Storage and File Management System

Operating system deals with storage of user information on mass storage, how storage space is managed (allocated, deallocated) and how user access that information through file system. The file system provides the resource abstraction typically associated with secondary storage. Normally processes initiate I/O operations to store and retrieve information from storage devices. For most users, the file system is the most visible aspect of an operating system. It provides the mechanism for on-line storage and access to both data and programs of the operating system and all the users of the computer system. The file system permits users to create files for long-term existence, with desirable properties like, sharable between processes and having an appropriate internal structure for specific application.

File Management System

A file management system is a set of system software that provides services to users and applications in the use of files. Typically, the only way that a user or application may access files is through the file management system. A file system poses two quite different design problems. The first problem is defining how the file system should look to the user. This task involves defining a file and its attributes, the operations allowed on a file, and the directory structure for organizing files. The second problem is creating algorithms and data structures to map the logical file system onto the physical secondary-storage devices. Normally any software with reasonable functionality and complexity is designed by having a hierarchical and modular structure. Architecture of file management system in Figure-1 is an example of a layered design. Each level in the design uses the features of lower levels to create new features for use by higher levels.

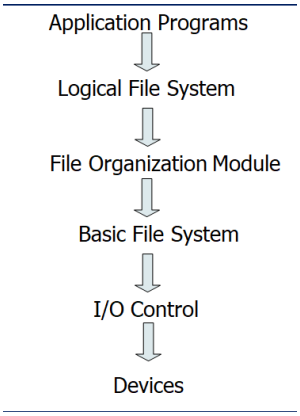

Figure-1: File System Architecture

At the lowest level, device drivers communicate directly with I/O devices or their controllers. The I/O control level consists of device drivers and interrupt handlers to transfer information between the main memory and the disk system. A device driver is responsible for starting I/O operations on a device and processing the completion of an I/O request. A device driver can be thought of as a translator. Its input consists of high level commands such as “retrieve block 123.” Its output consists of low-level, hardware-specific instructions that are used by the hardware controller, which interfaces the I/O device to the rest of the system. The device driver usually writes specific bit patterns to special locations in the I/O controller’s memory to tell the controller which device location to act on and what actions to take.

The next level is the basic file system, or the physical I/O level. The basic file system needs only to issue generic commands to the appropriate device driver to read and write physical blocks on the disk. Each physical block is identified by its numeric disk address (for example, drive 1, cylinder 73, track 2, sector 10). This layer also manages the memory buffers and caches that hold various file systems, directory, and data blocks. A block in the buffer is allocated before the transfer of a disk block can occur. When the buffer is full, the buffer manager must find more buffer memory or free up buffer space to allow a requested I/O to complete. Caches are used to hold frequently used file-system metadata to improve performance, so managing their contents is critical for optimum system performance.

The file-organization module knows about files and their logical blocks, as well as physical blocks. By knowing the type of file allocation used and the location of the file, the file-organization module can translate logical block addresses to physical block addresses for the basic file system to transfer. Each file's logical blocks are numbered from 0 (or 1) through N. Since the physical blocks containing the data usually do not match the logical numbers, a translation is needed to locate each block. The file-organization module also includes the free-space manager, which tracks unallocated blocks and provides these blocks to the file-organization module when requested.

Finally, the logical file system manages metadata information. Metadata includes all of the file-system structure except the actual data (or contents of the files). The logical file system manages the directory structure to provide the file-organization module with the information the latter needs, given a symbolic file name. It maintains file structure via file-control blocks. A file control block (FCB) contains information about the file, including ownership, permissions, and location of the file contents. The logical file system is also responsible for protection.

For file system working and functionality, we describe functions provided at different layers of Figure-1 using bottom-up approach.

Storage Devices

For storage purpose, one of the most used machine-readable devices is magnetic disk. Disks provide most of the secondary storage on which file systems are maintained. We describe structure of most commonly use storage device (disk), ways to perform IO operations and how disk-scheduling algorithms, schedule the order of disk I/O operations.

Magnetic Disks

Bulk of secondary storage for computer systems is provided by magnetic disks. Normally a disk drive consists of multiple platters with a flat circular shape, like a CD. Common platter diameters range from 1.8 to 3.5 inches. The top and bottom surfaces of a platter are covered with a magnetic material. Information is stored by recording it magnetically on the platters. A read–write head "flies" just above each surface of every platter. The heads are attached to a *disk arm* that moves all the heads as a unit as shown in Figure-2. The surface of a platter is logically divided into circular *tracks*, which are subdivided into sectors. The set of tracks that are at one arm position makes up a *cylinder*. There may be thousands of concentric cylinders in a disk drive, and each track may contain hundreds of sectors. The storage capacity of common disk drives is measured in gigabytes.

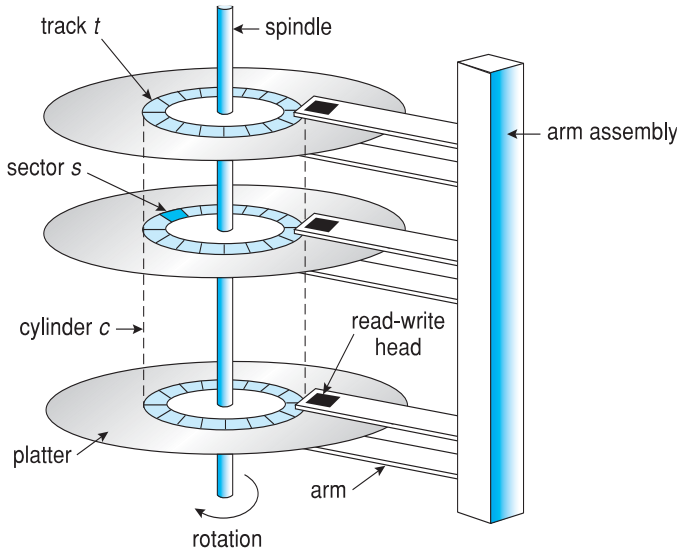

Figure-2: Magnetic disk mechanism

When the disk is in use, a drive motor spins it at high speed. Most drives rotate 60 to 250 times per second, specified in terms of rotations per minute (RPM). Common drives spin at 5,400, 7,200, 10,000, and 15,000 RPM. Disk speed has two parts. The transfer rate is the rate at which data flow between the drive and the computer. The positioning time, or direct-access time, consists of two parts: the time necessary to move the disk arm to the desired cylinder, called the *seek time*, and the time necessary for the desired sector to rotate to the disk head, called the *rotational latency*. Typical disks can transfer several megabytes of data per second, having seek times and rotational latencies of several milliseconds.

A disk drive is attached to a computer by a set of wires called an I/O bus. The data transfers on a bus are carried out by special electronic processors called *controllers*. The host controller is the controller at the computer end of the bus. A *disk controller* is built into each disk drive. To perform a disk, I/O operation, the computer places a command into the host controller, which sends the command via messages to the disk controller, and the disk controller operates the disk-drive hardware to carry out the command. Disk controllers usually have a built-in cache. Data transfer at the disk drive happens between the cache and the disk surface, and data transfer to the host, at fast electronic speeds, occurs between the cache and the host controller.

Disk Structure

Modern magnetic disk drives are addressed as large one-dimensional arrays of physical blocks, where the physical block is the smallest unit of transfer. The size of a logical block is usually 512 bytes, although some disks can be low-level formatted to have a different logical block size, such as 1,024 bytes. The one-dimensional array of logical blocks is mapped onto the sectors of the disk sequentially. Sector 0 is the first sector of the first track on the outermost cylinder. The mapping proceeds in order through that track, then through the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

By using this mapping, we can at least in theory convert a logical block number into an old-style disk address that consists of a cylinder number, a track number within that cylinder, and a sector number within that track. In practice, it is difficult to perform this translation, for two reasons. First, most disks have some defective sectors, but the mapping

hides this by substituting spare sectors from elsewhere on the disk. Second, the number of sectors per track is not a constant on some drives.

I/O Control System

Two main jobs of a computer are I/O and processing. In many of the cases, the main job is I/O. For example, on social media and web, users are mainly browsing web pages to read or enter some information. One of the main functions of operating system in computer I/O is to manage and control I/O operations and I/O devices. In most of the computer systems different types of IO devices are used with varying characteristics and interfaces. Operating systems provide IO services at an abstract level to users and applications by hiding all details of hardware interface. Users access all these IO devices trough an application interface.

Now we describe how IO operations are performed in different ways in different applications and systems. Usually, a device controller is responsible to control IO operations on a device. Operating system pass on IO commands to IO controller which manages IO operations on the device. Mostly following three approaches are used to perform IO operations in a system as shown in Figure-3.

Programmed I/O: The processor issues an I/O command, on behalf of a process, to an I/O module; that process then busy waits for the operation to be completed before proceeding.

Interrupt-driven I/O: The processor issues an I/O command on behalf of a process. There are then two possibilities. If the I/O instruction from the process is nonblocking, then the processor continues to execute instructions from the process that issued the I/O command. If the I/O instruction is blocking, then the next instruction that the processor executes is from the OS, which will put the process in a blocked state and schedule another process. After IO operation is completed, IO module will interrupt, and the process starts execution from the point of IO initiation.

Direct Memory Access (DMA): A DMA module controls the exchange of data between main memory and an I/O module. The processor sends a request for the transfer of a block of data to the DMA module and is interrupted only after the entire block has been transferred.

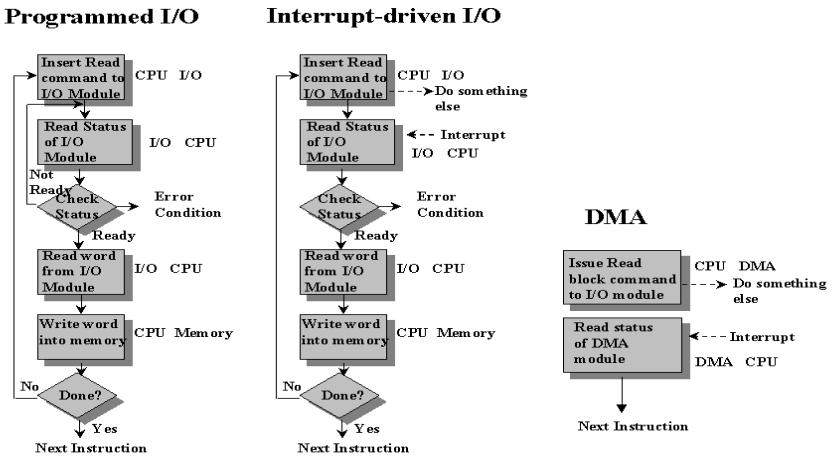

Figure-3: I/O Operations

Disk Scheduling

To use the hardware efficiently, particularly disk drives, disks must have fast access time and large bandwidth. 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. Both access time and the 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 along with information like; operation is input or output, disk and memory address for the transfer, and number of sectors to be transferred. 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, the operating system chooses which pending request to service next. How does the operating system make this choice? Any one of several disk-scheduling algorithms can be used:

- FCFS Scheduling
- SSTF Scheduling
- SCAN Scheduling
- C-SCAN Scheduling
- LOOK Scheduling

Here we describe working of different disk scheduling algorithms using an example of IO request pending in the queue. We assume a disk with 200 tracks and that the disk request queue has random requests in it. For our example, the requested tracks, in the order received by the disk scheduler are 98, 183, 37, 122, 14, 124, 65, 67. Disk head is positioned at 53.

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. For IO requests in the example disk head will first move from 53 to 98, then to 183, 37, 122, 14, 124, 65, and finally to 67, for a total head movement of 640 cylinders as shown in Figure-4. The wild swing from 122 to 14 and then back to 124 illustrates the problem with this scheduling.

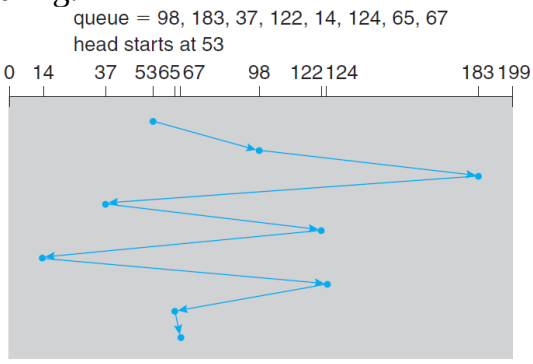

Figure-4: FCFS Scheduling

SSTF Scheduling

The basis for the shortest-seek-time-first (SSTF) algorithm is that it seems reasonable to service all the requests close to the current head position before moving the head far away to service other requests. 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.

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 as depicted in Figure-5. This scheduling method results in a total head movement of only 236 cylinders as compared to 640 cylinders for FCFS scheduling. Clearly, this algorithm gives a substantial improvement in performance. SSTF scheduling may cause starvation of some pending requests.

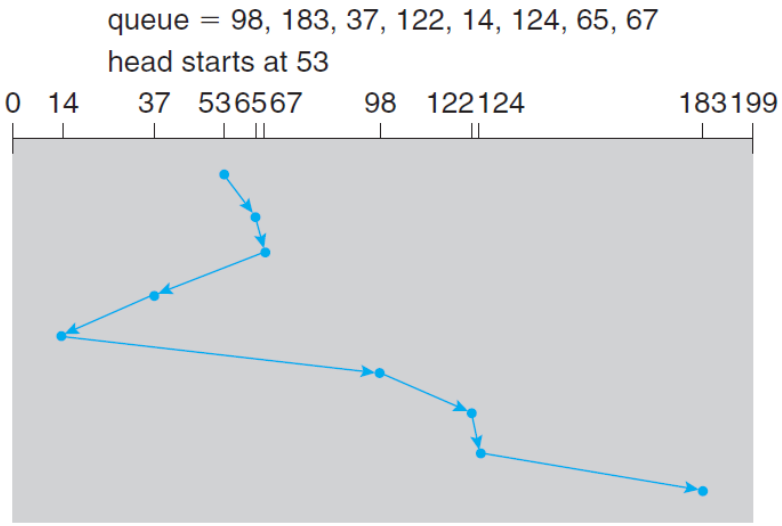

Figure-5: SSTF Scheduling

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.

For our example request queue, we need to know the direction of head movement in addition to the head's current position (at 53). Assuming that the disk arm is moving toward 0, 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 as depicted in Figure-6. 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.

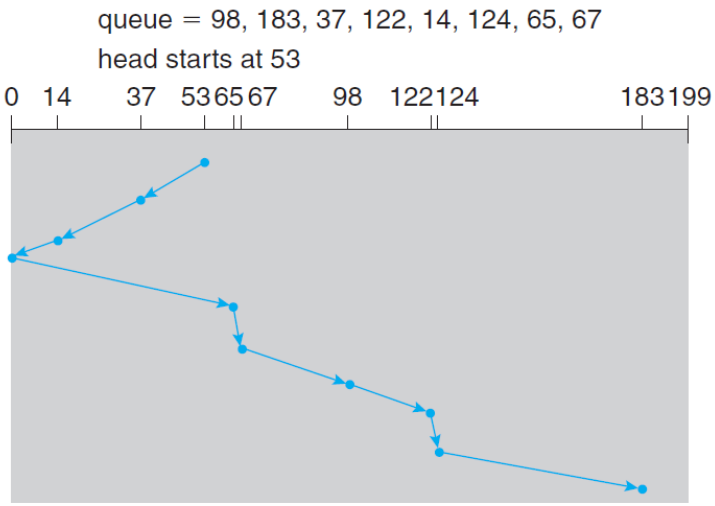

Figure-6: Scan Scheduling

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 as depicted in Figure-7 for request queue of the example. The C-SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

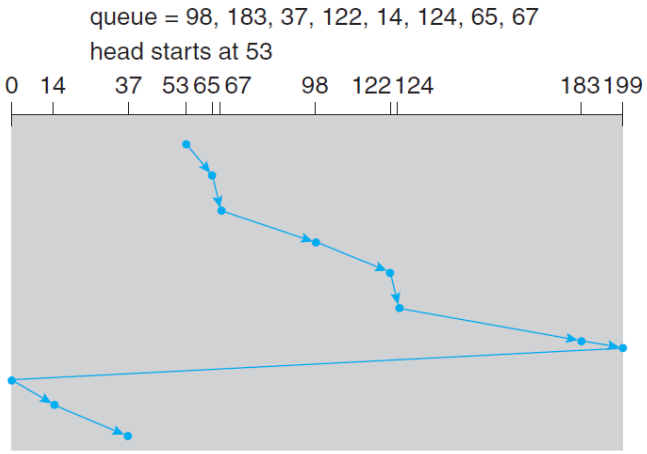

Figure-7: C-Scan Scheduling

LOOK Scheduling

Both SCAN and C-SCAN move the disk arm across the full width of the disk. In practice, neither algorithm is often implemented this way. More commonly, 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 as depicted in Figure-8 for C-Look scheduling using request queue of the example.

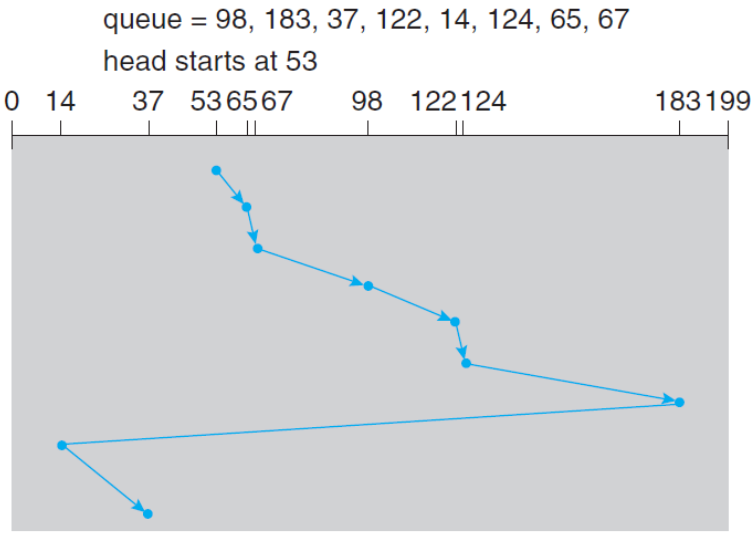

Figure-8: C-Look scheduling

With SSTF, SCAN, and C-SCAN, it is possible that the disk arm may not move for a considerable period of time. For example, if one or a few processes have high access rates to one track, they can monopolize the entire device by repeated requests to that track. High-density multi surface disks are more likely to be affected by this characteristic than lower-density disks and/or disks with only one or two surfaces. To avoid this "arm stickiness," the disk request queue can be segmented, with one segment at a time being processed completely. Two ways to implement this approach are N-step-SCAN and F-SCAN.

N-step-SCAN Scheduling

The N -step-SCAN policy segments the disk request queue into sub-queues of length N. Sub-queues are processed one at a time, using SCAN. While a queue is being processed, new requests must be added to some other queue. If fewer than N requests are available at the end of a scan, then all of them are processed with the next scan. With large values of N, the performance of N -step-SCAN approaches that of SCAN; with a value of N = 1, it works like FCFS scheduling.

F-SCAN Scheduling

F-SCAN Scheduling uses two sub-queues. When a scan begins, all the requests are in one of the queues, with the other one empty. During the scan, all new requests are put into the other queue. Thus, service of new requests is deferred until all the old requests have been processed.

Selection of a Disk-Scheduling Algorithm

Having many disk-scheduling algorithms, how to choose the one which is most appropriate for a computing environment? SSTF is common and has a natural appeal because it increases performance over FCFS. SCAN and C-SCAN perform better for systems that place a heavy load on the disk, because they are less likely to cause a starvation problem. With any scheduling algorithm, performance depends heavily on the number and types of requests.

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. Since every file must be opened to be used, and opening a file requires searching the directory structure, the directories will be accessed frequently. 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 must move the entire width of the disk. If the directory entry were on the middle cylinder, the head would have to move only one-half the width. Caching the directories and index blocks in main memory can also help to reduce disk-arm movement, particularly for read requests.

Because of these complexities, the disk-scheduling algorithm should be written as a separate module of the operating system, so that it can be replaced with a different algorithm if necessary. Either SSTF or LOOK is a reasonable choice for the default algorithm.