only a single job so, naturally more than 100 percent utilization is not possible. So, the CPU utilization will be less or equal 100 percent or 1.0 or 100 percent. So, naturally will try to be as close or as close as possible to the value 1, then comes throughput. So, throughput is the number of process that completes their execution per unit time. Now, you see that throughput is also consisting

of other factors like if a job is waiting for something. So, that way so, we if we are counting in terms of number of processes completed. So, we will be computing number of CPU burst that are completed per unit time.

So, each CPU burst will be considered as one process for the sake of calculation. So, we can say that how what is the throughput so, that is one possibility or maybe if we are counting in terms of number of jobs completed per unit time so, that is also possible. So, we can see like number of processes completed or number of CPU burst completed per unit time. So, that is how many jobs are completed per unit time. Then the turnaround time so, it the amount of time to execute a particular process. So, this is from the submission of the job till the job completes. So, this is from the new state to the terminated state or completed state.

So, what is the total time needed and that is the clock time; so that is not the CPU times that is a clock time. So, what is the total time needed for a particular process that is the turnaround time for the process. Waiting time as we have seen so, it is the amount of time a process is made to wait in the ready queue. So, during its entire lifetime in the system so, process is kept in the ready queue for quite some time depending upon the number of jobs that we have in the system. So, availability of the CPU is an issue so, that job may have to wait in the ready queue for quite some time.

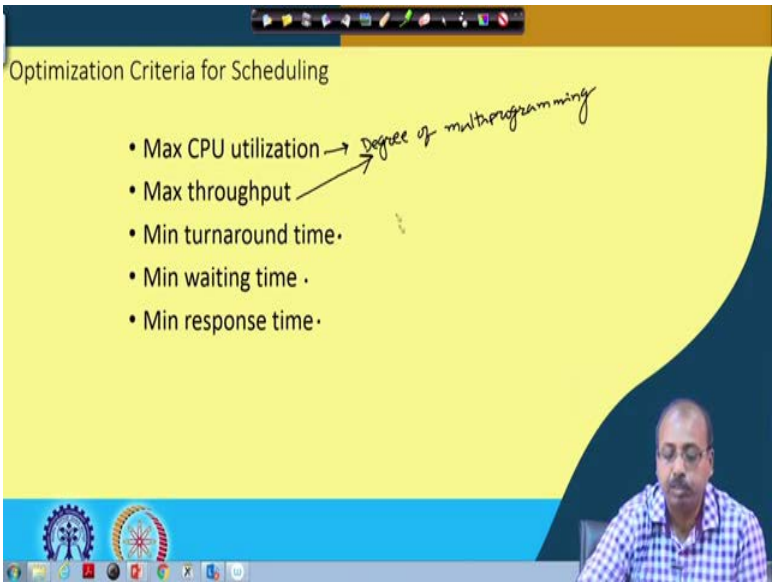
So, how much time the job is waiting in the ready queue so, that is a measure of the waiting time. Then we have got response time so, it is the amount of time taken from the submission of a request to the first response produced. So, it is you can see that when the job is submitted like a program is given for execution, when is the first response for the program comes.

As we have place in the last class a program maybe initially there may be some print statement that just say welcome the users to the system and to the work program and all maybe for computing roots of a quadratic equation. So, maybe it will be prompting for the values of a, b and c. So, from the beginning of the program how much time it takes to come to that states so, that is the response time.

So, response time is definitely important when we have got this time sharing system, multi user system. So, there we have to respond to all the users in a satisfiable fashion. So, that is all the users should feel that my program has been taken care of as soon as I

have submitted it. So, though the actual finish time or actual completion time may be large depending upon the load that you have in the system. So, this response time will ensure some sort of satisfaction or responsiveness of the system to the newly submitted job. So, these are the scheduling criteria. So, based on the scheduling criteria so, we can try to develop different scheduling policies and they have got different values for this things.

(Refer Slide Time: 04:56)



So, these are the optimization criteria for the scheduling algorithms. CPU utilization we want to maximize, throughput you want to maximize, turnaround time should we minimized, waiting time should we minimized and response time should also be minimized. So, as I have already said that it may not be possible to address multiple parameter optimization by a singles have scheduling policy, but different scheduling policies we can compute the values of these parameters. And accordingly you can come up with which may be a good scheduling policy for a particular class of jobs. So, when you are considering a particular system so, depending upon the users of the system.

So, we can see that the type of jobs coming to the system has got a particular nature for example, if it is mainly for say scientific competitions and all. So, those jobs are they are more of CPU bound jobs. So, they will be doing lot of CPU lot of computations so, the IO's will be less. So, on the other hand some jobs which are more of IO bond bound then there it is a more of IO operation will be there. So, basically this there for them the

response time may becomes an important issue. Because, if a program is highly interactive in nature and after submitting the request if the response does not come within a short span of time than the user will feel that as if the system is not working.

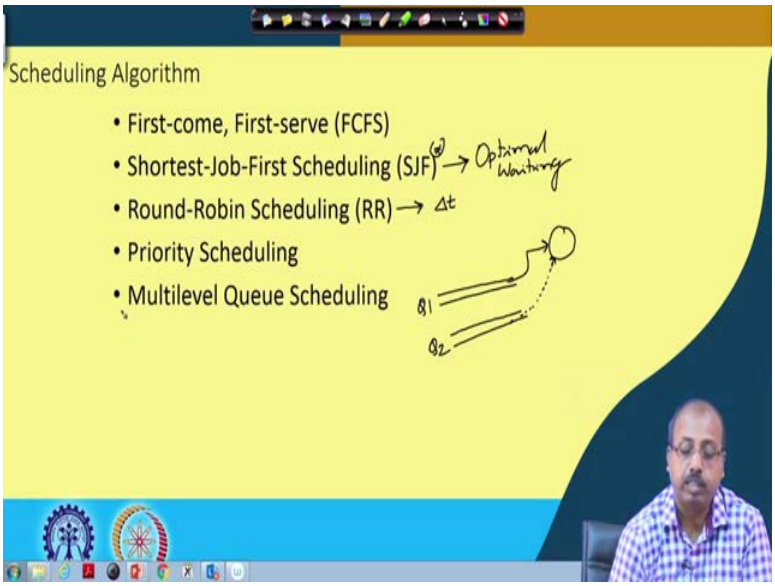
So, their response time is an important parameter whereas, CPU utilization is definitely 1 parameter that we want to maximize, but that may not be possible always. Like there may not be enough processes in the system where this CPU utilization can be made very high. So, if the CPU utilization is low one possible reason is that we do not have enough job in the ready queue to make a to keep the system busy of course, there are other reasons for which the CPU utilization maybe low.

So, that we will see later when we go to the memory management portion of our discussion. So, one reason for this low CPU utilization being low is that of this degree of multiprogramming being low. So, this there is a parameter called degree of multiprogramming. So, if this degree of so, this means that how many jobs are there in the system.

So, if this degree of multiprogramming is low; that means, number of jobs are less. So, naturally you cannot complete a number of jobs for unit as large number of per unit time as so, so utilization will be low. So, this will also affect this throughput because throughput is measured in number of jobs completed per unit time. So, if I do not have enough job in the system say then naturally this degree of multiprogramming, if it is low then the throughput will also be low. On the other hand this turnaround time will be low in that case because, now that the now I have got less jobs. So, they will be attended to very fast so, that way turnaround time for this jobs will be low if degree of multiprogramming is low.

Waiting time will also be low if the degree of multiprogramming is low because there are less jobs so, we have to wait less. Now, whether this response time so, that of course, depends on the scheduling policies. So, you cannot tell directly that whether having less number of jobs whether the min response time will be; response time will be less or not so, that is a debatable issue. So, we will come to that later. So, from the view from this angle of degree of multiprogramming this parameter values may vary.

(Refer Slide Time: 08:43)



So, next we will be looking into some scheduling policies. So, these are some well-known scheduling policies that are available in computer systems: one is the First Come First Serve-FCFS policies. So, this is whichever job has come to the system first so, that will get the chance for execution for first. So, it is a non-preemptive policy and this is a very fair policy you can say. So, in our day to day life also many a time we follow this first come first serve type of service and we have seen that is satisfy acceptable to most of the users ok.

So, first come first serve happens to be a good policy that way, second one is the Shortest Job First scheduling SJF scheduling. So, it tells that if there are a number of jobs which are in the ready queue so, you pick up the job which is of shortest duration. So, if you know the burst size of all the jobs then if you take up the shortest job first then it can be shown that the shortest job first or SJF policy so, this is optimal. So, this is it can be shown that this is optimal, in the sense that if you are trying to maximize throughput then this SJF will be sorry; if you are trying to maximize this minimize this waiting time. So, this minimizes the waiting time maximally.

So, you cannot have any other scheduling policy which can give waiting time less than this SJF policy so, will see that in detail later. So, that is why this is a very important scheduling policy. Then there are round robin scheduling. So, round robin schedulings

so, they are actually fixing some time quantum say we call it say delta t and whenever a job is given the CPU for execution so, it is given for delta t time unit.

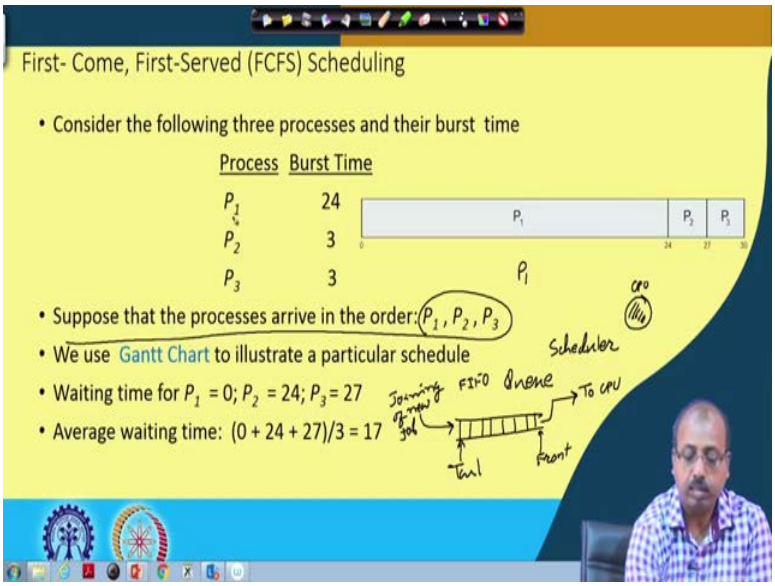
So, once that delta t time unit is over so, this job is taken out of the CPU and it is put back onto the ready queue and then the next job in the queue gets a chance for execution. So, value of delta t is an important persistent parameter and it in many of the systems. So, this is a tunable parameter in the sense that you can tune the this delta t value.

So, depending upon the type of jobs that are submitted, so, it may be necessary that we change this time quantum value with a increase it or decrease it. So, do some fine tuning. So, that way the system performance may be improved then of course, we have got priority scheduling. So, it may be that in a same in the same computer system jobs are submitted by different classes of users and those with their the jobs are of different priorities. Some of the jobs are very high priority so, they should be given chance for execution compared to others. So, maybe for example, if there are some scientific computation and there is some weather forecasting so, type of jobs.

So, those jobs they may be of low priority because they are not interactive in nature, they take lot of time for execution. Compared to that so, if we have got jobs like that response to save some fire alarm and all that. So, they should be have a; they should be have very high priority. So, these priorities are often fixed by the system or often fixed by the system administrator for different users and then accordingly this jobs are taken up based on priority. Then a combination of this policies like where we have got several levels of queues. So, in normally we have got a single ready queue in a system, but instead of that if you think of multiple ready queues.

So, if you have got multiple queues and the process. So, this is a Q 1 so, this is Q 2. So, like that if there are multiple queues then maybe the CPU will first take the jobs from Q 1 only and if this Q 1 is empty then only jobs in be taken from Q 2. So, there so that the jobs of Q 2 they are of lower priority than jobs of Q 1. So, we have got a combination of this priority and maybe the arrival time the first come first. So, may be combination of them to get this multilevel queue scheduling. So, we will look into this each of the scheduling policies one by one and discuss about the pros and cons of each of them.

(Refer Slide Time: 13:16)



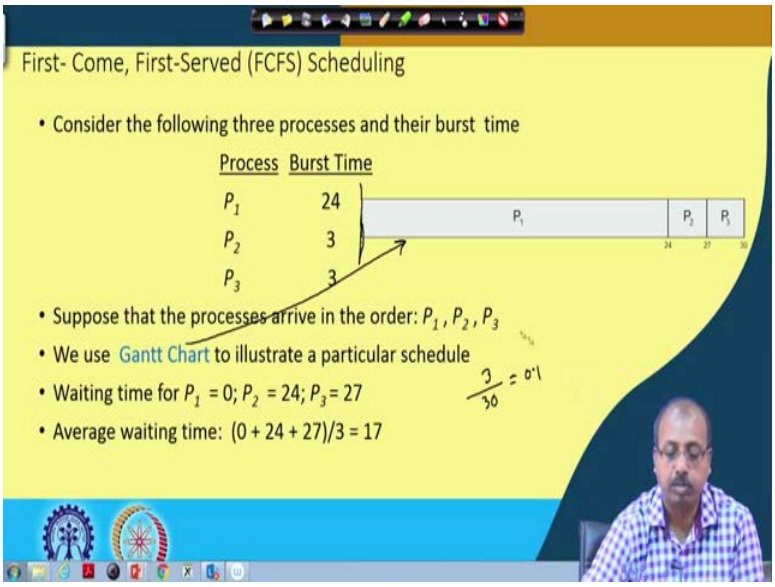
So, to start with will be looking into the First Come First Serve Scheduling or FCFS scheduling. So, we take an example to explain the case like say we have suppose we have got three processes P 1 P 2 P 3 and we assume that they are so, they are sorted by their arrival time. So, it is assumed that arrival of so, it is assumed that this P 1 it has arrived first so, P 1 so, this they the arrival order is like this. First P 1 then P 2 then P 3 so, at some point of time when the scheduler the processor is free. So, as you know that we have got there so, this P 1 P 2 P 3 they are in the ready queue and at present the CPU was busy.

This is CPU was busy doing some other job and after sometime that job finishes. So now, the scheduler has to be invoked to check to take one of this job from P 1 P 2 P 3 and assign it to the CPU. Now, in FCFS scheduling policy the decision by the scheduler is very simple. It just looks into the arrival time of the job and accordingly it takes it takes that job in to CPU. So, you see that this can be implemented if you have got a FIFO queue First In First Out queue. So, if a so, the ready queue that you have see if it is organized as a FIFO queue. So, you have got two ends; one is the front and other is the tail. So, this is the front end; so, this is the front end and this is the tail end.

So, whenever a new job comes so, it joins in the tail end and whenever a job has to be given to the CPU so, it is taken from the front end. So, this is going to CPU and this for tail end so, it is join so, joining of new job. So, it is joining of new job into the system.

So, that way we have got this Q and then the jobs are joining into the Q and the scheduler is taking from the front. So, P 1 is its burst time is 24 units. So, if I take the start time to be time 0 then we can say that it starts at time 0 and finishes at time 24. So, just before time 24 that instant so, this is P 1 is finished, now job is P 2. So, P 2 is picked up and it executes for 3 time units. So, from 27 time instant 27 P 3 executes and goes up to time instant 30.

(Refer Slide Time: 16:05)



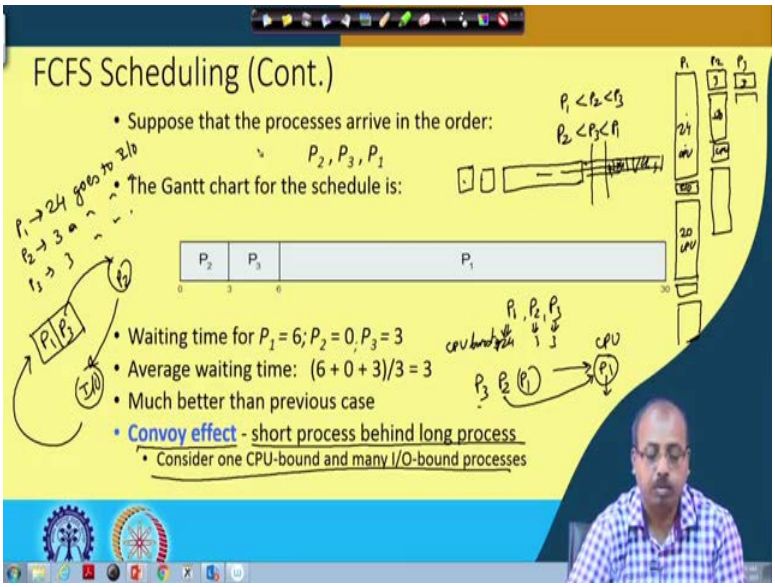
So, the diagram that we have drawn here so, this is known as the Gantt chart. So, for any scheduling policy you can draw the Gantt chart showing the duration of individual processes over the total timeline. So, this way so, waiting time for P 1. So, P 1 does not have to wait for because as soon as it so, as it is like a so, it we have started the as if the scheduler was starting from this point only and there are no further jobs. So, P 1's waiting time is 0, somebody may argue P 1 has arrived first. So, at that time something was going on in the CPU. So, P 1 had to wait for sometime, but that we are not taking into consideration.

We are just thinking in terms of the processes P 1 P 2 and P 3 and then the our time starts from the point when the first job is put into the CPU. So, out of this P 1 P 2 P 3 when the first job is put into the CPU so, from that time from that point we are starting our time. So, naturally waiting time for P 1 become equal to 0, waiting time of P 2 is equal to 24 because from 24th time instant this P 2 could start. And P 3's waiting time is 27 because

it can start from 27th time instant. So, average waiting time is 0 plus 24 plus 27 by 3 so, that is equal to 17. So, the throughput so, if you want to compute throughputs so, number of jobs completed per unit time.

So, we have completed 3 jobs over 30 time unit so, the throughput is equal to 0.1 ok. So, you have computed average waiting time, then turnaround time cannot be calculated because you do not know the arrival time of the jobs. So, the turnaround time could not be calculated and then we have got say so, it is not round robin. So, you cannot have this other parameters.

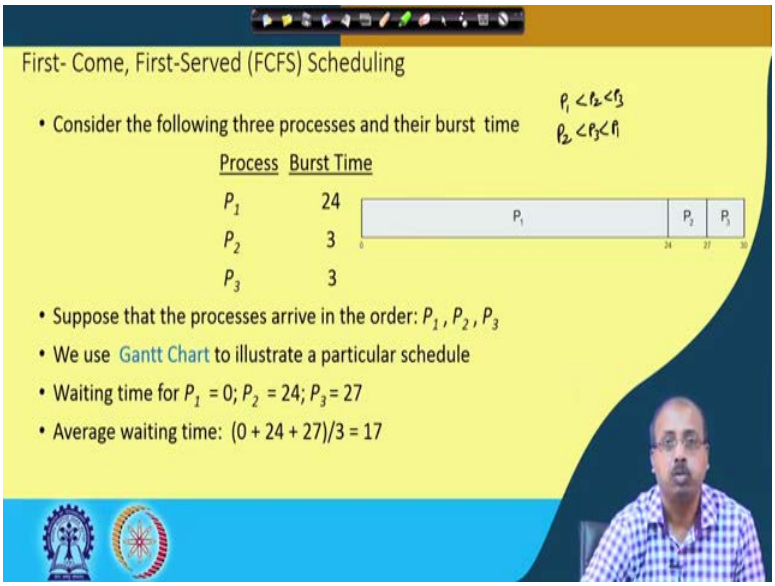
(Refer Slide Time: 18:05)



Now, going forward suppose this P 1 P 2 P 3 this process is they arrived in a different order. So, previously the order was P 1 came first followed by P 2 followed by P 3 so, this was the order. Now, we are taking the order that P 2 came first followed by P 3 and then followed by P 1. So, P 1 the largest job it came at the end. So, since P 2 came first so, P 2 is taken up for execution so, first. So, P 2 executes for 3 time units then P 3 comes on next came P 3. So, P 3 execute for 3 time units and then P 1 executes a 24 time units.

So, in this case you see the average wait time is for P 2 it is 0, for P 3 it is; for P 3 it is for P 1 it is 6 basically. So, for P 1 the time wait time is 6, for P 2 it is 0 and for P 3 it is 3. So, 6 plus 0 plus 3 equal to 3 by divided by 3 so, that is equal to 3.

(Refer Slide Time: 19:14)



So, average wait time is 3 unit whereas, in the previous case the average wait time was 17 unit. So, you see that set of jobs remain same only thing what happened is that arrived they have arrived in a different order, instead of this largest job coming at the beginning. So, the shorter jobs they came first and as a result they got scheduled earlier and then the largest job got scheduled. So, we get so, for just the arrival order of the job so, that has given us the difference in the average waiting time and that difference is a significant difference from 17 it has come down to 3; the average waiting time. So, it is much better than the previous case.

So, the this leads to some sort of idea like if there are a number of jobs to be done so, if you do the shorter job first then it is of it is expected that the waiting time will be overall waiting time will be less. And that we do in our day to day life also so, if you have got some jobs to do and the job which takes small amount of time we do it first and then try to do the larger jobs ok. So, that we do in our day to day life also and that is a very good policy; we will see very shortly after this that is a very good policy. Now, the situation that we have got here so, this is known as convoy effect. So, this is called a convoy effect so, convoy effect is like this. So, consider the previous situation when P 1 was P 1 came first followed by P 2 and followed by P 3.

P 1 was a large jobs so, P 1 was 24 unit so, P 2 and P 3 they were 3 units; so, they are pretty small. Now, if P 1 happens to be a compute bound job then so, since its these are

the CPU burst size for the jobs P for P 1 P 2 P 3. So, these were these are the CPU burst sizes ok. So, the from the nature you can understand that what happened is P 1 once it gets the time for execution, it execute for a long time for 24 time units and then it is going for some IO operations. So, after that it is going for some IO operation. On the other end P 2 and P 3 so, they are very simple so, they are P 2 is executed. So, once it is once it gets a chance for execution, it executes for 3 time unit and then it goes into IO operation.

And for P 3 also it is similar, P 3 also it executes for 3 time units and then it goes into an IO operation. So, in some sense we can understand that P 1 is a compute bound job whereas, P 2 and P 3 they are some it is very much likely that they are IO bound jobs. So, in a compute bound job what happens is that once its once CPU burst is burst is over then it goes into very small very short duration IO burst. So, this is CPU burst so, this is the next IO burst and after that it will again enter into a big CPU burst. So, that is the nature of this CPU bound jobs may be this time the CPU burst size is 20. So, this is again a CPU burst time and then after that it will again do some small amount of IO and then again it will go for a large amount of CPU burst.

So, that way it works, on the other and this IO bound jobs so, they will be doing some small amount of CPU then a good amount of IO; then again it will come back may be doing some small amount of CPU. So, this is again a CPU burst, but this duration is much smaller than IO then again it goes for a IO operation. Now, you see there are same is true for P 3 also.

Now, what is happening is that so, this P 1 P 2 P 3 they were in the ready queue and in this order when P 1 was; P 1 was scheduled earlier than P 2 and P 3 then when so, P 1 P 2 P 3. So, if you as if P 1 this is the situation and P 1 gets chance for execution. So, it get it gets a CPU for execution, now P 2 and P 3 they see that there is a big job going on in the CPU so, as a result we have to wait.

And after this long time of 24 time units so, P 1 goes out of CPU then P 2 gets a chance for execution. Now, P 2 uses it for only small amount of time and then P 2 goes to IO. So, P 1 executes for 24 time units and then goes to IO, then P 2 get chance P 2 comes P 2 executes for 3 units and then goes to IO and then P 3 comes and executes a 3 units and goes to IO. Now, in the IO cycle since P 1 is compute bound jobs so, its IO time will be

less. So, it will come back to the ready queue and join the ready queue very fast. Maybe even before P 3 has got scheduled the P 1 will join again so, the queue the situation maybe like this. So, P 3 and P 1 so, in the ready queue that is the situation. So, P 2 is in IO now, P 3 when P 2 when P 2 was in executes in the CPU. Now, as soon as P 2 goes from CPU to IO it goes from CPU to IO.

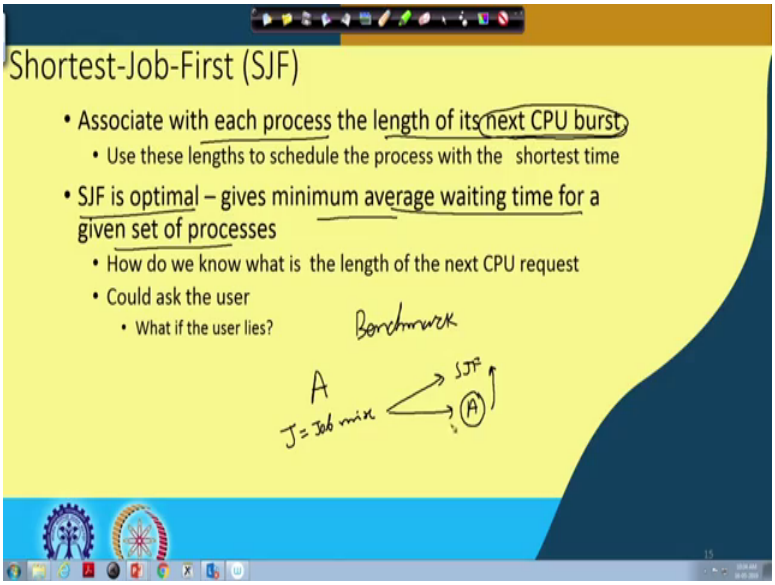
So, P 3 will get a chance for the see P 3 moves and by that times after sometime P 2 will finish of IO and comeback to this ready queue, but it will find that there is a big job P 1 in front of me. So, every time P 1 every time P 2 P 3 they complete their IO and come to this ready queue they find that there is a big job in front of them which needs to be completed first only after that it can do the operation. So, this is some sort of convoy like if there is a big convoy which is going on then what happens is that there are small if there are some small cars behind it then this is small cars. So, if there is a crossing here suppose there is a road crossing here then this convoy takes a good amount of time to cross that; cross that crossing.

So, but this P 2 this small car so, they cannot cross the cross them or maybe they cannot do takes some other path because this big convoy is going. Then again after crossing this so, maybe it again finds that. So, after this convoy this big car big convoy has gone to this position. So, this small cars gets a chance to cross this crossing and then again finds that this big car is there. So, that way so, at every point it finds that in front of me there is a big convoy that is waiting and we have to wait for that is running and that is going slowly and we have to wait for that. So, this situation is occurring every time. So, this is known as the convoy effect a short processes behind long processes. So, we have got short processes behind long processes.

So, this situation whenever it occurs so, this short processes their waiting time will increase significantly. So, every time it comes to the ready queue it finds that there is a big job before me that has to be finished. So, that way the waiting time of this short process will be going up. So, this happens when this type of situation occurs that we have got one CPU bound job and many IO bound jobs. So, this IO bounds so, CPU bound job acts as the big process and this IO bound jobs so, they are the small processes. So, as a result we get the situation similar to a convoy.

So, this is a problem with the first come first serve scheduling policy that we have to; we have to take care of this convoy effect. Now, how do you take care that is of course, a different issue as far as FCFS policy is concerned so, it does not tell you anything about how to take care of that. Maybe we need to do something hybrid up so, that this can be taken care of.

(Refer Slide Time: 27:34)



So, next will be looking into another scheduling policy which is known as the shortest job first. So, as the name suggests so, this is the we do the shortest job first. So, this is the situation that from the name we can understand that ok, if there are a number of jobs in the ready queue. So, the job to be selected next for execution is the one that has got the shortest executive shortest CPU burst type. So, we what we do for every process with for with each process we attach the length of its next CPU burst ok. So, we attach the length of its next CPU burst and then we use these lengths to schedule the process with the shortest time. So, whichever CPU burst is the smallest so, we take it.

Now, SJF is optimal so, it can be shown that if this SJF policy is optimal in the sense that it gives minimum average waiting time for a given set of processes. Given a set of processes and if you know the next CPU burst sizes and if you schedule based on this policy that is the process with the smallest next CPU burst will be schedule next. So, that is a optimal policy. So, you cannot design any other scheduling policy which can be doing better that. So, how do we, but the problem that we have is how do we know the

length of the next CPU burst so, that is a big question. So, how do we know the so, next CPU burst size?

So, until and unless the process has executed so, it is given the CPU and it completes the next CPU burst we cannot know what is the size of CPU burst. So, this is a big issue and one possible solution is could you ask the user, like can I tell the ask the user like what whenever a process is submitted what is the size of your next CPU burst. So, that is for a difficult for the user also to answer and at the same time that is also the user sometimes may be to get a benefit from the system maybe telling the value which is much less then the actual one. So, as by that gets chance into CPU and since once a process gets into CPU so, we do not a it is not preemptive. So, the process will continue till it completes the current CPU burst and go to the next IO burst.

So, the as a result if a user is lying then we cannot take care of that. So, we cannot take the process back from the CPU and give it to somebody else. So, this is the shortest job first policy and this is theoretical one and it often and so, this acts as a benchmark. So, why do I say so, is that suppose I come up with some new scheduling policy A ok, some scheduling policy A. Now how good or bad is my scheduling policy? So, if you try to compare then this average wait time. So, this is an important parameter for a any scheduling policy. So, what we can do we can take a job mix so, we can take a job mix J and then for this set of jobs. So, this job mix has got a number of jobs and these jobs we can schedule following the SJF policy and following the algorithm A.

And since SJF is optimal so, average waiting time may figure that we get from A cannot be less than this SJF. So, it can at most reach the value SJF, but it cannot be less than that. So, how close is your value to the average waiting time written by SJF so, that actually gives the goodness of the algorithm A or the policy A, that is why it is a theoretical algorithm, theoretical policy.

And this is existing in the literature because of this reason because of this bench marking for these if you are trying to develop a new scheduling policy then you can compare with SJF. So, in our next class we will see how we can approximate this SJF so, that this burst size can be estimated.