UNIT – 4

File and Disk Management (System, Secondary storage structure)

File concept: A file is a name given to a collection of related information that is recorded on secondary storage. From user's perspective, a file is the smallest allotment of logical secondary storage; that is data cannot be written to secondary storage unless they are within a file. Examples of files are program files (both source and object forms) and data. Data files may be numeric, alphabetic, alphanumeric or binary.

The information in a file is defined by its creator. A file has a certain defined structure, which depends on its types. A text file is a sequence of characters organized into lines (and pages). An executable file is a series of code sections that the loader can bring into memory and execute.

File attributes: File attributes vary from one operating system to another but typically consists of the following:

- Name: The symbolic file name in human readable form.
- Identifier: A unique tag, usually a number that identifies the file within the file system.
- Type: This information is used by system that supports different types of files.
- Location: This information is a pointer to a device and to the location of the file on that device.
- Size: The current size of the file (in bytes).
- Protection: Access control information determines who can read, write, execute and so on.
- Time, date, and user identification: This information can be stored for creation, last modification and last use.

File operations: To define a file properly, we need to consider the operations that can be performed on files. The operating systems can provide system calls to create, write, read, reposition, delete, and truncate files.

- Creating files: Two steps are necessary to create a file. First allocate space in the file system for the file, second, an entry for the new file in the directory.
- Writing a file: To write a file, we make a system call specifying both the name of the file and the information to be written to the file. The system keeps a write pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.
- Reading a file: To read from a file, we use a system call that specifies the name of the file and its location. The system needs to maintain a read pointer to the location in the file where the next read is to take place. The read pointer is updated whenever read occurs.
- Repositioning within a file: The current file position pointer is re-positioned to a given value, this file operation is also known as a file seek.
- Deleting a file: In delete file operation. The named file is searched and once found the file system releases all file space, so that it can be reused by other files, and erase the directory entry.
- Truncating a file: This operation erases the contents of a file but keeps its attributes.

File Types

A common technique for implementing file types is to include the type as a part of the file name. The name is split into two parts – a name and an extension, usually separated by a period character. (e.g. Resume.doc, server.java etc.). In this way, the user and the operating system ca recognize from the name alone what the type of a file is.

The system uses the extension to indicate the type of the file and the type of operations that can be done on that file. for example the file with extension .exe can be executed. Application programs also use extension to indicate file types in which they are interested. For example Microsoft word expects its files to end with a .doc extension.

File structure

The files must be created as per the file structure that can be identified and accepted by an underlying operating system. For example every OS is in need of specific file structure for its executable files. The OS has to determine to the location in main memory as to where exactly to load the file to be executed and what could be the address of first instruction in that file in order to commence execution.

Byte stream kind of file structure: Here the file is unstructured collection of bytes. Operating systems such as MS – DOS, Windows and UNIX adopted this approach.

Record stream kind of file structure: Here the file is organized into a sequence of fixed – length records (80 – character records and 132 – character records). A read operation returns one record, and write operation overwrites or appends one record.

Access Methods

File store information that must be accessed and read into computer memory. Some system provides only one access method for files while other systems such as those of IBM, support many access methods.

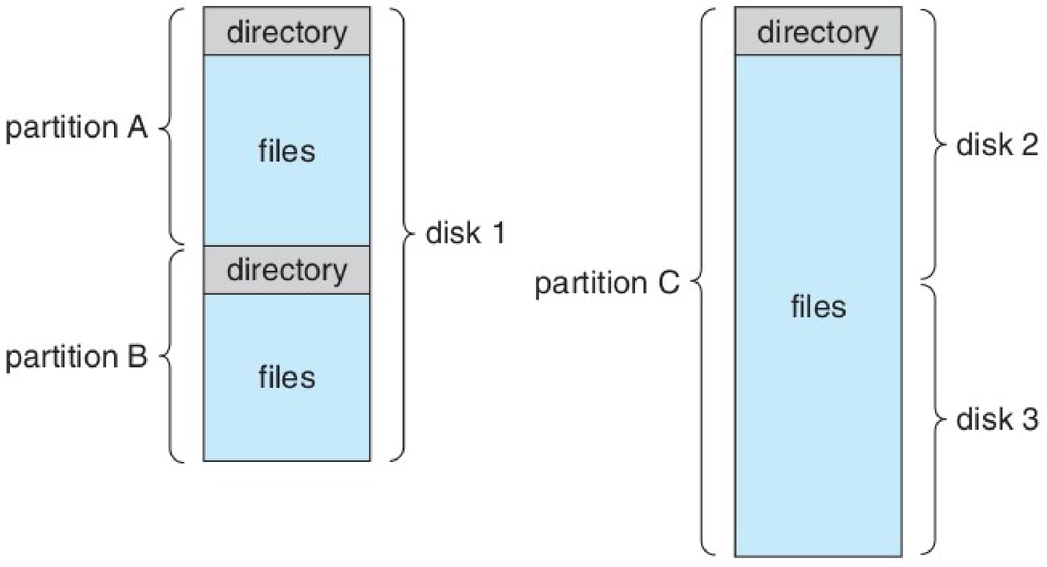
Sequential access: This is the simplest access method. Information in the file is processed in order, one record after the other. For example editors and compilers usually access files in this fashion. A read operation (read next) read the next portion of the file and automatically advances a file pointer. Similarly, the write operation (write next) appends to the end of the file and advances to the end of the newly written data (the new end of file).

Direct access: Direct access is also known as relative access. A file is made up of fixed length logical records that allow programs to read and write records rapidly without any particular order. For direct access, the file is viewed as a numbered sequence of blocks or records. Thus we may read block 14, then read lock 52, and then write block 8. There are no restrictions on the order of reading or writing for a direct access file.

Database is often of this type. When a query for a subject arrives, we compute which block contains the answer and then read that block directly for the required information. For the direct access method, the file operations must be modified to include the block number as a parameter. Thus, we have read n, instead of read next where n is the block number. Similarly we have write n rather than right next.

Directory Structure

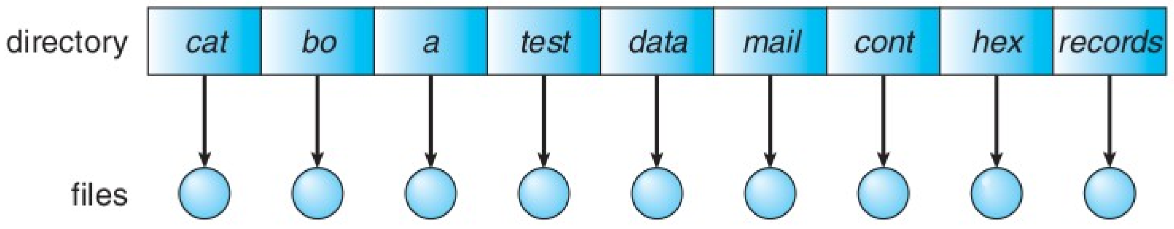
A storage device can be used in it’s entirely or can be subdivided. For example, a disk can be partitioned into quarters, and each quarter can hold a file system. Partitioning is useful for limiting the sizes of individual file systems, putting multiple file system types on the same device, or leaving part of the device for other uses or left unformatted (raw) disk space.



A typical file-system organization.

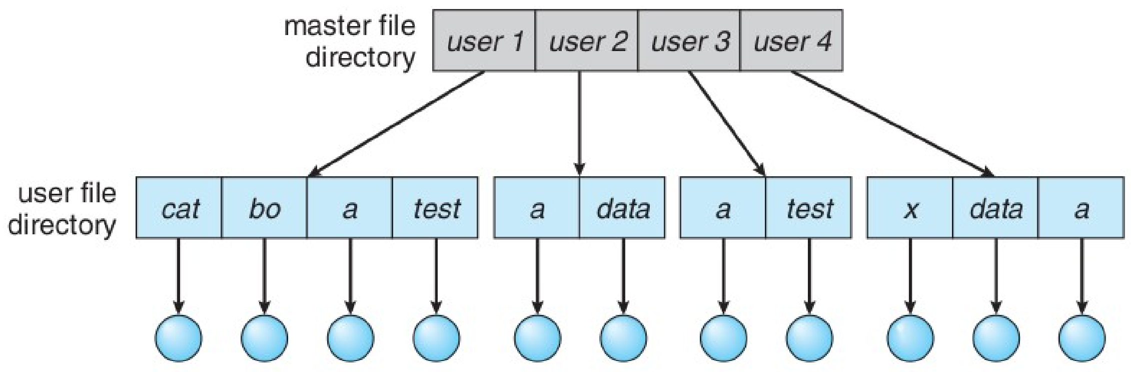
A file system can be created on each of these parts of the disk. Any entity containing of file system is generally known as a volume. A volume may be a subset of a device, a whole device, or multiple devices linked together into a RAID set. Volumes can also store multiple operating systems, allowing a system to boot and run more than one operating system.

Single – level directory: The simplest directory structure is the single level directory. All files are contained in the same directory, which is easy to support and understand. A single – level directory has significant limitations, for example, when the number of files increases or when the system has more than one user. Since all files are in the same directory, they must have unique names. If two users call their data file test, then the unique – name rule is violated.



Single-level directory.

Two-level Directory: In the two level directory structure, each user has his own user file directory (UFD). The UFDs have similar structures, but lists only the files of a single user. When a user logs in, the systems master file directory (MFD) is searched. When a user refers to a particular file, only his own UFD is searched. Thus different users may have files with the same name, as long as each UFD contains files with unique name.



Two-level directory structure.

File System Structure

Disks provide secondary storage on which a file system is maintained. Disks have the following two characteristics that make them a convenient medium for showing multiple files.

1. It is possible to read a block from the disk, modify the block, and write it back into the same place.
2. A disk can access directly any block of information it contains.

File systems provide efficient and convenient access to the disk by allowing data to be stored, located and retrieved easily. The file system is generally composed of many different levels. An example of a layered structure is shown in following figure.

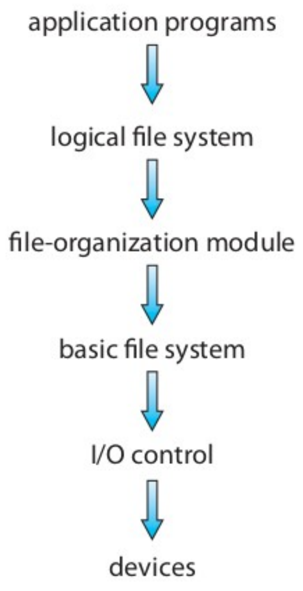


Figure Layered file system.

Each level in the design uses the features of lower levels. The lowest level (I/O control) consist of devices drivers and interrupt handlers to transfer information between the main memory and the disk system.

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 (example drive 1, cylinder 73, track 2, and sector 10).

The file organization module knows about files and their logical block as well as physical blocks. By knowing the type of file allocation used and location of the file, the file organization module can translate logical block address to physical block address for the basic file system to transfer.

The logical file system manages metadata information. Metadata includes all of the file system structure except the actual data. The logical file system manages the directory structure to provide the file organization module with the information for a given symbolic file name. It maintains file structure in the form of file control block. A file control block contains information about the file, including ownership, permissions, and location of the file contents. The logical file system is also responsible for protection and security.

Protection: When information is stored in a computer system, we want to keep it safe from physical and improper access. For a small single – user system, we might provide protection by physically removing the floppy disk and locking them in a desk drawer or file cabinet.

Types of access: Protection mechanism provides controlled access by limiting the types of file access that can be made. Several different types of file operation may be controlled.

- Read: Read from the file.
- Write: Write or rewrite the file.
- Execute: Load the file into memory and execute it.
- Append: Write new information at the end of the file.
- Delete: delete the file and free its space.

The most common approach to the protection problem is to make access dependent on the identity of the user. Different users may need different types of access to a file or directory. The most general scheme is to associate with each file and directory an access – control – list (ACL) specifying user names and the types of access allowed for each user. When a user requests access to a particular file, the operating system checks the access list associated with that file. if that user is listed for the requested access, the access is allowed. Otherwise, a protection violation occurs, and the user job is denied access to the file.

Consistency Semantics

Consistency semantics represents an important criterion for evaluating any file system that supports file sharing. These semantics specify how multiple users of a system are to access a shared file simultaneously. These semantics specify when modification of data by one user will be observable by other users.

UNIX Semantics

The UNIX file system uses the following consistence semantics:

- Writes to an open file by a user are visible immediately to other users who have this file open.
- One mode of sharing allows users to share the pointer of current location into the file. Thus, the advancing of the pointer by one user affects all sharing users.

Session Semantics
The Andraw file system (AFS) uses the following consistency semantics.

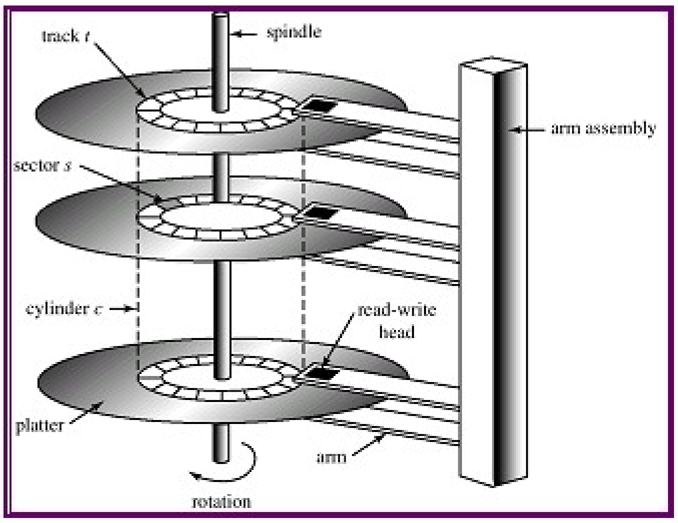
- Writes to an open file by a user are not visible immediately to other users that have the same file open.
- Once a file is closed, the changes made to it are visible only in session started later.

Immutable – shared – files semantics
In immutable shared files once a file is declared as shared by its creator it cannot be modified. An immutable file has two key properties: its name may not be reused, and its contents may not be allowed to change. Thus, the name immutable file signifies that the contents of the file are fixed.

Disk Structure

Magnetic disk provides the bulk of secondary storage for modern computer systems. Each disk platter has a flat circular shape. Common platter diameters range from 1.8 to 5.25 inches. The two surfaces of a platter are coated with 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. The surface of a platter is logically divided into circular tracks, which are subdivided into sectors. The set of tracks at one arm position is called a cylinder. There may be thousands of concentric cylinders in a disk drive, and each track may contain hundreds of sectors.



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 random access time is the time required to move the disk arm to the desired cylinder is 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.

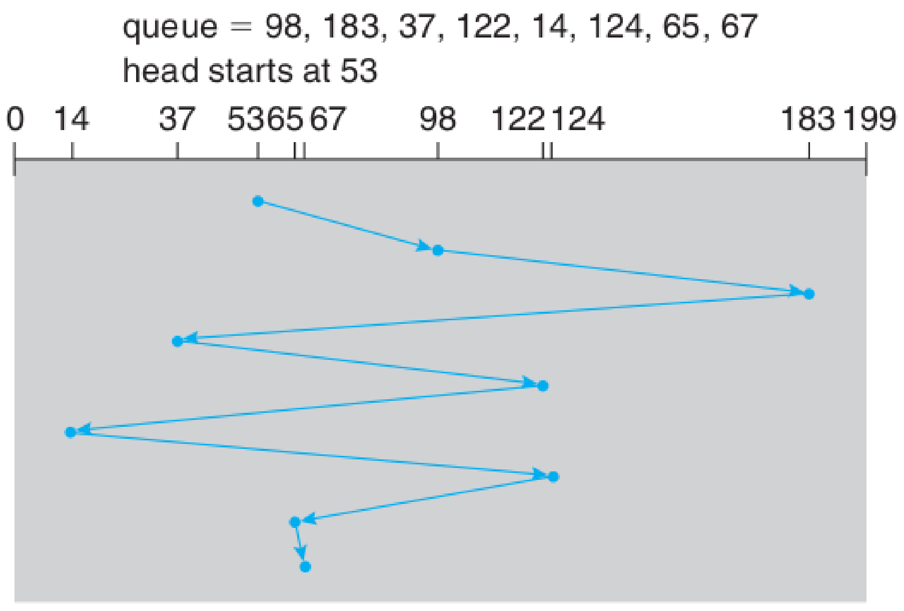
Modern disk are addressed as large one dimensional array of logical blocks. The size of a logical block is usually 512 bytes. The one dimensional array of logical blocks is mapped onto the sectors of the disk sequentially. Sector 0 is the first sector of the first track on the outermost cylinder. The mapping proceeds in order through that track, then through the rest of the tracks in that cylinder, and then through the rest of the cylinder from outermost to inner most.

Disk Scheduling

One of the responsibilities of the operating system is to use the hardware efficiently. We require disk drives with fast access time and large disk bandwidth. The access time has two major components. The seek time is the time for the disk arm to move the head 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 of a service and the completion of the last transfer.

FCFS Scheduling

The first come first served (FCFS) algorithm is the simplest form of disk scheduling. But it generally does not provide the fastest service. Consider for example, a disk queue with request for I/O to blocks on cylinders. 98, 183, 37, 122, 14, 124, 65, 67.



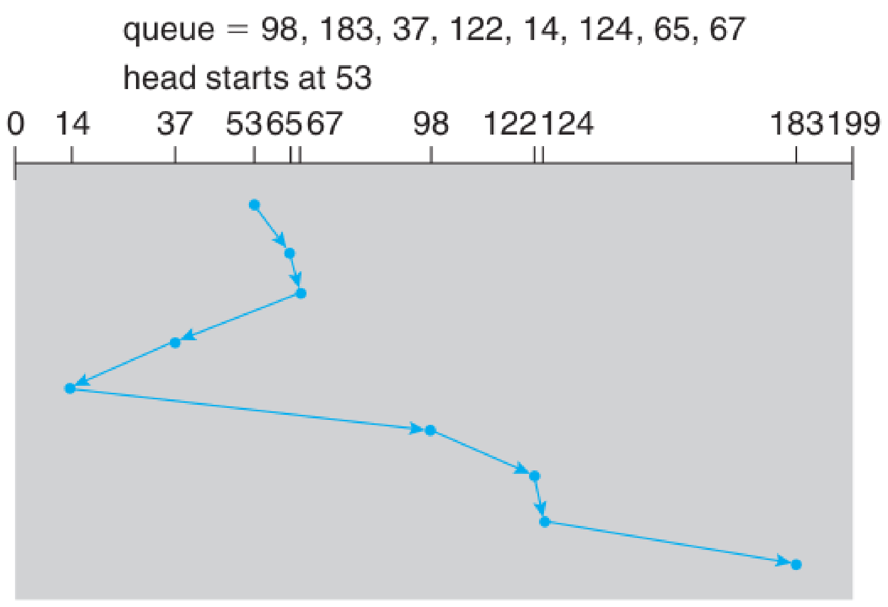
FCFS disk scheduling.

In this order if the disk head is initially at cylinder 53, it will first move from 53 to 98, then to 183, 37, 122, 14, 124, 65 and finally to 67, for a total head movements of 640 cylinders. This schedule is diagrammed in figure.

SSTF Scheduling

In this method, request close to the current head position are serviced before moving the head far away to service other requests. The shortest seek time first (SSTF) algorithm selects the request with the least seek time from 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. This scheduling method results in a total head movement of only 236 cylinders.



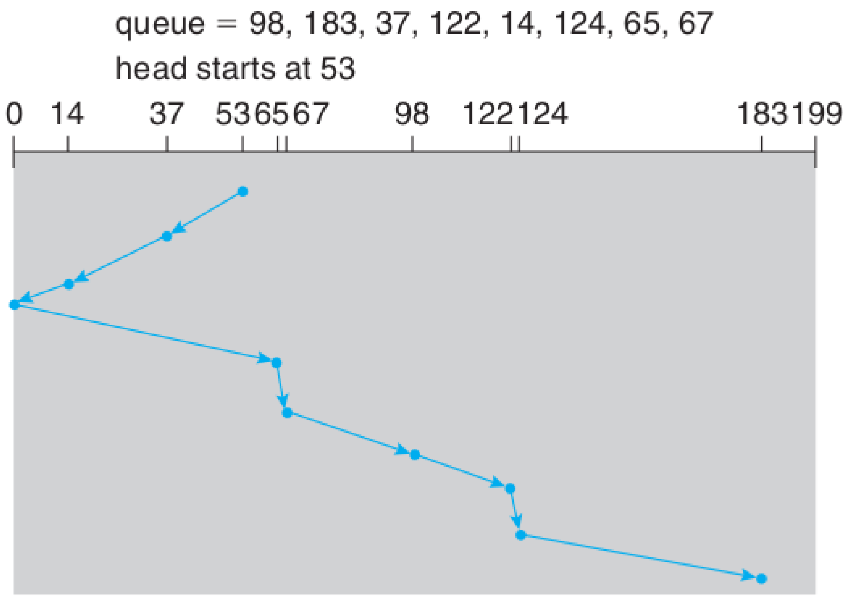
SSTF disk scheduling.

Although SSTF scheduling algorithm gives a substantial improvement in performance, it may cause starvation of some requests. Suppose that we have two requests in the queue for cylinders 14 and 186. Since request may arrive at any time, therefore while the request from 14 is being serviced, a new request near 14 may arrive. This new request will be serviced next making the request at 186 wait. While this request is continued stream of request near one another could cause the request for cylinder 186 to wait indefinitely. Thus SSTF algorithm is a substantial improvement over the FCFS algorithm but it is not optimal.

SCAN Scheduling

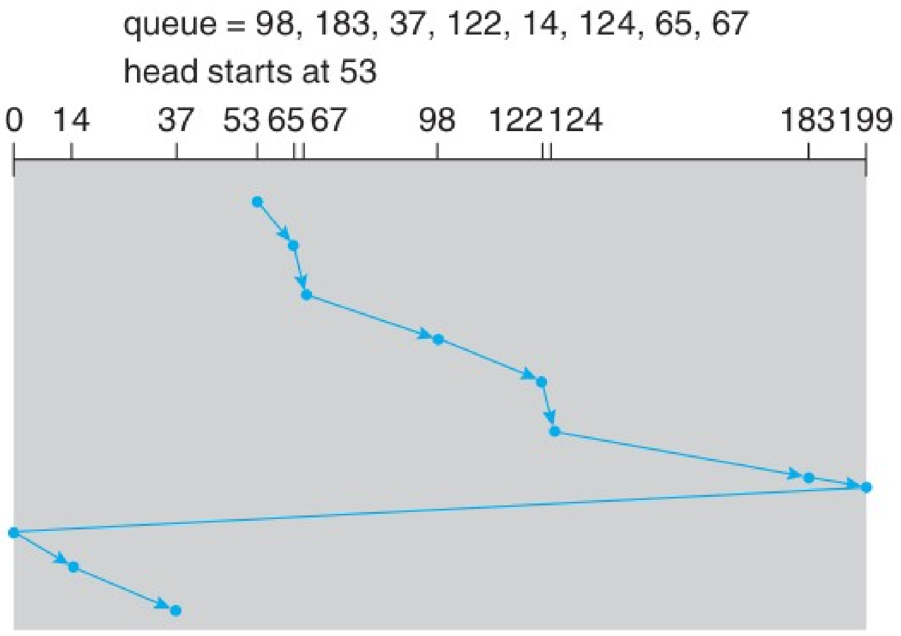
In the scan algorithm, the disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing contains. The head continuously scans back and forth across the disk. The SCAN algorithm is sometimes called the elevator algorithm.

Assuming the direction of disk arm in moving towards 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 towards the other end of the disk, servicing the requests at 65, 67, 98, 122, 124, and 183. 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, reverse direction, and come back.



SCAN disk scheduling.

C-SCAN Scheduling
Circular SCAN (C-SCAN) scheduling is a modification in SCAN to provide a uniform wait time. Like SCAN, C-SCAN moves the head form one end of the disk to other, servicing request along the way. When the head reaches the other end, it immediately returns to the beginning of the disk without servicing any requests on the return trip.



C-SCAN disk scheduling.

LOOK Scheduling

Both SCAN and C-SCAN move the disk arm across the full width of the disk. In practice, neither of these is implemented in this way. Generally, 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. The version 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.

Disk Management

The operating system is responsible for several other aspects of disk management.

Disk formatting

A new magnetic disk is a platter of 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 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 consists of a header, a data area (about 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).

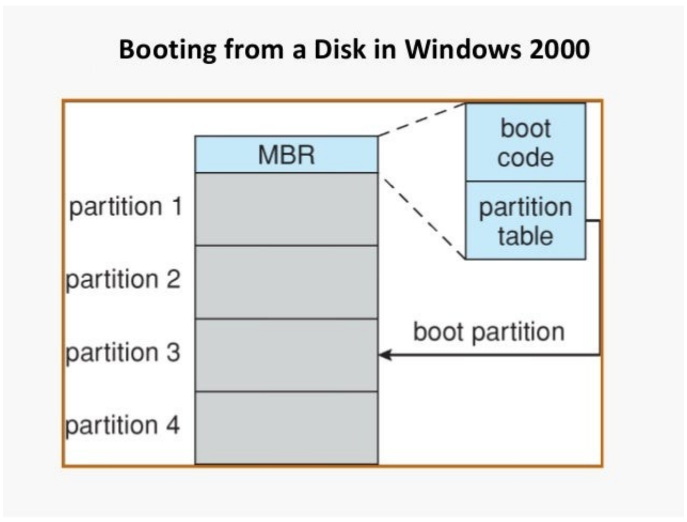
Before a disk can be used to hold files, the operating system still needs to record its own data structure on the disk. It is done in two steps. The first step is to partition the disk into one or more groups of cylinders. The operating system can treat each partition as a separate disk. The second step is logical formatting or creation of a file system. In this step, the operating system stores the initial file system data structure onto the disk. These data structure may include maps of free and allocated space (a FAT or inodes) and an initial empty directory.

Boot blocks

When a computer system is powered up or rebooted it must have an initial program to run. This initial bootstrap program initializes CPU registers, device controllers, then contents of main memory, and then starts the operating system.

In most computer systems, the bootstrap is stored in read only memory (ROM), because ROM needs no initialization and is at fixed location that the processor can start executing when powered up or reset. Since ROM is read only, it cannot be infected by a computer virus.

As an example the windows 2000 system places its boot code in the first sector on the hard disk termed as the master boot record or MBR. Further, windows 2000 allows a hard disk to be divided into one or more partitions, one partition is identified as the boot partition, contains the operating system and device drivers.



Booting begins by running code that is present in the systems ROM memory. This code directs the system to read the boot code from the MBR. The MBR contains a table listing the partitions for the hard disk and a flag indicating which partition to be used for booting. The boot process is illustrated in the above figure.

Swap – space management

Swap space management a low level task of the operating system. Virtual memory uses disk space as an extension of main memory. Since disk access is much slower than memory access, using swap space decreases system performance. The main goal for design and implementation of swap – space is to provide the best throughput for the virtual memory system.

Swap – space use

Swap space is used in various ways by different operating systems, based on memory management algorithm they use. For example, system that implements swapping may use swap space to hold an entire process image, including the code and data segments. Paging systems may simply store pages that have been pushed out of main memory. Therefore the amount of swap space needed by a system can vary from a few megabytes to gigabytes of disk space, depending on the amount of physical memory, the amount of virtual memory it is backing, and the way in which the virtual memory is used.

Swap space location

A swap space can reside in one of two places. It can be designed out of the normal file system or it can be in a separate disk partition. If the swap space is a large file within the file system, normal file system routines can be used to create it, name it, and allocate its space. This approach, though easy to implement but is inefficient.

Alternatively, swap space can be created in a separate raw partition. No file system or directory structure is placed in this space. A separate swap space storage manager is used to allocate and deallocate the blocks for the raw partition.

Some operating systems are flexible and can swap both in raw partitions and in file system space. Linux is an example: the policy and implementation are separate, allowing the administrator to decide which type of swapping to use.