DISK SCHEDULING

- For magnetic disks, the access time has two major components. The **seek time** is the time for the disk arm to move the heads to the cylinder containing the desired sector. The **rotational latency** is the additional time for the disk to rotate the desired sector to the disk head.
- The 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.
- We can improve both the access time and the bandwidth by managing the order in which disk I/O requests are serviced.
- If the desired disk drive and controller are available, the request can be serviced immediately. If the drive or controller is busy, any new requests for service will be placed in the queue of pending requests for that drive.
- For a multiprogramming system with many processes, the disk queue may often have several pending requests. Thus, when one request is completed, OS chooses which pending request to service next.
- several disk-scheduling algorithms can be used

1. FCFS Scheduling
2. SSTF Scheduling
3. SCAN Scheduling
4. C-SCAN Scheduling
5. LOOK Scheduling
6. C-LOOK scheduling

FCFS Scheduling

- The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. This algorithm is intrinsically fair, but it generally does not provide the fastest service.
- Consider, for example, a disk queue with requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67 in that order. If the disk head is initially at cylinder 53, then FCFS works as below

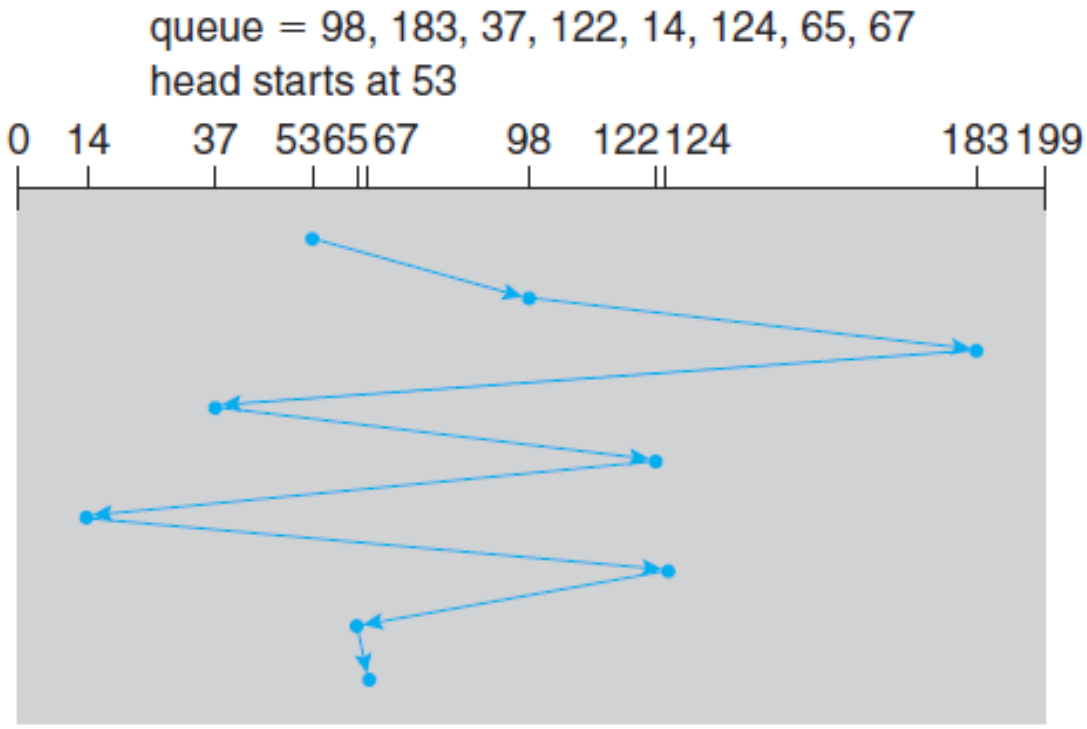

Figure 10.4 FCFS disk scheduling.

- It will first move from 53 to 98, then to 183, 37, 122, 14, 124, 65, and finally to 67

- Total head movement can be calculated as follows: (98-53) + (183-98) + (183-37) + (122-37) + (122-14) + (124-14) + (124-65) + (67-65) = **640 cylinders.**
- The wild swing from 122 to 14 and then back to 124 illustrates the problem with this schedule. If the requests for cylinders 37 and 14 could be serviced together, before or after the requests for 122 and 124, the total head movement could be decreased substantially, and performance could be thereby improved.

SSTF Scheduling

- It seems reasonable to service all the requests close to the current head position before moving the head far away to service other requests.
- This assumption is the basis for the **shortest-seek-time-first (SSTF) algorithm**.
- The SSTF algorithm selects the request with the least seek time from the current head position.
- In other words, SSTF chooses the pending request closest to the current head position.

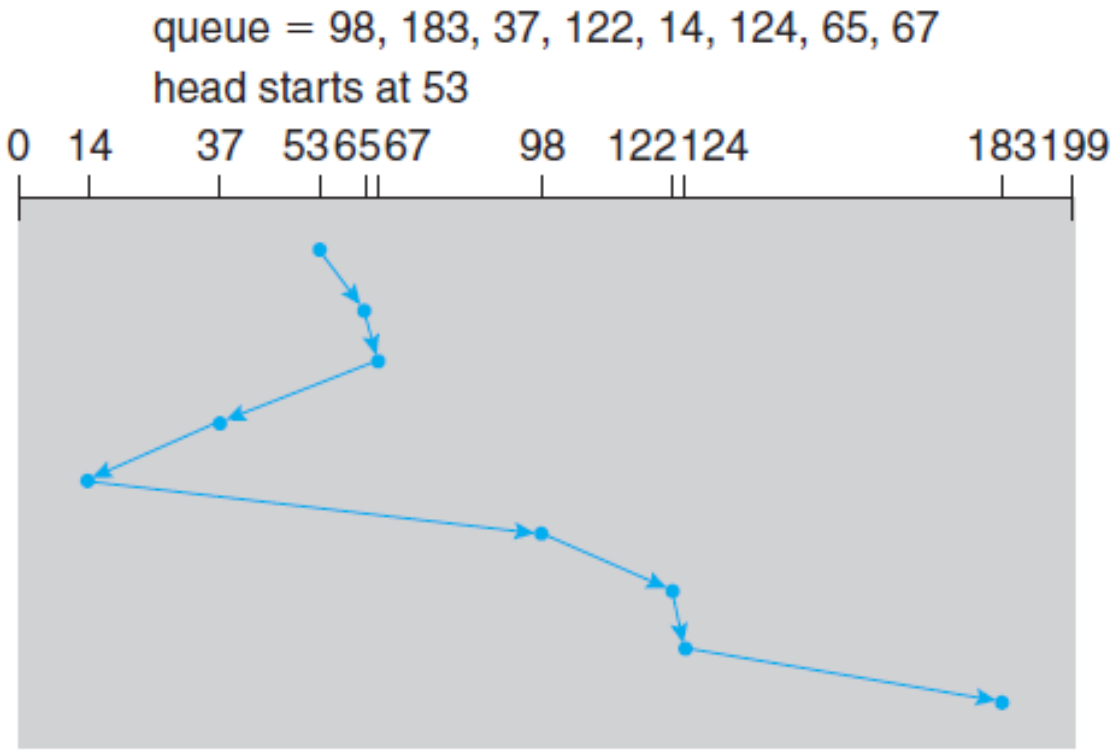

Figure 10.5 SSTF disk scheduling.

- For our example request queue, the closest request to the initial head position (53) is at cylinder 65. Once we are at cylinder 65, the next closest request is at cylinder 67. From there, the request at cylinder 37 is closer than the one at 98, so 37 is served next. Continuing, we service the request at cylinder 14, then 98, 122, 124, and finally 183.
- Total head moves can be calculated as follows: (65-53) + (67-65) + (67-37) + (37-14) + (98-14) + (122-98) + (124-122) + (183-124) = **236 cylinders**
- Clearly, this algorithm gives a substantial improvement in performance.

- SSTF scheduling is essentially a form of shortest-job-first (SJF) scheduling; and like SJF scheduling, **it may cause starvation** of some requests.
- Remember that requests may arrive at any time. Suppose that we have two requests in the queue, for cylinders 14 and 186, and while the request from 14 is being serviced, a new request near 14 arrives. This new request will be serviced next, making the request at 186 wait.
- While this request is being serviced, another request close to 14 could arrive. In theory, a continual stream of requests near one another could cause the request for cylinder 186 to wait indefinitely. This scenario becomes increasingly likely as the pending-request queue grows longer.
- Although the SSTF algorithm is a substantial improvement over the FCFS algorithm, **it is not optimal**. In the example, we can do better by moving the head from 53 to 37, even though the latter is not closest, and then to 14, before turning around to service 65, 67, 98, 122, 124, and 183. This strategy would again reduce the total head movement

SCAN Scheduling

- In the **SCAN algorithm**, the disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk.

- At the other end, the direction of head movement is reversed, and servicing continues.
- The head continuously scans back and forth across the disk.
- The SCAN algorithm is sometimes called the **elevator algorithm**, since the disk arm behaves just like an elevator in a building, first servicing all the requests going up and then reversing to service requests the other way.
- Consider our example. Before applying SCAN, **we need to know the direction of head movement in addition to the head's current position.**
- Assuming that the disk arm is moving toward 0 and that the initial head position is again 53, the head will next service 37 and then 14.
- At cylinder 0, the arm will reverse and will move toward the other end of the disk, servicing the requests at 65, 67, 98, 122, 124, and 183

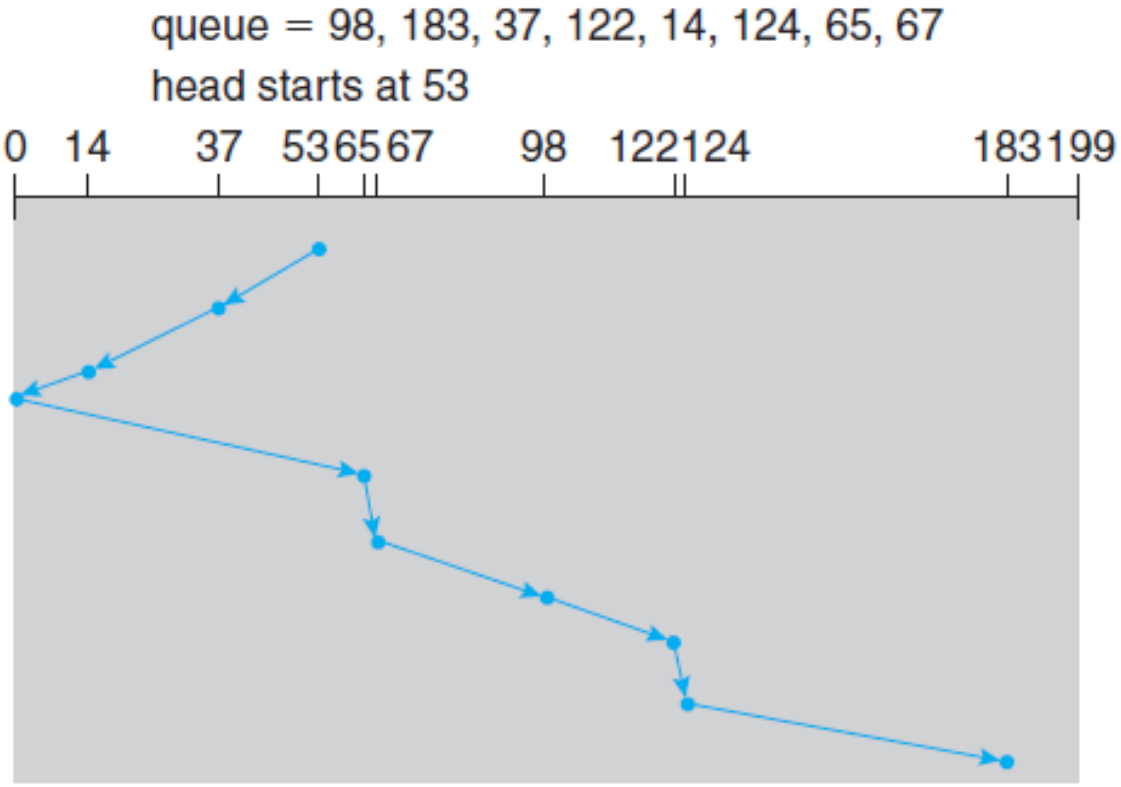

Figure 10.6 SCAN disk scheduling.

- Total head moves: (53-37) + (37-14) + (14-0) + (65-0) + (67-65) + (98-67) + (122-98) + (124-122) + (183-124) = **236 cylinders.**
- If a request arrives in the queue just in front of the head, it will be serviced almost immediately; a request arriving just behind the head will have to wait until the arm moves to the end of the disk, reverses direction, and comes back.
- Assuming a uniform distribution of requests for cylinders, consider the density of requests when the head reaches one end and reverses direction. At this point, relatively few requests are immediately in front of the head, since these cylinders have recently been serviced.

- The heaviest density of requests is at the other end of the disk. These requests have also waited the longest, so why not go there first? That is the idea of the next algorithm.

C-SCAN Scheduling

- **Circular SCAN (C-SCAN) scheduling** is a variant of SCAN designed to provide a more **uniform wait time**.
- Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way.
- When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip.
- The C-SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.
- In our example, assume that the head movement was in upward direction,

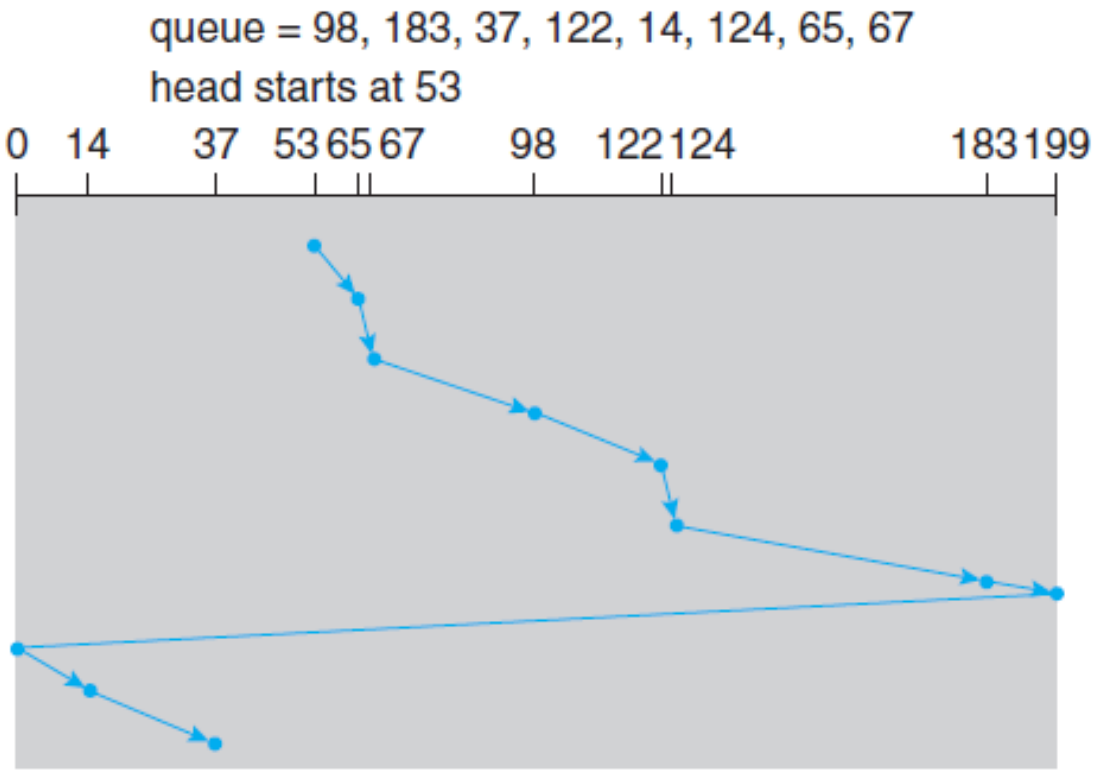

Figure 10.7 C-SCAN disk scheduling.

- **It gives more importance to the average waiting time than the number of head moves.**
- Total head moves include a swing from 199 to 0 also.
- ***(Home work: Assume that the head movement is in downward direction. Find the total head moves in C-SCAN)***

LOOK and C-LOOK Scheduling

- Both SCAN and C-SCAN move the disk arm across the full width of the disk.

- Practically, the arm goes only as far as the final request in each direction. Then, it reverses direction immediately, without going all the way to the end of the disk.
- Versions of SCAN and C-SCAN that follow this pattern are called **LOOK** and **C-LOOK scheduling**, because they ***look*** for a request before continuing to move in a given direction

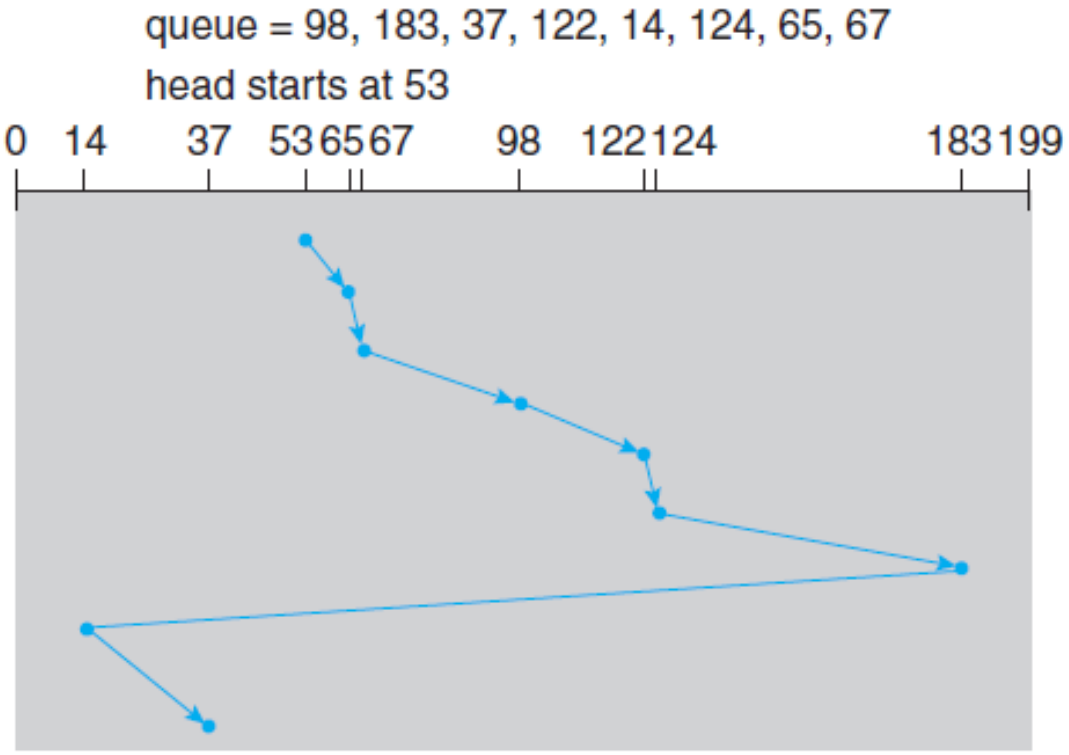

Figure 10.8 C-LOOK disk scheduling.

- ***(Home work: Assume that the head movement is in downward direction. Find the total head moves in LOOK and C-LOOK)***

Selection of a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal because it increases performance over FCFS.
- SCAN and C-SCAN performance depends heavily on the number and types of requests.
- Suppose that the queue usually has just one outstanding request. Then, all scheduling algorithms behave the same.
- Requests for disk service can be greatly influenced by the **file-allocation method**.
- A program reading a contiguously allocated file will generate several requests that are close together on the disk, resulting in limited head movement. A linked or indexed file, in contrast, may include blocks that are widely scattered on the disk, resulting in greater head movement.
- The **location of directories and index blocks** is also important. Suppose that a directory entry is on the first cylinder and a file's data are on the final cylinder. In this case, the disk head has to move the entire width of the disk.
- **Caching** the directories and index blocks in main memory can also help to reduce disk-arm movement.
- **Because of these complexities, the disk-scheduling algorithm should be written as a separate module of OS, so that it can be replaced with a different algorithm if necessary.**

- **Either SSTF or LOOK is a reasonable choice for the default algorithm.**

DISK FORMATTING

- A new magnetic disk is a blank slate
- Before a disk can store data, it must be divided into sectors that the disk controller can read and write. This process is called **low level formatting or physical formatting**
- Low-level formatting fills the disk with a special data structure for each sector.
- The data structure for a sector typically consists of a **header, a data area (usually 512 bytes), and a trailer**.
- The header and trailer contain information used by the disk controller, such as a **sector number and an Error Correcting Code (ECC)**
- When the controller writes a sector of data during normal I/O, the ECC is updated with a value calculated from all the bytes in the data area.
- When the sector is read, the ECC is recalculated and compared with the stored value. If the stored and calculated numbers are different, this mismatch indicates that the data area of the sector has become corrupted and that the disk sector may be bad **(bad sector)**
- The ECC is an *error-correcting* code because it contains enough information, if only a few bits of data have been corrupted, to enable the controller to identify which bits

have changed and calculate what their correct values should be. It then reports a recoverable **soft error**

- The controller automatically does the ECC processing whenever a sector is read or written.
- Most hard disks are low-level-formatted at the **factory** as a part of the **manufacturing process**. This formatting enables the manufacturer to test the disk and to ensure defect-free sectors on the disk.
- The data are is normally 512 bytes. It may be of 256 or 1024 bytes also.
- Before using a physically formatted disk, the OS does two more steps
- The first step is to **partition the disk** into one or more groups of cylinders. The operating system can treat each partition as though it were a separate disk.
- The second step is **logical formatting or creation of a file system**.
- OS stores the initial file-system data structures onto the disk as empty folders
- To increase efficiency, most file systems **group blocks together into larger chunks called clusters**
- Some OS give special programs the ability to use a disk partition as a large sequential array of logical blocks, without any file-system data structures. This array is sometimes called the **raw disk,** and I/O to this array is termed ***raw I/O.***