

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

Lecture – 28
Scheduling (Contd.)

The Scheduling policies that we have seen so far; so they were taking processes from the ready queue; and putting them onto the CPU for execution, so that was there. Now in the one thing that we have that was common across all these strategies, is that there was a single queue onto which the process is joining. So, when a process is submitted to the system, so, it is joining into the ready queue and there is a single ready queue.

(Refer Slide Time: 00:55)

Multilevel Queue

- Ready queue is partitioned into separate queues, eg:
 - foreground (interactive)
 - background (batch)
- Process permanently in a given queue
- Each queue has its own scheduling algorithm:
 - foreground – RR
 - background – FCFS
- Scheduling must be done between the queues:
 - ✓ Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
 - ✓ Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
 - 20% to background in FCFS

Now, the next policy that we will look into so that is of the we call it a multilevel queue structure.

So, the ready queue is partitioned into separate queue. For example, we may have two different queues; one for foreground jobs and one for background jobs. In a system that we have so what happens is that, there are some users of the system, so who are submitting their jobs and getting that thing done. And there are certain other jobs which also needs to be done like say, some system health monitoring, then some say keeping a taking a backup and all. So those jobs are there, but those jobs are not that important not that much relevant for the users. So, and if the those jobs are given higher priority, then

the problem that we get is the users will feel that as if those my jobs are not taken care off.

So, to do a better than do a something better than that, what we do is that the ready queue that you have, so, instead of having a single ready queue, we keep we take two ready queues. So, if there was a single ready queue, then both the user jobs and the system jobs they will be joining the; they will be in the same queue. Now, whatever the scheduling policy that you are following, FCFS, round robin, priority, SJF, whatever, so, everywhere, so these jobs will compete between each other's each other to get scheduled.

So, that way if the user will find that I do not there may not of the many user in the system, but the system is busy doing say its own health monitoring and things like that. But that is important from the systems perspective, but on the users perspective that is not so important. So, what is suggested is that, instead of having a single ready queue, so if we have got two ready queues. So, one ready queue for the user level jobs and another ready queue for the system level jobs ok. So, when this first so the CPU it will be getting jobs from this first ready queue. And when there is no job in the first ready queue then only it will be taking jobs from here.

So, many a time you have seen that in a computer system so, we can put some jobs in the background, so this background jobs they get executed only when there is no foreground job. So, the interactive jobs that we have in the system so, they are put on to foreground because they needs interaction from the user. So, they are made foreground jobs and the system maintenance then backup and these types of jobs; so they are put in to the background and they are often batch type of job.

So, process permanently in a given queue, so each queue also so a process when it joins based on the nature of the process, it joins either the either of the ready queue. So, it so interactive jobs they will join the foreground queue and this the batch jobs so they will join the background queue. Now after a job has been taken from this say this is the internal suppose, this is my interactive queue and this is my batch queue then if a job is taken from this interactive queue then it is getting the CPU.

Now whatever be my scheduling policies, if it is round robin then after the time quantum expires the job will be coming back to this queue. Similarly if there was no interactive job and the CPU was CPU took up batch job then after this batch job time slice expires

so it will come back to this queue. So, it will not happen that a job starting at this interactive queue and it is going to the batch queue after being swapped out from the CPU that will not happen.

So, if the process is permanently attached to a given queue. Now each queue may have its own scheduling algorithm. So in this previous example that I just talked about, so, I said that both where both the places we have got round robin, but that may not be the case. So, it may be the case that is the foreground queue it is following a round robin scheduling whereas the background queue so, it is following an FCFS scheduling. Because background jobs there round robin does not have much meaning because anyway all these jobs are to be finished. So, what is the point in putting some context switches at some intervals of time?

Because that will not help in the system throughput. So, therefore background jobs so, we normally do not have round robin. So, that is that follows FCFS. That is if a background job is taken by the CPU, then it is executed to its completion then only the next job will be taken up. Of course, we can argue that when this background job is executing, when if a foreground job comes.

And then again that will give rise to another type of policy, which is the PMT policy PMT multilevel queue. And then, whenever this foreground job comes then we have to take the CPU back from the background job and give it to the foreground job for execution and all.

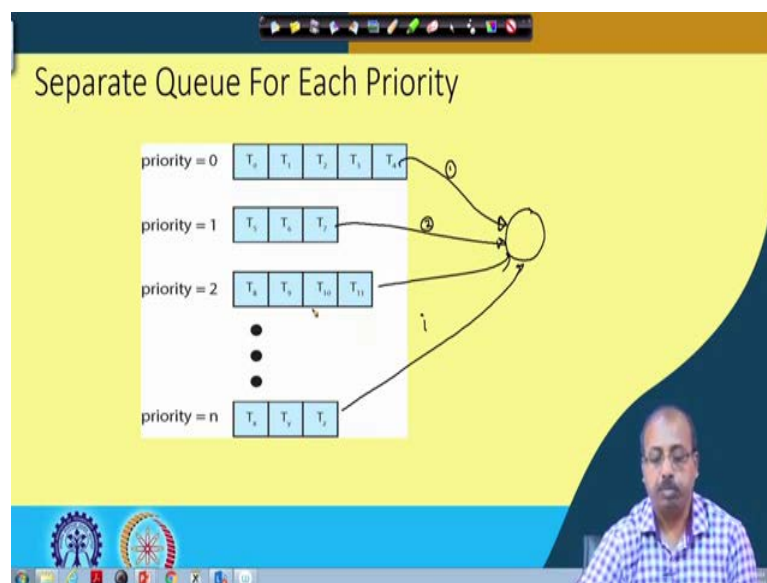
So, scheduling must be done between the queues, we have got fixed priority scheduling that is the serve all from foreground then from background. So, this is the one possibility like, the way I was telling that when there is no foreground job then only background job will be executed. So, the difficulty with this particular policy is that, if there is a continuous flow of foreground jobs then the background jobs will never get a chance for execution.

So, if you have got this fixed priority scheduling, if the priorities fixed between these two. So, this is of higher priority than this one, then if there is a continuous flow of jobs in the this high priority queue, the foreground queue then there is a high chance that is background jobs, so they will be suffering and they will not get chance for execution and they will stop.

So, fixed priority scheduling has got this problem. Then other possible other alternative is the time slice, so it says that each queues gets a certain amount of CPU time which it can schedule amongst its processor. For example, you can say for 80 percent of the time the scheduler will take up job from the interactive queue or the foreground queue. And for 20 percent of the time it will be taking from the batch queue. So, that is good. So, and this 80 percent to the batch queue, 80 percent to the foreground queue and within that 80 percent time so it does round robin ok.

So, that is it gives the fraction of the 80 percent time to each of the jobs that is there in the queue. On the other hand for FCFS for the background queue part, so it is giving 20 percent of the time. And 20 percent of the time then that time is given to the jobs in a first come first serve fashion FCFS fashion. So, this way we have got this multilevel queue structure and we can categorize between this scheduling policies between among the queues. We can also differentiate between them in terms of the amount of time that is given to jobs of individual queue. So, this is a good compromise, because it tries to differentiate between the types of jobs that we have in the system and we are trying to put them in all together different queue.

(Refer Slide Time: 08:18)



So, this is a typical situation. So, previous example that we were looking into so there we had only two levels, ok so, the foreground job and the background job. So, in general I can have separate queue for each priority level. Suppose I am supporting say n plus 1

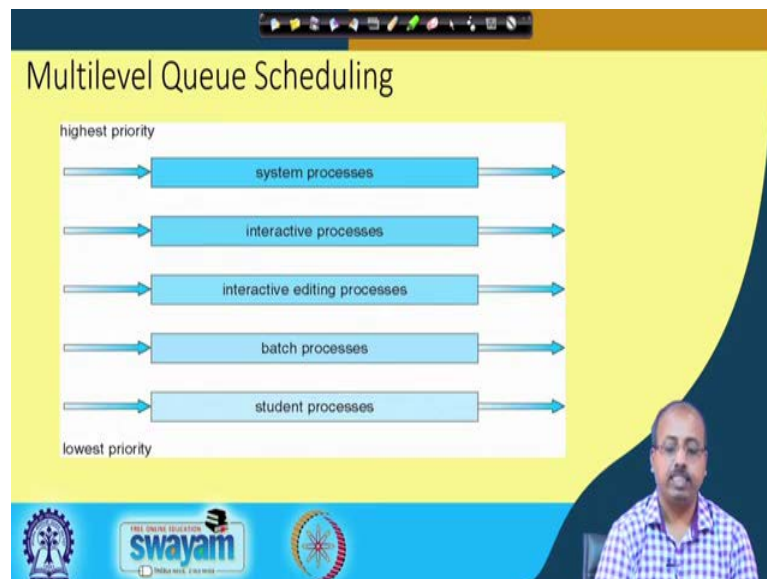
different priority levels then there is each priority level may be implemented using a different queue.

So, priorities 0 here, priority 1 here, 2 here. So, the when the scheduler comes so, the scheduler when the scheduler comes so scheduler will they are first consider that this jobs from this queue. And within this queue so maybe it will be doing round robin. And once if there is no job in this queue so this has got the highest priority then it takes job from this queue.

So, this is the next highest priority and again it will be doing round robin across the jobs T 5, T 6, T 7 so like that it will be taking from other queues. And the so if the all previous levels all higher levels queues are empty then only it will take up jobs from the current level. So, it will be taking jobs from the level 2, only if the level 1 and level 0; so these two are free these two are empty. And within level 2 so, it will be doing a round robin.

So, that is how this we can have separate queue for each priority level and it can be we can have some scheduling policy there.

(Refer Slide Time: 09:47)



So, a typical example of this higher different priority level is like this. So, we have got different priority levels like here, may be the system processes they are of the highest priority.

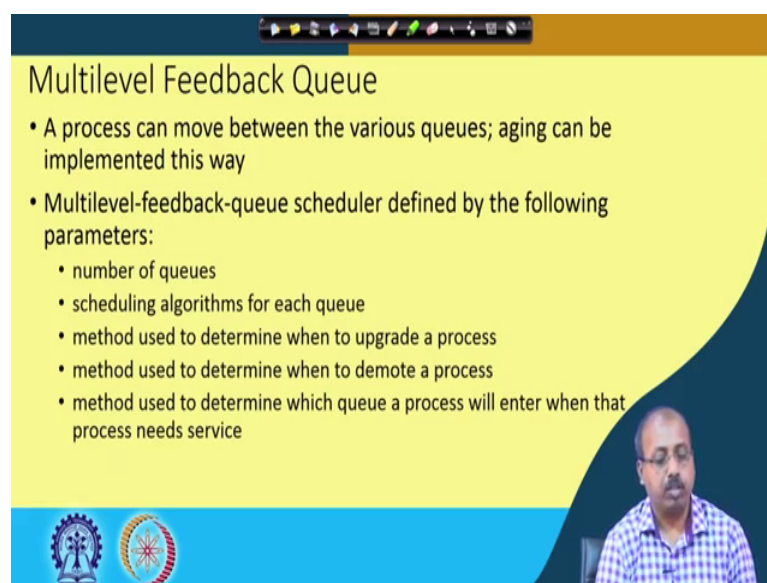
So, this is one possible type of priority assignment, which says that the system processes are very important processes so they must be given the highest priority. Then comes the interactive processes, that gets the may be some input values and compute some data and puts it onto the output so like that so they are interactive processes.

Another class of process may be interactive editing processes. So, editing process means a user will be typing some making some text file and writing something on the text file etcetera. So, that is an that is also interactive, because the user key process have to be stored and all, but the point is that since it is an editing process, so it is much slower compared to the previous one interactive processes. So, this editing process, so it there is the wait time is the IO wait times are generally much higher than the normal interactive process.

Then we have got batch processes and maybe in academic institution maybe the student processes they may be given the lowest priority; because there may be many other jobs that the system is busy to do, so they may have the lowest priority.

So, from the lowest priority to highest priority we have got different queues. So, this is just a hypothetical example so in a particular system depending upon the classes of users that we have in the system. So, we have we may do this type of scheduling multilevel queue scheduling.

(Refer Slide Time: 11:29)



Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
 - number of queues
 - scheduling algorithms for each queue
 - method used to determine when to upgrade a process
 - method used to determine when to demote a process
 - method used to determine which queue a process will enter when that process needs service

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. At the bottom right, there is a small video inset showing a man in a blue and white checkered shirt speaking.

So, in multilevel queue scheduling one problem that we had is that one process, so, a process is that is there in the in a particular level of queue. So, look into this diagram. So, if a process joins in the priorities 0 queue, then it does not go to some other priority levels. So, after getting the time quantum it joins priority 0 queue only or say priority 1 queue, so, it is after finishing its time quantum it is joining priority 1 queue also only.

So, it is there is no provision by which a process can go from say priority level 1 to priority level 0. Or some priority level 2 to priority level 1 to priority level 0. So, this type of movement is not possible when we have got this multilevel queue. But many times, so what is happening is at the beginning of the program or beginning of the process itself we are determining what should be the priority of the process based on the characteristic of it.

Now, this characteristic of a process may change over time. So, normally if you look into this diagram, so what we are trying to do in this diagram what we are trying to do is that the jobs which are more interactive in nature so they are put on to the higher levels of queues.

Now it may so happen that a process is initially compute bound. So, it does lots of computation, but what time its characteristics changes, it becomes more of IO bound job. And vice versa a job is initially IO bound, but after sometime after getting all the after reading all the input data from different files; now it goes into a deep computation phase and then where does lots of computation.

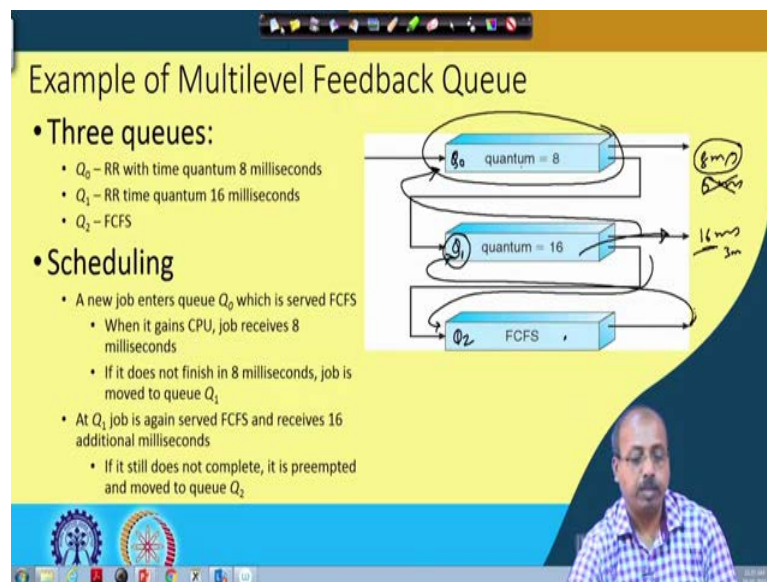
So, this way we can have the difference. So, we can the nature of the process may change over time. Now this type of queue structure that we have here, so it cannot take care of that because once a job joins interactive queue it remains in the interactive queue only and interactive queue the problem is that it is getting chance for execution faster. But every time it gets a chance so, it will be executed for some small time quantum only, but that is not good enough for the process to compute. So, again it has to compute with other processes to see like if it is if when the next CPU words will be available.

So, that way computation will be more, whereas a batch process, so normally when we take up a job from the batch processes to that is given chance in terms of FCFS. So, once it gets the CPU, so it executes to its completion. So, if a job is has changed its nature from IO bound to compute bound, then it is better that it gets higher chunk of time

quantum while going into execution. So, that we may have a policy or may have a procedure by which this queue of a process changes over its lifetime.

So, the next scheduling policies that we are going to see, so that is doing that one so it is a multilevel feedback queue. So, a process can move between the various queues and aging can be implemented this way. So, this multilevel feedback queue scheduler is defined by this parameter; number of queue, scheduling algorithms for each queue, method used to determine when to upgrade a process, method used to determine when to demote a process and method used to determine which queue a process will enter when that process need service.

(Refer Slide Time: 15:05)



So, this diagram actually tells the situation better, like we have got 3 queues here; Q_0 , Q_1 and Q_2 . Q_0 it uses round robin scheduling with time quantum value equal to 8. Q_1 it uses it also uses round robin scheduling, but the time quantum value is 16. And this Q_2 is first come first serve. So, whenever the new job arrives so it joins in this time, this first top level queue Q_0 and since this queue has got the highest priority, so, it gets the chance for execution. So, it joins a goes to the CPU first and execute for 8 allowed to execute for 8 time units.

Now, two things can happen, one thing is that the job may be so if the job may execute for entire 8 milliseconds and if it, so it is the job is given 8 millisecond. So, the job maybe it execute for say 5 millisecond and releases the CPU it comes out of goes into an

IO operation after 5 millisecond. So, in that case the job is an interactive job, so the in that case the job will be joining back the queue the top level queue Q 0 only.

Whereas, if the job is a bit compute bound then what will happen is that the it will this 8 millisecond will not be sufficient for the job to be completed. So, it will use up entire 8 millisecond and after 8 millisecond time the timer will expire and the scheduler will come. And scheduler will see that the job that was given CPU so it has executed for 8 millisecond and asking for more.

So, it understand that this is more of a compute bound job. So, instead of putting it back onto the queue Q 0. So, it will put in on to the queue Q 1. So, the job is taken to the queue Q 1. So now, the job that is in Q 1, so this will get a chance for execution only when this higher level queue is empty that is fine.

But assuming that higher level queue is empty, so this when its job is taken from this queue for execution so, time quantum value is 16 milliseconds, so, it gets CPU for more time ok. And then after 16 millisecond also if the job has not finished; that means, though the job is heavily compute bound job and it is it demotes to this next level so, its comes to the FCFS one. And vice versa it can so happen that a job using the queue in the queue Q 2 and we find that within a very small amount of time it releases the CPU going to IO operation. Then instead of putting put it on to this queue Q 2, so it may go to queue Q 1 ok.

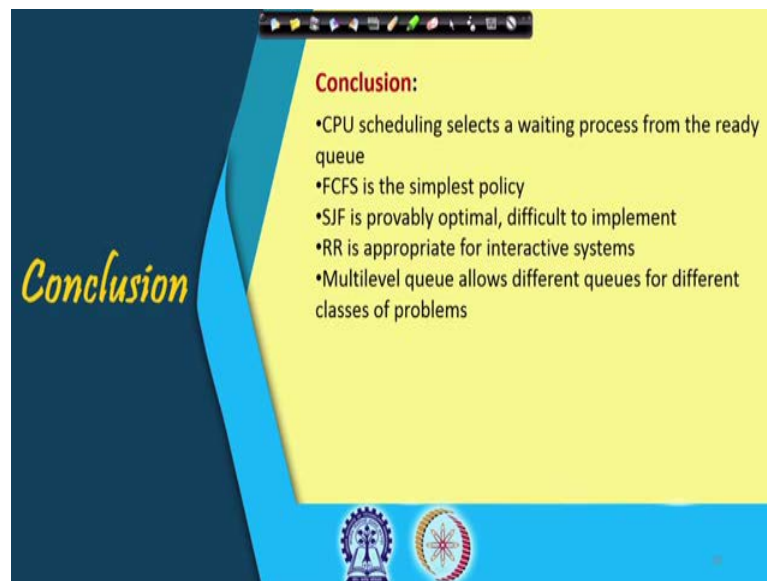
And similarly, if it if a job from Q 1 is given CPU for execution, so it executes a16 it execute for say 3 millisecond and then releases the CPU then we understand that this is this has now become more of a interactive job so, it is moved from here and put on to the queue Q 0.

So, this movement can be both way. So, once a job enters the system it enters into this top level and if it uses up its complete time quantum, so it will be going down and down into the queues into the queues structure. On the other hand, if a job from this lower level queue, it finishes or it releases the CPU earlier than the time quantum that is given to it. Then it may bubble up and it may go to the higher queue level.

So, this way this multilevel feedback queue structure, so, we can move further. So, to summarize, so a new job enters queue Q 0 and which is served as first serve basis. And

then when the CPU it gains the job, so it is getting a 8 millisecond it will get. If it does not finish within 8 millisecond, the job is moved Q 1. The job at Q 1 again served first come first serve basis, but for 16 additional millisecond but it is a 16 milliseconds. If it is still is not completed, so it comes to Q 2 and then the Q 2 from Q 2 it is totally first come first serve for the entire duration. So, this structure is very good we can have this multilevel feedback queue structure and this is implemented in many of the operating system also.

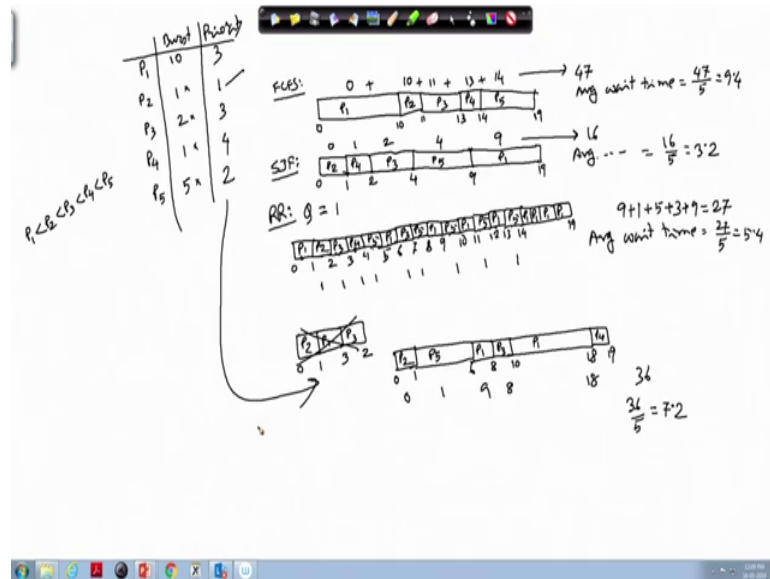
(Refer Slide Time: 19:32)



So, to summarize so have got this CPU scheduling policies that selects waiting process from the ready queue. We have got this FCFS is the simplest scheduling policy, we have got SJF which is optimal, but it is difficult to implement. Then round robin policy so it is appropriate for interactive systems and we have got this multilevel queue that allows different queues for different classes of problems. So, we have seen different types of problem and after that we have got other scheduling policies like for real time systems.

So, there are scheduling policies for this others this multilevel multiprocessor system. So, there are scheduling policies, so the so that is beyond scope of this particular course, so we can discuss about them in maybe we can go to them in some books ok. So, what we will do next is that, we will take up some example and try to solve all those examples and of this scheduling problems and see how do they behave.

(Refer Slide Time: 20:52)



So, let us take an example of some processes; so, suppose I have got 5 processes P 1, P 2, P 3, P 4 and P 5. These are the 5 processes and these processes their burst times are like this, burst times are like say 10, 1 then 2, 1, 5 and their priorities are priority values are like this.

So, this is 3 this is 1, this is 3, this is 4 and this is 2. And the arrival order of this processes is P 1 it came first, followed by P 2, followed by P 3, followed by P 4, followed by P 5. So, if we try to see the FCFS type of scheduling for this set of processes, then at time 0, P 1 has arrived. So, P 1 burst time is 10, so it will execute for 10 time units.

It will execute for 10 time units then P 2 will execute for 1 time unit, this is my P 2. Then P 3 will execute for 2 time units. So, this is P 3, so that is time is 13. Then P 4 has come, P 4 execute for 1 time unit. And then P 5 execute for 5 time units. So, P 5 executes for 5 time units so that is 19. So, in this case so if we try to calculate the wait times, then for P 1 the wait time is 0 for P 2 wait time is 10, for P 3 it is 11, P 4 is 13, P 5 is 14.

So, so total wait time is sum of all these values, 13 plus 27 plus 11, 37 then 47 total wait time is 47. So, average wait time so average wait time is equal to 47 by 5 so it is 9.4 ok. Now if we follow the SJF policy is SJF policy, then what happens is that the shortest job that will be done first so P 2 and P 4 both are shortest.

So, let us say that we take up P 2, after that P 4 is taken up so P 2, P 4 done. Now P 3, P 3 is taken up for 2 time unit, so it goes to 4. Then we have got P 5 for 5 units so, that that makes it 9. And then P 1 for 10 unit that is 19. So, wait time P 2 is 0, P 4 is 1, P 3 is 2 then P 5 is 4, P 1 is 9. So, the total time total wait time is 9 plus 4 13 plus 3, 16. So, average wait time equal to 16 by 5, 3.2, so, average wait time is 3.2.

Now, if we do a round robin, if you do a round robin with time quantum with the Q value equal to 1 ok. The queue value equal to 1 then what will happen? First P 1 will be executed for 1 time unit, then P 2 will be executed for 1 time unit then P 3 will be executed for 1 time unit. So, I can so P 3 will be executed for 1 time unit. Then P 4 will be executed for 1 time unit then P 5 will be executed for 1 time unit.

So, at present so then P 1 will again be executed for 1 time unit ok. So, P 1 will be executed for 1 time unit then, again P 2 will be executed so, P 2, P 2 is over so P 2 will no more be executed. Now P 3 will be executed for 1 time unit. Then P 4 is also over. Then P 5 will be executed for 1 time unit ok. So, my P 2 and P 4 they are done. Now P 3 is also over. So, what remains is P 1 and P 5 out of that P 5 has executed for two units and P 2 has also P 1 has also executed for two unit. So, now, P 1 will come 9, then P 5 will come 10. Then again P 1 will come so, P 1 we have got 1, 2, 3, 4 and P 5.

We have got and 1, 2, 3, 4. So, another P 5 will another P 1 and P 5 will come; and now P 5 will also over and for P 1 so it will execute for 4 time units. So, this context switching will happen. So, this will be finishing at time 10 5 time 19. So, they will be P 1 only.

So, you can see like how long time how much time P 1 is waiting in the ready queue? So P 1 is waiting so P 1 actually it is it is waiting at all this time. So, this is 1,1,1,1 then P 3 P 5 is 1 this P 5 is 1. So, whenever so P 1 was there for the entire duration and whenever the CPU is doing something else P 1 is waiting. So, it is waiting at this time this time 1, 2, 3, 4, 5, 6, 7, 8, 9.

So, it is waiting for 9 units. So, P 1 is waiting for 9 units plus P two. So, P 2 was waiting here and after the P 2 was not waiting. So, P 2 is waiting for 1 unit then P 3, P 3 is waiting for 1, 2 then 3, 4, 5 ok. So, P 2 P 3 is waiting for 5 units then P 4 P 4 is waiting for 1, 2, 3 and then P 4 does not wait anymore. So, P 4 is 3 unit, then P 5 P 5 is waiting for 1, 2, 3, 4 then again 5, 6, 7, 8, 9 so nine units. So, this is the wait times for different

processes in the ready queue. So, this value is 9 plus 1, 10 plus 5, 15, 18, 23, 27. So, the average waiting time is 27 by 5.

So, average waiting time is 27 by 5. So, that is equal to 5.4. So, this way you can make it for the priority based scheduling also like between; so initially P 2 and P 3, P 2 and P 2 has the minimum priority. So, first P 2 will be scheduled for 1 time unit and so once P 2 is done. Now P 1 and P 3, they have got they have got this thing they have got same priority.

So, if I assume that my time quantum value is two units. So, this is P 1 will be scheduled for 2 time unit, then P 3 will be scheduled for P 3 will be scheduled for 2 time units. And now P 3 will be over sorry before that P 2 will come P 5, P 5 will come. So, the first P 0 first P 2 will come, for 1 time unit and after that P 5 will come for 5 time units.

So, this is P 2 this is P 5, for 5 time units and after that there will be a this P P 1 and P 3 if I assume that they will be done in a round robin fashion for 2 time unit; that the time quantum is taken to be 2 time units then P 1 and P 3. So, then now P 1 is the process, that has that has got the highest priority and P 1 has already executed for 2 time units.

So, P 1 has already executed for 2 time unit. So, remaining 8 time units it will execute. And then after that we have got this P 1, P 2, P 4, P 4 will come and it will execute for 1 time unit. So, this is P 4 this is finish at 19. Now we can calculate the wait times. So, P 2 is a wait time is 0 P 5's wait time is 1, P 1's wait time is here it is 6 plus 1, 7 plus 2, 9. So, P 1's wait time is 9 P 1's wait time is 9.

Now, P 3's wait time is equal to was 1 plus 6 that is 8, P 3's wait time is 8 and P 4's wait time is 18. So, if we add them, so 18 plus 8 26 plus 9, 35, 36. So, the average wait time is 36 by 5, so that is 7.2. So, this waits time so based on this priority. So, we can the scheduling like this. So, this way you can do different types of scheduling of different cases. And accordingly you can determine like, which one is giving you the minimum average wait time and that may give a study about which algorithm doing better for a particular job mix.