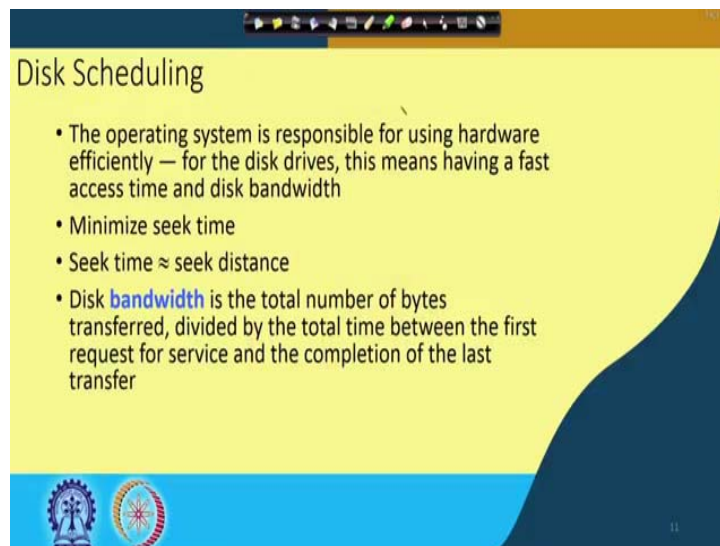


Operating System Fundamentals
Prof. Santanu Chattopadhyay
Department Of Electronics And Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 58
File System and Secondary Storage (Contd.)

Next we will be looking into the concept of Disk Scheduling.

(Refer Slide Time: 00:28)



So as we know that when processes are running in a computer system, so they will generate lot of request for disk access and by this time we understand if the disk access is random. So depending on the in terms of the sector numbers that the processes are requesting to access, if it is fully random, then this seek time will come. Many a seek time will come repeatedly.

So, that way the disk head movement will become problematic. So, this that will increase the total time to access the individual information. So, this operating system it is responsible for using this hardware efficiently and for the disk types, this means having a fast access time and disk bandwidth. So, this we want that this access time should be low and this bandwidth should be high. So, it will be able to transfer more amount of data per unit time while we are willing to spend very little amount of time as access time.

So, how to reduce this? So we want to you have to minimize the seek time and seek time is proportional to seek distance that is by how many tracks we have to move, or how many cylinders we have to move. So, that determines the seek time. So, this seek time has to be reduced. So, how can we reduce this seek time and this disk bandwidth is the total number of bytes transferred divided by the total time between the first request for service and the completion of the last transfer.

So from if I have got a number of requests, so when the first request was generated till the last transfer is over. So, over this time, so how much data we could transfer, so that will determine the bandwidth. So, this bandwidth we need to make it high and this time we want to reduce.

(Refer Slide Time: 02:20)

Disk Scheduling (Cont.)

- There are many sources of disk I/O request
 - OS
 - System processes
 - Users processes
- I/O request includes input or output mode, disk address, memory address, number of sectors to transfer
- OS maintains queue of requests, per disk or device
- Idle disk can immediately work on I/O request, busy disk means work must be queued.
 - Optimization algorithms only make sense when a queue exists

Handwritten diagram: $P_1 \rightarrow \begin{matrix} (30) \\ (45) \\ 15 \end{matrix}$

So, how to do that? So, we will see. So there are many sources of disk IO requests. Who can generate this IO request? Like operating system can generate some requests to access particular disk because operating system programs may be stored in the disk.

So, it has to access the disk. In the system processes, they may request for some disk access, ok. So, similarly the user processes also they can do some file operation and accordingly ask for some disk access. This IO request it includes input or output mode, disk address, memory address, number of sectors to transfer. So any IO request input output requests that you want to give. So that should have this whether it is a read operation or a write operation.

So if it is a read operation that is that means we are trying to read something from the secondary storage to the primary storage. So, that is an input operation for the primary storage or it is a write operation or output operation from the main memory, we want to write some data onto the secondary storage. So there is the output mode in the disk address which particular block or which particular sector we want to write, ok. So that is the disk address and then the memory address like if it is a transfer from memory to the disk, then from which memory block we are willing to transfer.

So, transfer is always in terms of blocks. So, which particular memory block we are willing to transfer to the disk or if it is a disk read operation? So after reading the particular sector where are you going to put it in the main memory. So, that way this memory address is identifying the corresponding memory block number, block address and number of sectors to transfer. So, how many sectors you want to transfer? So, these are the things that should be noted down in while making an IO request.

Now operating system it maintains the queue of requests per disk or device. So it maintains a queue because the queue may be served in a different order but for optimization, so it may try to do that some it may try to do something, so that the seek time is less because seek is the major time. So, you want to reduce the seek time. So, that way we may like to do that, then idle disk can immediately work on IO. Request by busy disk means work must be queued.

So if the device if the disk was idle, then of course whenever an IO request comes, so it can move on to that particular request, but if it is the disk is currently busy doing some other transfer, then the work has to be queued and it will be; it will be done after some time only. Optimization algorithms only make sense when a queue exists. If there is no queue, then naturally I do not have anything to do. So, there is no optimization.

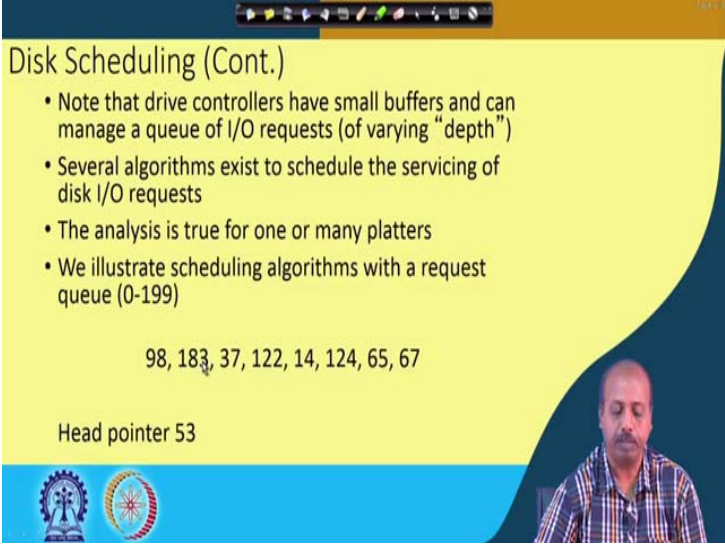
For example if I have got requests generated by a single process, suppose in my computer system there is only one process. Now P1 no other processes process is there in the system. So, how and this P1, it is doing some disk access, it is may be it is writing some blocks on to the disk. Now it first tells that I want to write at sector number 30, then it wants to say that I want to write on sector number 45, then it tells I want to do it at sector number 15 so, like that.

Now in these type of cases this disk scheduling does not have any meaning because this program after telling that I want to write this on to block number 30, it will wait for the transfer to be over. Then only it will come it will be generate a request for writing on to block number sector number 45. So, that way there is no point in trying to do some manipulation in this ordering. So, there is no point making this request queued up because the at one point of time there is only one requests that the disk drive will see.

So, naturally there is nothing to be there is nothing to be done by the disk scheduling algorithms. So, that is that behavior will be same. So, optimization algorithms they will make sense only if there are requests from multiple processes and there is a queue. So, maybe that we have got say ten different processes and each of them they are generated some; generating some random disk service requests. Now how are you going to sort them, in which order you are going to sort them, so that makes it really a sensible operation.

So, will be looking into this optimization algorithms keeping in view that this is applicable only when a queue is existing.

(Refer Slide Time: 07:04)



The slide is titled "Disk Scheduling (Cont.)" and features a yellow background with a dark blue curved shape on the right side. At the top, there is a navigation bar with various icons. The main content consists of a bulleted list and a sequence of numbers. At the bottom left, there are two logos: one of a gear and another of a circular emblem. A video inset in the bottom right corner shows a man with a beard and a plaid shirt speaking.

Disk Scheduling (Cont.)

- Note that drive controllers have small buffers and can manage a queue of I/O requests (of varying "depth")
- Several algorithms exist to schedule the servicing of disk I/O requests
- The analysis is true for one or many platters
- We illustrate scheduling algorithms with a request queue (0-199)

98, 183, 37, 122, 14, 124, 65, 67

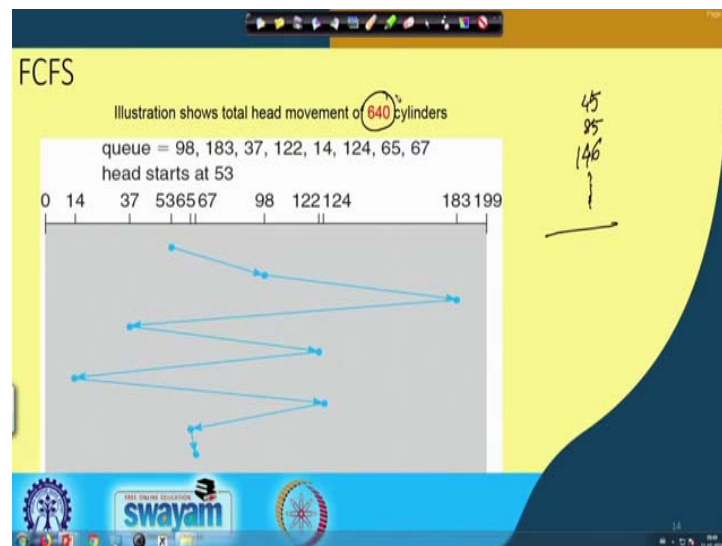
Head pointer 53

So, the drive controllers have small buffers and can manage a queue of IO requests of varying depths. So, we assume that this drive controller the disk drive controller it will have some amount of memory associated with it that can keep a note of this queue the requests that are waiting now.

Several algorithms exist to schedule the servicing of the IO requests. We will see some of those algorithms. The analysis is true for one or many platters. It does not matter whether it is one plate or many plates. So, will be request will be considering that we have got this we have got the request on this particular cylinders 98, then 183, then 37, then 122. So, and this total the request queue, so it can range from 0 to 199.

So, there are in my disk. So, there are cylinders 0 to 199 and my requests are like this initially my head pointer is at cylinder 53 and then the queue is like this. There is a request at cylinder number 98, then 183, then 37, 122. So, this is at this point now we have to find out what can be the scheduling order for this request assuming that at this point of time, the head is at cylinder number 53.

(Refer Slide Time: 08:28)



So, the first and the very simple type of strategy is the first come first serve. FCFS So, FCFS since this 98 came first. So, from 53 it goes to 98 and from 98 it goes to 183, from 183 it comes back to 37 ok, then from 37 it goes to 122, comes back to 14, goes to 124, comes back to 65, then goes to 67 and then since there is no more request, so it remains at cylinder number 67.

Now, you can compute what is the total seek time that is that has been seen by this particular strategy, ok. So, this from 53 to 98. So, this is 45 ok, then from 98 to 183. So, that is 85, then from 185, 183 to back to 37, so, that is 6. So, by 146 cylinders, it will

move. So, you can find out you can sum of all these time all these movement across cylinders.

So, in this case if you sum it up, then this number turns out to be 640. So, the total head movement is of 640 cylinders. So, whether it is good or bad, so that is the different issue, but the policy that it is following is the first come first serve, so, FCFS policy ok.

(Refer Slide Time: 10:16)

SSTF

- Shortest Seek Time First -- selects the request with the minimum seek time from the current head position
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests
- Illustration shows total head movement of **236** cylinders

queue = 98, (83)37, 122, 14, 124, 65, 67
head starts at 53

0 14 37 53 65 67 98 122 124 183 199

12
2
30
⋮

So, there can be other policies as we will see like say. So, like for jobs, CPU jobs we had the shortest job first. So, here also we can think about the shortest seek time first SSTF algorithm. So, in SSTF we will select the request with the minimum seek time from the current head position and then, so this is some sort of SJF scheduling. So, which is like this that we start at cylinder number 53 just like a just in the previous example you start at 53, then among these entire queue up request.

So, 65 is the nearest one. So, it goes to 65, from 65, 67 is the nearest. So, it goes to 67, from 67 it comes to 37 because that is the nearest. Now from 37 it goes to 14, then it goes to 98, then 122, 124 and finally 183.

So, it goes this way and again if you sum up this seek times, then this is from 53 to 65 that is 12, from 65 to 67, 2, from 67 to 37, 30. So, if you sum them up in this way the value turns out to be 236 cylinders. So, it has to move by 236 cylinders. So, definitely this is the optimal one because you cannot have seek time less than that because we are

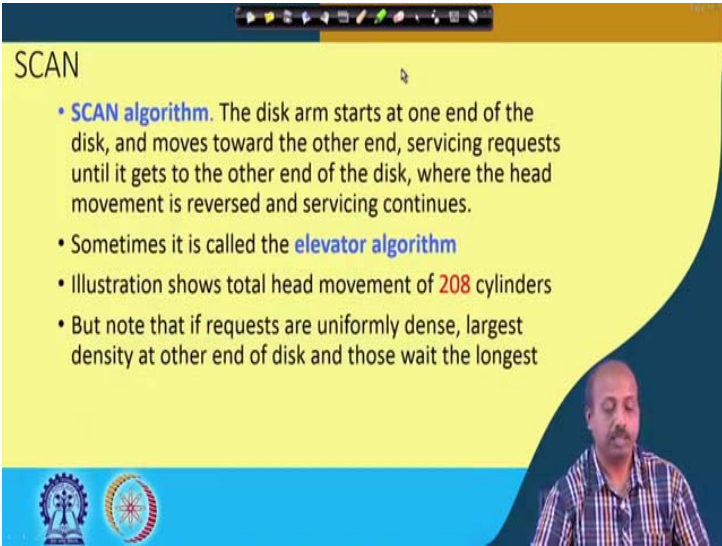
going in the Shortest Seek Time First policy. So, it is the best one, however that the problem is that it can create starvation for some requests.

So, since the processes are always running in the system, so it may so happen that the processes they are generating this disk requests continually in that order. For example, this one process is generated this request of 183 which here you see is served at the end, but assume that after it has serviced 124, there is continually request generated in this in the vicinity between say.

So, in this region all these requests are generated, so that the disk movement, the head movement remains in this region only because of the Shortest Seek Time First. So, this never goes to this 183 request. So, that way there is starvation for this particular request. So, SSTF policy is fine, but it has got the potential for starvation. So, that has to be taken care of, but of course for SSTF we cannot do that. So, our seek time is optimized so, but the starvation problem may come.

So, that is why so this may not be a very good strategy and also of course there is nothing like prediction like at one point of time we can see like which in job which requests are pending. So, based on that it is trying to figure out like what is the next, where to go next.

(Refer Slide Time: 13:17)



SCAN

- **SCAN algorithm.** The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- Sometimes it is called the **elevator algorithm**
- Illustration shows total head movement of **208** cylinders
- But note that if requests are uniformly dense, largest density at other end of disk and those wait the longest

The slide features a yellow background with a dark blue curved border on the right side. At the bottom left, there are two circular logos. At the bottom right, there is a video inset showing a man with a beard and a plaid shirt speaking.

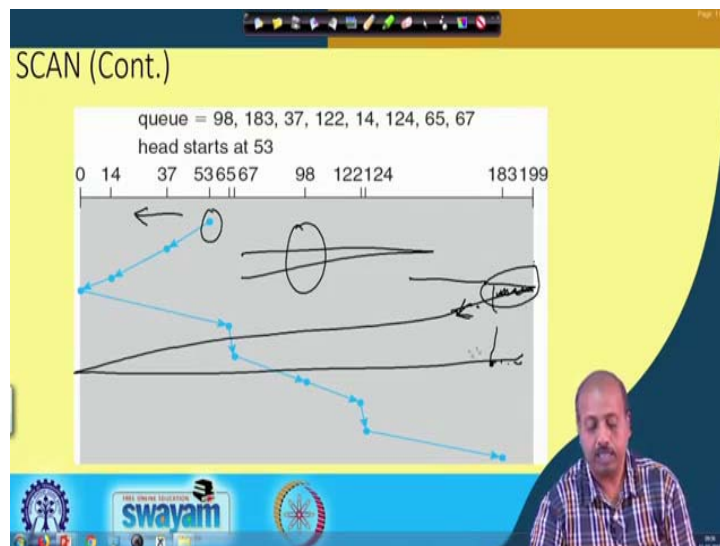
Then another type of algorithm is known as the SCAN algorithm. So, this SCAN algorithm, so this is very simplistic one. The disk arm starts at one end of the disk and moves towards the other end. So, it scans the disk from one end to the other end.

So, servicing requests until it gets to the other end of the disk and there the disk movement is reversed and servicing continues. So, this is some sometimes it is called an elevator algorithm. So, if the elevator is designed in such a fashion that it starts from say ground floor and starts and goes to the top floor stopping at each of the intermediary floors and after reaching the top floor, again it comes down to the ground floor, again stopping at each of the intermediary floors.

So, that type of algorithm is known as SCAN algorithm. Whether is it is good or bad, so that is of course an issue, but we will see that this particular example. So, this will equal 208 cylinder movements. So, SSTF so which was recording 236 cylinder movement. This will equal 208 cylinder movement, but if the requests are uniformly dense, so largest density at other end of the disk and those with the longest.

So, basically; so basically what happens is that the towards the end the requests are served much later. So, that way it is very difficult for this request that are coming to the end.

(Refer Slide Time: 14:52)



So, this is the thing. So, SCAN algorithm so initially suppose it is at cylinder number 53 and we assume that the head was moving in this direction, so, it comes to 0 ok. In the process in the journey it serves this requests are 37 and 14 and then it comes to 0.

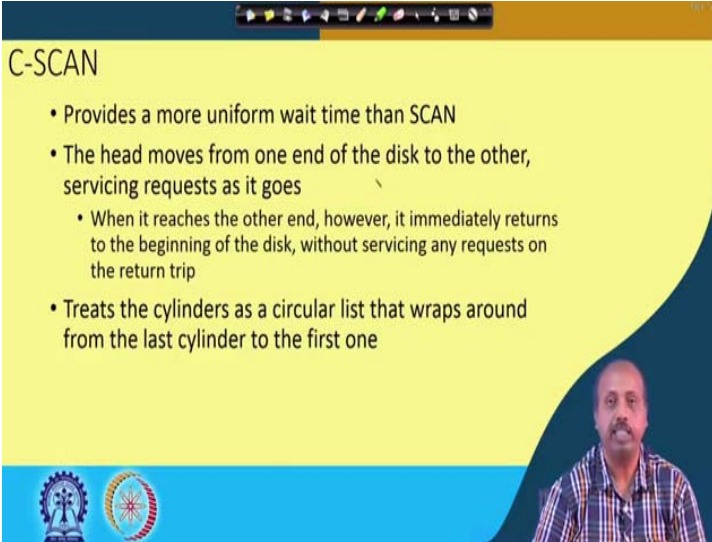
From 0 again it starts going to the other end of the disk and while going the towards other end, now it serves all these requests 65 67 98 122 124 like that. So, see what is happening is that this intermediary requests, so they are being served once in this direction, once in that direction, but see this if you consider say this region. So, if you consider this region, then what happens is that the requests when the head was moving in this direction maybe no requests had come.

So, it has come to this and then it goes like this. When it has left this point in this direction, when it has left maybe after that the requests started coming in this region, but they will not be served till the head has come here and again it has gone to this side. So, now at this time these requests will be served. So, the requests towards the end of the disk, so they are not served very frequently, ok.

So, that is the problem with the this particular policy the SCAN policy because it is going to the end and the disk requests which are close to the end and if this somehow miss the exact timing of the arrival of the disk head, then it will be; it will be getting chance after long time after that after the disk head has come back traversing the entire disk twice, almost twice.

So, another difficulty with this particular algorithm is that if even if there is no requests like see, so this we had a request at 14 here, but after that after 14, so it did not it when to say at a cylinder number 0, but that was not necessary because the cylinder numbers. So, this will be this is useless after in this range is there is no requests, but the head moved till this rather it should have come back from this point onwards, ok. So, that is another possibility and that gives rise to another algorithm which is which will which is known as the look algorithm.

(Refer Slide Time: 17:33)



The slide is titled "C-SCAN" and features a yellow background with a blue and orange border. It contains a list of bullet points describing the C-SCAN disk scheduling algorithm. In the bottom right corner, there is a small video inset showing a man with a mustache, wearing a plaid shirt, speaking. At the bottom left of the slide, there are two circular logos: one with a gear and a person, and another with a gear and a sun.

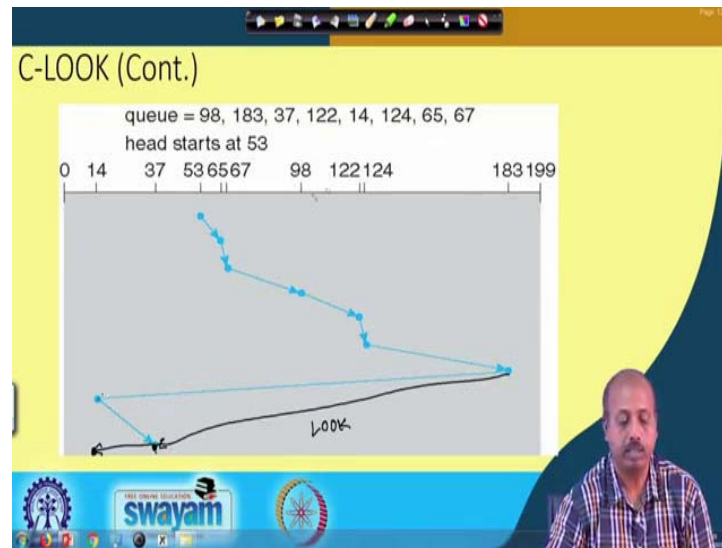
- Provides a more uniform wait time than SCAN
- The head moves from one end of the disk to the other, servicing requests as it goes
 - When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one

So, we will see that. So, we will see that. So, this we will see that there is this another variant of this SCAN algorithm called C-SCAN or Circular SCAN. It provides a more uniform wait time than the SCAN algorithm.

Head moves from one end of the disk to the other end servicing requests as it goes. So, that way the head moves in the similar to SCAN from one end to the other end, but when it reaches the other end, it immediately returns the beginning of the disk without servicing any request on the return trip. So, as if the elevator has gone to the top floor and then it immediately comes back to the ground floor and while it is coming down. So, it does not solve any request. So, similarly here the disk head once it reaches the last cylinder, then it comes back to the first cylinder without servicing any request on the path.

So, it treats cylinders as circular list that wraps around from the last cylinder to the first one. So, after the last cylinder the next cylinders in is the first cylinder as a because the way it is designed. So, the head does not solve any requests. So, it simply comes back to the first cylinder. So, it is like a circular situation.

(Refer Slide Time: 18:55)



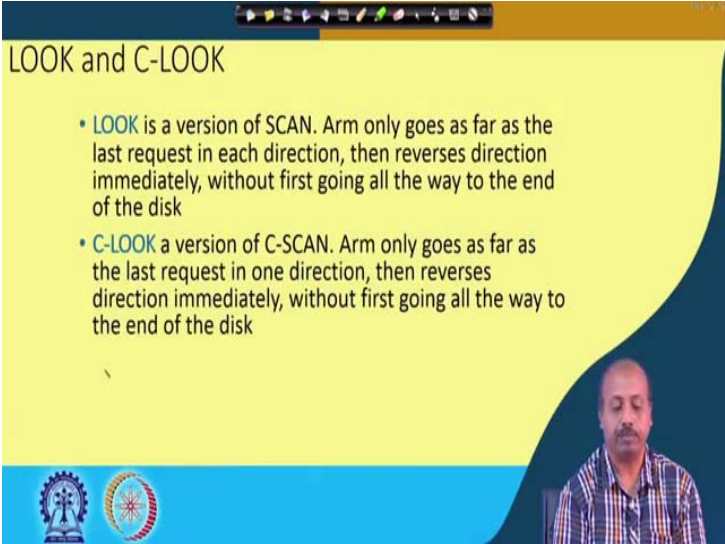
So, this is the C-SCAN algorithm which started at 53, it went like this then which was going towards the higher side of the disk. So, it served all these request, but when it reached here, then it immediately came back to the cylinder number 0 and started servicing again from here.

So, it starts servicing from cylinder 0, going towards the highest possible cylinder and from there it comes back and it comes to the first cylinder without servicing any request in between. So, this is the circular scan type of policy. Of course, you can argue that ok. The amount of disk movement head movement is large, but since it is coming back here, so it since this end of the disk. So, this was serviced long back ok.

Because the disk head started from this point, it went to the other end and then it is coming back here. So, in the meantime there may be a large number of requests that accumulated in the lower side of the disk. So, instead of servicing this intermediary request, so if it comes back here and serves this requests, then it is likely that it will be doing more justice to the disk requests and maybe that this intermediary requests. So, that were being served twice ok.

So, that is that will not happen that is not necessary. So, it can come back. So, with that policy actually this Circular SCAN has been designed that it comes back to the first cylinder after servicing the last cylinder.

(Refer Slide Time: 20:32)



LOOK and C-LOOK

- **LOOK** is a version of SCAN. Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk
- **C-LOOK** a version of C-SCAN. Arm only goes as far as the last request in one direction, then reverses direction immediately, without first going all the way to the end of the disk

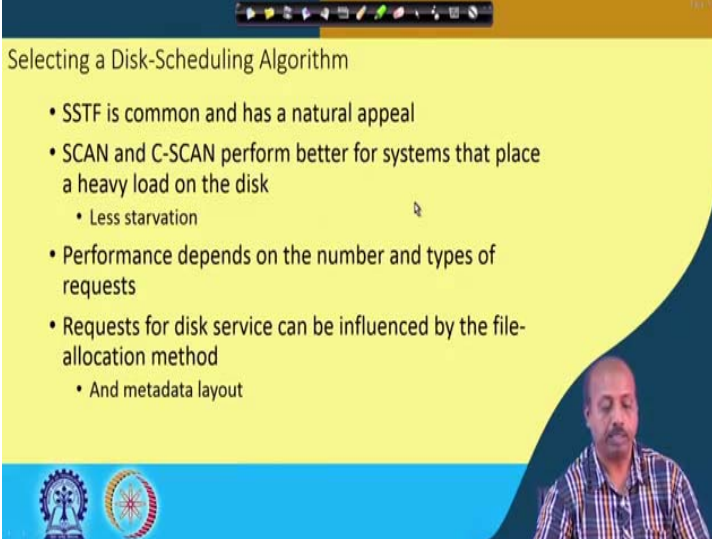
So, another problem with the SCAN algorithm that I was talking about is that even if there is no requests or disk head goes to the extreme end of the disk. So, there is another variant of this SCAN which is known as the LOOK algorithm. So, this arm only goes as far as the last request in each direction and then reverses direction immediately without first going all the way to the end of the disk.

So, it is like this. So, this is the Circular LOOK. So, this is going from here and then after it has reached the last after it has come to the highest one, it does not go to the since in this region there is no request. So, it does not go to the end of the disk. So, rather it comes back and it serves this request and then it comes here. So, whatever wherever is the last trigger, so this is the C-LOOK, Circular LOOK version and the LOOK will be servicing the some intermediary requests if it is there in the process.

But if it is not there, then it will come back. So, this circular, this normal C-LOOK normal look algorithm will be. So, this normal LOOK algorithm will behave like this. So, it will be coming from here it will serve this request and then it will be going there. That will be the normal LOOK algorithm and this is the C-LOOK and C-LOOK actually comes back to the other end till the first request on the other side, ok. So, the least cylinder number on the other side, so it comes to 14 and then goes to 37, but if it is a normal look then it will be it will serve all the request while coming back and you see that this request 37 will be coming first.

So, this will serve this and then it will come to 14. So, these are the basic differences between these Circular LOOK and LOOK. So, the it is better than SCAN in terms of this disk head movement.

(Refer Slide Time: 22:50)



The slide is titled "Selecting a Disk-Scheduling Algorithm" and features a yellow background with a dark blue curved shape on the right side. A video inset in the bottom right corner shows a man with a beard and a plaid shirt speaking. The slide contains the following bullet points:

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk
 - Less starvation
- Performance depends on the number and types of requests
- Requests for disk service can be influenced by the file-allocation method
 - And metadata layout

At the bottom left of the slide, there are two circular logos: one with a gear and a person, and another with a sun-like pattern.

Now, how do you select a disk scheduling algorithm? So, that is very important. So, SSTF is common and has got a natural appeal because we are going for Shortest Seek Time First. So, this shortest seek time means that it will be trying to minimize the seek time. So, just like SJF it is expected that it is it will give you better result.

But we have seen that SSTF may not give always the better result. SCAN and C-SCAN they perform better for systems that place a heavy load on the disk because this SCAN, so it will be servicing from one end to the other end and Circular SCAN will also do similar to that. So, if there is a more heavy load on the disk, then it will be going properly and the starvation is less because um because this it will be servicing all the requests. So, possibility of starvation is less.

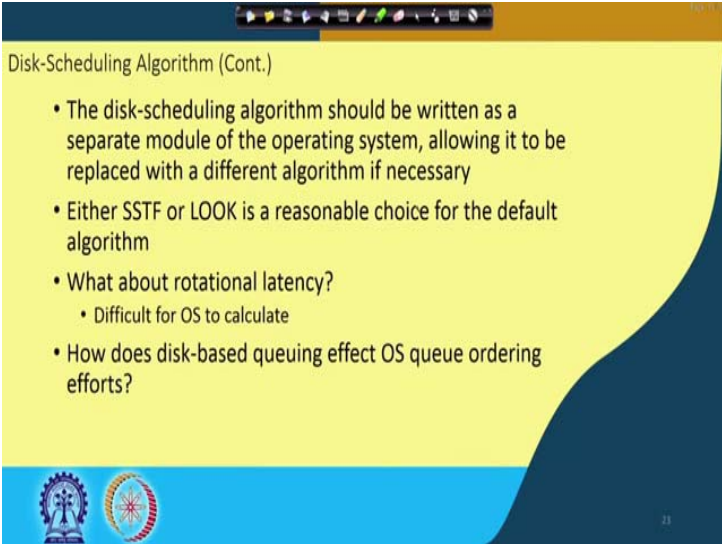
Performance depends on the number and types of request definitely. So, how many requests are coming and what is the type that read, write and all. So, that way that based on the type it will vary requests for disk service can be influenced by the file allocation method, ok. So, file allocation method like where, so like a file may be very large. A file may be say 1 gigabyte large. Now this the sectors that we have, so they are of limited size say 4 kilobyte.

Now, this 1 gigabyte of file space, so that is divided into 4 gigabyte blocks. Now where are those blocks located in the disk ok? So, how are this file how were this disk blocks located? Now when a program is accessing this particular file whether it accesses the file- blocks in a sequential order, so if it is accessing in the sequential order, then the way the blocks have been organized in the disks so that will have a role to play about this access time because the seek time will vary accordingly. And if it is not accessing them sequentially the program is not accessing them sequentially, then that access pattern has to got has got something to do with this time performance. .

Because then this access pattern will tell like which block will be accessed and as a result this disk cylinder requests. So, they will also vary. So, this is based on this file allocation method and this program behavior that will determine this influence on the disk service and then the metadata layout like how normally the files are organized in terms of directories and index files and all, how were they organized. So, that way the search will proceed and this directory is again a file only.

So, those files have to be accessed. So, that will take time. So, that how are they organized and all, so that will have how are this may how is this metadata organized. So, that will have a role to tell like how this disk cylinder requests will come.

(Refer Slide Time: 25:52)



Disk-Scheduling Algorithm (Cont.)

- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary
- Either SSTF or LOOK is a reasonable choice for the default algorithm
- What about rotational latency?
 - Difficult for OS to calculate
- How does disk-based queuing effect OS queue ordering efforts?

21

The Disk-Scheduling Algorithm should be written as a separate module of the operating system allowing it to be replaced with a different algorithm if necessary. So, normally it

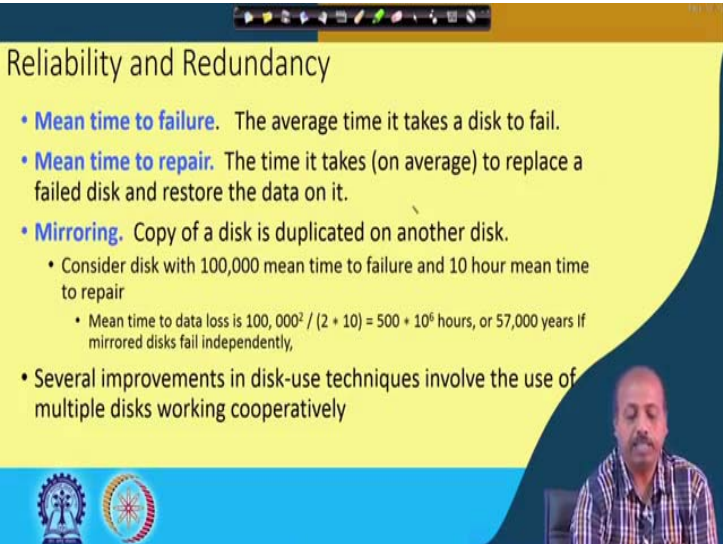
is done like that. So, it is written as a separate module and as and when if it is required that it was following some disk scheduling policy. So, instead of that it should follow some other policy.

So, that has to be taken care of. So, the you should be able to do that either SSTF or look is a reasonable choice for the default algorithm rotational latency. So, it is difficult for OS to calculate because how much time it will go need to go to a particular sector. So, that will determine that and that is only the rotational delay. So, rotational delay is much less compared to this mechanical movement of the disk head. So, that way this rotational latency is often ignored and we can take it on an average half of the rotation time, ok.

So, that way the rpm is available revolutions per minute for the disk plates. So, you can take it half rotation as the average of this rotational latency and how does this disk based queuing effect OS queue ordering efforts. So, that definitely this and so disk based queuing is. So, this will also effect this OS queuing because that will be when there are a number of request for disk access. So, all those processes are blocked.

So, this disk queue service, so that will also determine how the processes will move from blocked state to the ready state. So, that ready queue management, so that will also be dependent on this disk head.

(Refer Slide Time: 27:32)



Reliability and Redundancy

- **Mean time to failure.** The average time it takes a disk to fail.
- **Mean time to repair.** The time it takes (on average) to replace a failed disk and restore the data on it.
- **Mirroring.** Copy of a disk is duplicated on another disk.
 - Consider disk with 100,000 mean time to failure and 10 hour mean time to repair
 - Mean time to data loss is $100,000^2 / (2 * 10) = 500 * 10^6$ hours, or 57,000 years if mirrored disks fail independently,
- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively

The slide features a yellow background with a blue footer containing two circular logos. A small video inset in the bottom right corner shows a man with a mustache, wearing a plaid shirt, speaking.

Talking in terms of reliability and redundancy. So, the because this data in the secondary storage so they are going to stay for a long period of time. So, if there is a data corruption or so that has to be; that has to be taken care of and also during the transfer of data from this main memory to the secondary storage. So, there may be some failures while writing some writing on to the disk and all.

So, they are to be taken care of. So, we have got certain parameters in that direction. One is the meantime to failure - the average time it takes a disk to fail, then meantime to repair the time it takes to replace a failed disk and restore the data on it. So, this depends on the technology that you are using. So, a major concept is known as mirroring. So, a copy of a disk is duplicated on another disk. So, it is, so if I have got a disk with 100,000 meantime to failure and 10 hour meantime to repair, then the meantime to data loss is $100,000 \text{ square divided by } 2 \text{ into } 10$.

So, that is above 57000 years if mirror disk failed independently. So, assuming that both the disk has failed, so that probability is pretty less. So, mirroring actually improves the system performance. This disk redundancy reliability power values and the several improvements in the disk use techniques, involve the use of multiple disk working cooperatively. So, we have got instead of copy keeping one disk, so it keep multiple disks. So, that data may be duplicated or data may be distributed across this disk, so that the possibility of this error you will become less possibility of failure will become less and this mean time to failure will be pretty high, ok. So, we will continue with this in the next class.