SECONDARY STORAGE MANAGEMENT

Disk Structure

Disk provides bulk of secondary storage of computer system. The disk can be considered the one I/O device that is common to each and every computer. Disks come in many size and speeds, and information may be stored optically or magnetically. Magnetic tape was used as an early secondary storage medium, but the access time is much slower than for disks. For backup, tapes are currently used.

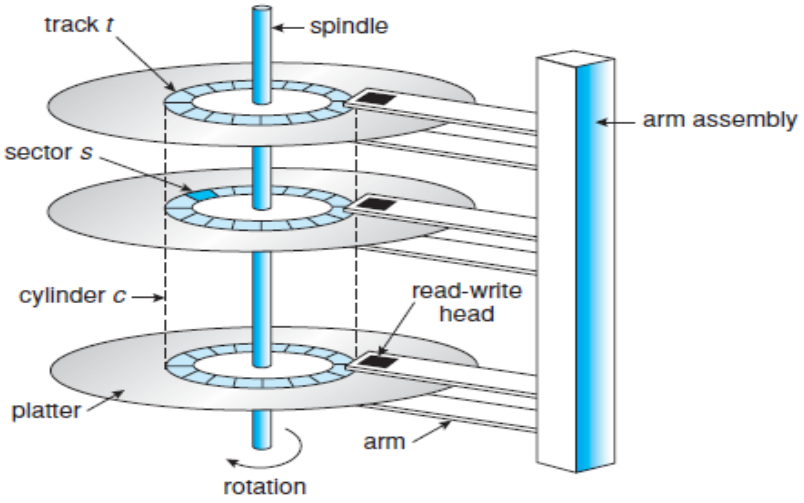
Modern disk drives are addressed as large one-dimensional arrays of logical blocks, where the logical block is the smallest unit of transfer. The actual details of disk I/O operation depend on the computer system, the operating system and the nature of the I/O channel and disk controller hardware.

The basic unit of information storage is a sector. The sectors are stored on a flat, circular, media disk called platter. This media spins close to one or more read/write heads. The heads can move from the inner portion of the disk to the outer portion.

When the disk drive is operating, the disk is rotating at constant speed. To read or write, the head must be positioned at the desired track and at the beginning of the desired sector on that track. Track selection involves moving the head in a movable head system or electronically selecting one head on a fixed head system. These characteristics are common to floppy disks, hard disks, CD-ROM and DVD.

Physical structure of secondary storage device

The figure below shows a physical structure of a secondary storage device.



The basic unit of information storage is a sector. The sectors are stored on a flat, circular, media disk called platter. Each disk platter has a circular shape, like a CD. The two surfaces of a platter are covered with a magnetic material. Information is stored by recording it magnetically on platters.

A read-write head moves just above each surface of every platter. The heads are attached to a **disk arm** that moves all the heads as a unit. The surface of a platter is logically divided into circular **tracks,** which are subdivided into **sectors.** The sets of tracks that are at one arm position makes up a **cylinder.** There may be thousands of concentric cylinders in disk drive, and each track may contain hundreds of sectors.

Disk Performance Parameters

When the disk is in use, a drive monitor spins it at high speed, specified in terms of rotations per minute. Disk speed is determined by two measures, namely; **transfer rate** and **positioning time /random access time. Transfer rate** is the rate at which data flow between the drive and the computer. **Positioning time** consists of two parts; **Seek time and rotational latency. Seek time** is the time necessary to move the disk arm to the desired cylinder while **rotational latency** is the time necessary for the desired sector to rotate to disk head.

To read or write, the head must be positioned at the desired track and at the beginning of the desired sector on that track. Track selection involves moving the head in a movable-head system or electronically selecting one head on a fixed-head system. On a movable-head system, the time it takes to position the head at the track is known as **seek time.** The seek time consists of two key components: the initial startup time and the time taken to traverse the tracks that have to be crossed once the access arm is up to speed.

When once the track is selected, the disk controller waits until the appropriate sector rotates to line up with the head. The time it takes for the beginning of the sector to reach the head is known as **rotational delay,** or rotational latency. The sum of the seek time, if any, and the rotational delay equals the **access time,** which is the time it takes to get into position to read or write.

Once the head is in position, the read or write operation is then performed as the sector moves under the head; this is the data transfer portion of the operation; the time required for the transfer is the **transfer time.**

Disk Scheduling

One of the responsibilities of the operating system is to use hardware efficiently. For the disk drives, meeting this responsibility entails having fast access time and large disk bandwidth. Access time entails seek time (time for the disk arm to move the heads to the cylinder containing desired sector) and rotational latency (time for the disk to rotate the desired sector to the disk head). Disk bandwidth is the total number of bytes transferred, divided by the total time between first request for service and the completion of last transfer. Both access time and bandwidth can be improved by managing the order in which disk I/O requests are serviced.

Whenever a process needs I/O to or from the disk, it issues a system call to the operating system. The request specifies the following information;

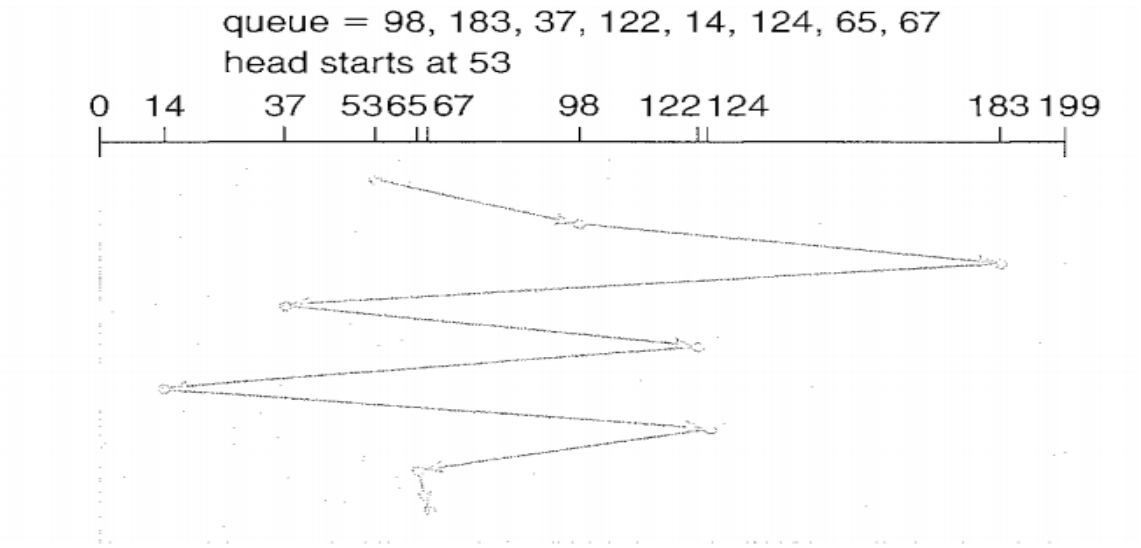
- Type of operation, i.e. input or output operation
- Disk address for the transfer (where in disk to read or write)
- Memory address for the transfer (where in memory to read or write)
- Number of sectors to be transferred

Once this information is specified (i.e. a request is made), and 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. In a multiprogramming system with many processes, the disk queue may often have several pending requests. Therefore, when one request is completed, the operating system chooses which pending request to service next. How does the operating system make this choice? It is at this point disk scheduling algorithms are used.

Disk scheduling algorithms

1. **FCFS Scheduling**

First-come, first-served disk scheduling algorithm, is of course the simplest algorithm. This algorithm selects the next request in the order of arrival. It is intrinsically fair, but it generally does not provide the fastest service. For example, consider a disk queue with requests for I/O to blocks on the following cylinders, 98, 183, 37,122,14,124,65,67. Assuming the disk head is initially at cylinder 53. Using FCFS algorithm, the disk head will first move from 53 to 98, then 183, 37, 122, 14, 124, 65, and finally 67 in that order, for a total head movement of 640 cylinders as shown below.

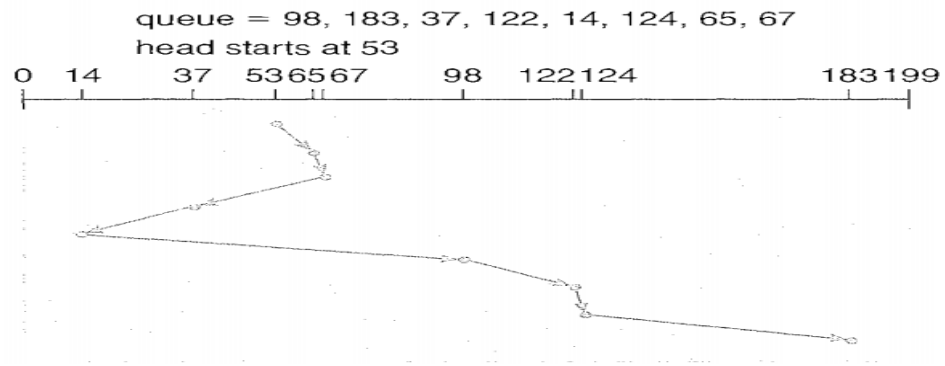


The wild swing from 122 to 14 and then back to 124 illustrates the problem with this schedule. If the requests for cylinders 37 and 14 could be serviced before or after the requests for 122 and 124, the total head movement could be decreased substantially, and performance could thereby improve.

2. 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 basis for the **shortest-seek-time-first (SSTF) algorithm.** It selects the request with the least seek time from the current head position. Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the head position.

For example, using the queue in the example above, the closest request to the initial head position (53) is at cylinder 65. From 65 the next closest is 67 followed by 37, since is closer than 98. The next served is 14, then 98, 122, 124 and finally 183 as shown below.



It results in a total head movement of only 236 cylinders – little more than one-third of distance needed for FCFS scheduling. Clearly this gives a substantial improvement in performance. It however, may cause starvation of some requests.

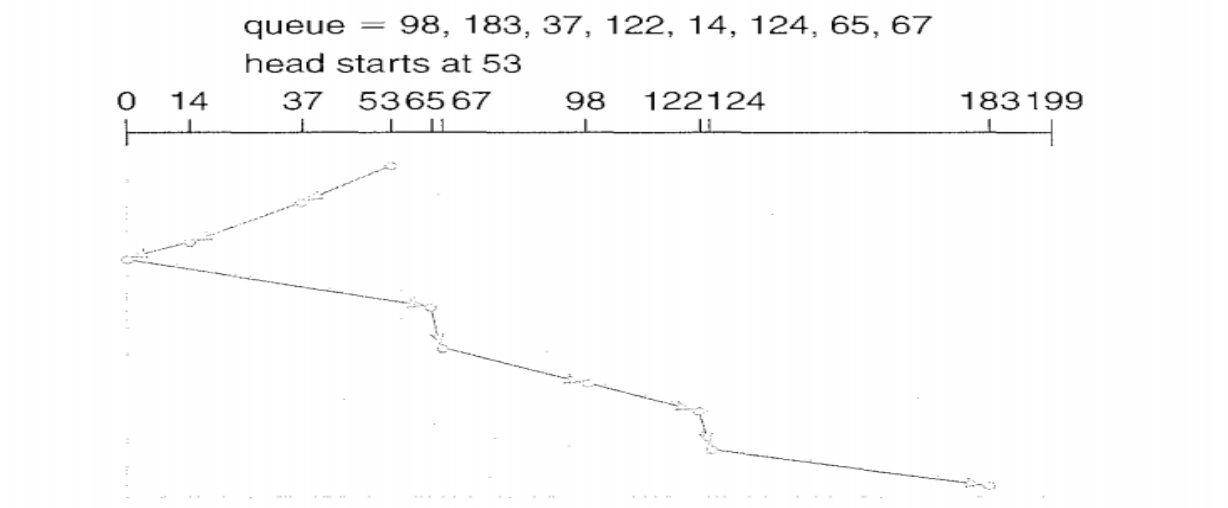
NB: requests arrive at any time. Suppose, while servicing request 14, a new request near 14 arrives. The new request will be serviced hence making request at 183 wait. If another close to 14 arrives, then 183 may wait indefinitely.

Although it improves FCFS substantially, it is not optimal. For example, if from 53 we serve 37 even though it is not closest, then to 14, 65, 67, 98, 122, 124, and finally 183, we could do better since the strategy reduces the total head movement to 208 cylinders. This is the idea of the next algorithm.

3. SCAN Scheduling

In this 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. It is also called **elevator algorithm** since the disk arm behaves like an elevator, first servicing all requests going up and then reversing to service requests the other way.

Considering our similar example, request queue remains 98, 183, 37, 122, 14, 124, 65, and 67. We need to know the direction of head movement and current position of the head. Suppose the head is again at cylinder 53 and moving towards cylinder 0. The head will next service 37 then 14. At cylinder 0, the arm will reverse and will move towards the other end servicing requests at 65, 67, 98, 122, 124, and finally 183 as shown below.

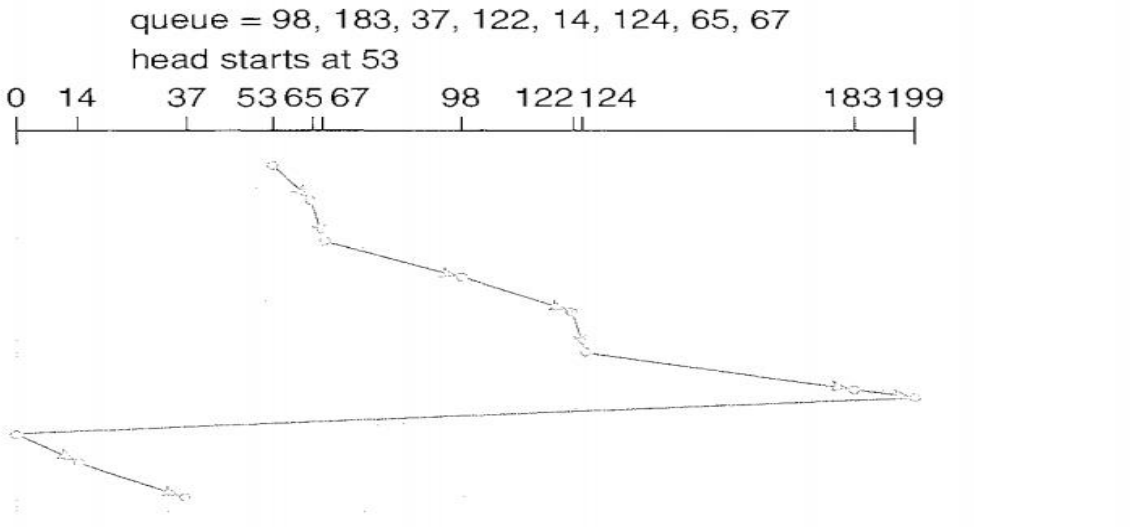


If a request arrives in the queue just in front of the head, it will be serviced almost immediately. However, if a request arrives just behind the head, it will have to wait until the head moves to the end of the disk, reverses direction, and comes back.

If we consider density of requests, few requests are immediately in front of head since most cylinders have been serviced. The heaviest density of requests is at the other end of the disk. Again, they have waited the longest and therefore, why not go there first? This forms the idea of the next algorithm.

4. C-SCAN Scheduling

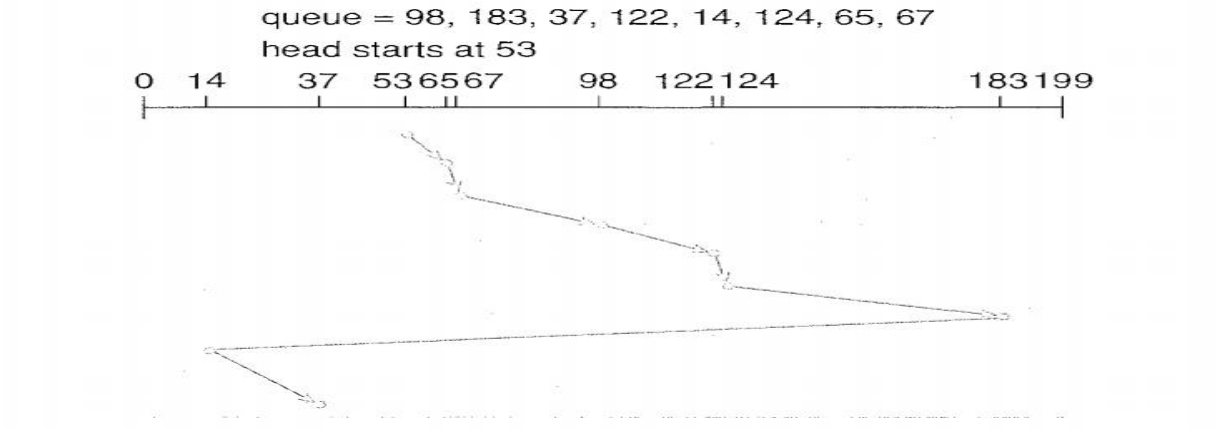
Circular SCAN (C-SCAN) algorithm is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip as shown below. It essentially treats the cylinders as a circular list that wraps around from the final cylinder to the final one.



5. LOOK Scheduling

Note that both SCAN and C-SCAN move the disk arm across the full width of the disk. With LOOK scheduling, the arm goes only as far as the final request in each direction. It then reverses the direction immediately without going all the way to the end of disk. Version of SCAN that follows this pattern are called LOOK and C-LOOK scheduling for C-SCAN, because they look for a request before continuing to move in a given direction.

Using similar example, C-LOOK scheduling is illustrated below



Disk management

For disk management, the operating system is responsible for performing a number of tasks. Most important is disk formatting, booting from disk and bad sector recovery.

i. Disk formatting

A new magnetic disk is a blank slate, i.e. it is just a platter of a magnetic recording material. Before a disk can store data, it must be divided into sectors that the disk controller can read and write. This process is referred to as **low-level /physical formatting.** This type of 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 and a trailer. The header and trailer contain information used by the disk controller, such as 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. 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. 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 initialize the mapping from logical block numbers to defect-free sectors on the disk.

Before a disk can be used to hold files, the operating system still needs to record its own data structures on the disk. It does so in two 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. For instance, one partition can hold a copy of the operating system's executable code, while another holds user files. The second step is **logical formatting**, or creation of a file system. In this step, the operating system stores the initial file-system data structures onto the disk. These data structures may include maps of free and allocated space and an initial empty directory.

ii. Booting from Disk

For a computer to start running—for instance, when it is powered up or rebooted—it must have an initial program to run called **bootstrap**. Bootstrap initializes all aspects of the system, from CPU registers to device controllers and the contents of main memory, and then starts the operating system. To do its job, the bootstrap program finds the operating-system kernel on disk, loads that kernel into memory, and jumps to an initial address to begin the operating-system execution. For most computers, the bootstrap is stored in **read-only memory (ROM)**. This location is convenient, because ROM needs no initialization and is at a fixed location that the processor can start executing when powered up or reset. And, since ROM is read only, it cannot be infected by a computer virus. The problem is that changing this bootstrap code requires changing the ROM hardware chips. For this reason, most systems store a tiny bootstrap loader program in the boot ROM whose only job is to bring in a full bootstrap program from disk. The full bootstrap program can be changed easily: a new version is simply written onto the disk. The full bootstrap program is stored in the "boot blocks" at a fixed location on the disk. A disk that has a boot partition is called a **boot disk** or **system disk**.

iii. Bad sector recovery

Because disks have moving parts and small tolerances (recall that the disk head flies just above the disk surface), they are prone to failure. Sometimes the failure is complete; in this case, the disk needs to be replaced and its contents restored from backup media to the new disk. More frequently, one or more sectors become defective. Most disks even come from the factory with **bad blocks**. Depending on the disk and controller in use, these blocks are handled in a variety of ways.

On simple disks, such as some disks with IDE controllers, bad blocks are handled manually. One strategy is to scan the disk to find bad blocks while the disk is being formatted. Any bad blocks that are discovered are flagged as unusable so that the file system does not allocate them. If blocks go bad during normal operation, a special program (such as the Linux badblocks command) must be run manually to search for the bad blocks and to lock them away. Data that resided on the bad blocks usually are lost.

More sophisticated disks are smarter about bad-block recovery. The controller maintains a list of bad blocks on the disk. The list is initialized during the low-level formatting at the factory and is updated over the life of the disk.

Low-level formatting also sets aside spare sectors not visible to the operating system. The controller can be told to replace each bad sector logically with one of the spare sectors. This scheme is known as **sector sparing** or **forwarding**.

A typical bad-sector transaction might be as follows:

• The operating system tries to read logical block *n*.

• The controller calculates the ECC and finds that the sector is bad. It reports this finding to the operating system.

• The next time the system is rebooted, a special command is run to tell the controller to replace the bad sector with a spare.

• After that, whenever the system requests logical block *n*, the request is translated into the replacement sector's address by the controller.