Overview of Physical Storage Media

Several types of data storage exist in most computer systems. These storage media are classified by the speed with which data can be accessed, by the cost per unit of data to buy the medium, and by the medium's reliability. Among the media typically available are these:

• **Cache**. The cache is the fastest and most costly form of storage. Cache memory is small; its use is managed by the computer system hardware. We shall not be concerned about managing cache storage in the database system.

• **Main memory**. The storage medium used for data that are available to be operated on is main memory. The general-purpose machine instructions operate on main memory. Although main memory may contain many megabytes of data, or even gigabytes of data in large server systems, it is generally too small (or too expensive) for storing the entire database. The contents of main memory are usually lost if a power failure or system crash occurs.

• **Flash memory**. Also known as *electrically erasable programmable read-only memory* (*EEPROM*), flash memory differs from main memory in that data survive power failure. Reading data from flash memory takes less than 100 nanoseconds (a nanosecond is 1/1000 of a microsecond), which is roughly as fast as reading data from main memory. However, writing data to flash memory is more complicated—data can be written once, which takes about 4 to 10 microseconds, but cannot be overwritten directly. To overwrite memory that has been written already, we have to erase an entire bank of memory at once; it is then ready to be written again. A drawback of flash memory is that it can support only a limited number of erase cycles, ranging from 10,000 to 1 million.

Flash memory has found popularity as a replacement for magnetic disks for storing small volumes of data (5 to 10 megabytes) in low-cost computer systems, such as computer systems that are embedded in other devices, in hand-held computers, and in other digital electronic devices such as digital cameras.

• **Magnetic-disk storage**. The primary medium for the long-term on-line storage of data is the magnetic disk. Usually, the entire database is stored on magnetic disk. The system must move the data from disk to main memory so that they can be accessed. After the system has performed the designated operations, the data that have been modified must be written to disk.

The size of magnetic disks currently ranges from a few gigabytes to 80 gigabytes. Both the lower and upper end of this range have been growing at about 50 percent per year, and we can expect much larger capacity disks every year. Disk storage survives power failures and system crashes. Disk-storage devices themselves may sometimes fail and thus destroy data, but such failures usually occur much less frequently than do system crashes.

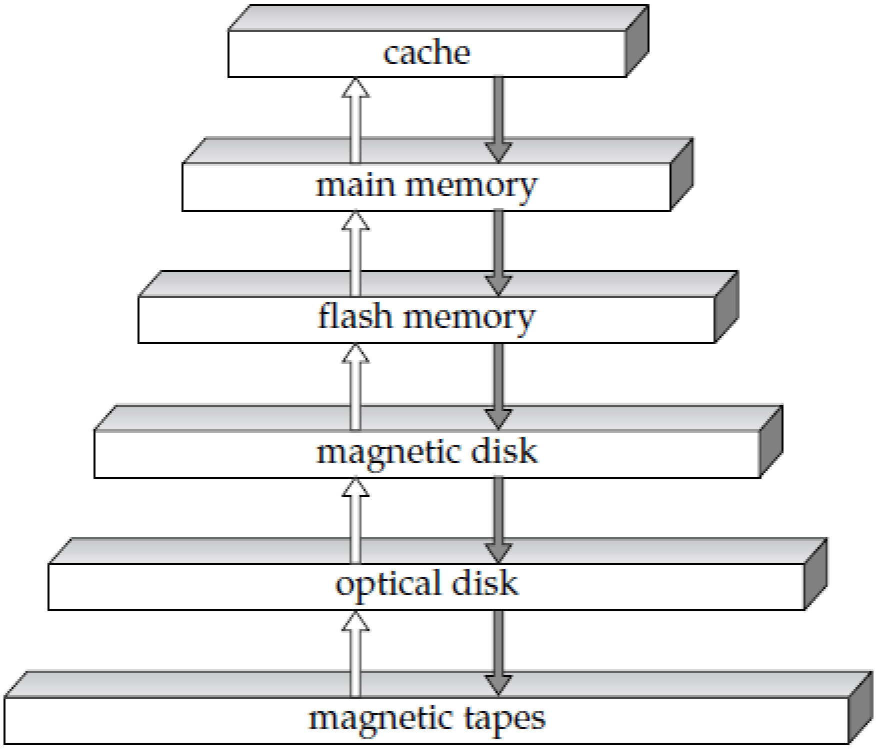
• **Optical storage**. The most popular forms of optical storage are the *compact disk* (CD), which can hold about 640 megabytes of data, and the *digital video disk* (DVD) which can hold 4.7 or 8.5 gigabytes of data per side of the disk (or up to 17 gigabytes on a two-sided disk). Data are stored optically on a disk, and are read by a laser. The optical disks used in read-only compact disks (CD-ROM) or read-only digital video disk (DVD-ROM) cannot be written, but are supplied with data prerecorded.

There are "record-once" versions of compact disk (called CD-R) and digital video disk (called DVD-R), which can be written only once; such disks are also called **write-once, read-many** (WORM) disks. There are also "multiple-write" versions of compact disk (called CD-RW) and digital video disk (DVD-RW and DVD-RAM), which can be written multiple times. Recordable compact disks are magnetic–optical storage devices that use optical means to read magnetically encoded data. Such disks are useful for archival storage of data as well as distribution of data.

• **Tape storage**. Tape storage is used primarily for backup and archival data. Although magnetic tape is much cheaper than disks, access to data is much slower, because the tape must be accessed sequentially from the beginning. For this reason, tape storage is referred to as **sequential-access** storage. In contrast, disk storage is referred to as **direct-access** storage because it is possible to read data from any location on disk.

Tapes have a high capacity (40 gigabyte to 300 gigabytes tapes are currently available), and can be removed from the tape drive, so they are well suited to cheap archival storage. Tape jukeboxes are used to hold exceptionally large collections of data, such as remote-sensing data from satellites, which could include as much as hundreds of terabytes (1 terabyte = 1012 bytes), or even a petabyte (1 petabyte = 1015 bytes) of data

The various storage media can be organized in a hierarchy (in given Figure) according to their speed and their cost. The higher levels are expensive, but are fast. As we move down the hierarchy, the cost per bit decreases, whereas the access time increases.



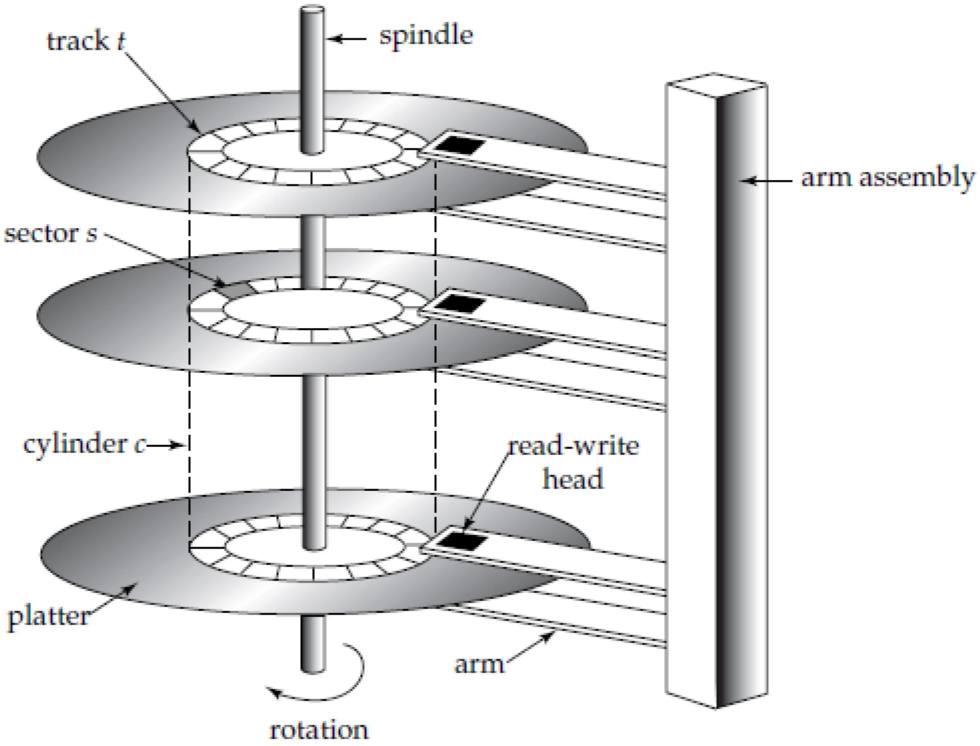
This trade-off is reasonable; if a given storage system were both faster and less expensive than another—other properties being the same—then there would be no reason to use the slower, more expensive memory. In fact, many early storage devices, including paper tape and core memories, are relegated to museums now that magnetic tape and semiconductor memory have become faster and cheaper. Magnetic tapes themselves were used to store active data back when disks were expensive and had low storage capacity. Today, almost all active data are stored on disks, except in rare cases where they are stored on tape or in optical jukeboxes.

The fastest storage media—for example, cache and main memory—are referred to as **primary storage**. The media in the next level in the hierarchy—for example, magnetic disks—are referred to as **secondary storage**, or **online storage**. The media in the lowest level in the hierarchy—for example, magnetic tape and optical-disk jukeboxes—are referred to as **tertiary storage**, or **offline storage**.

In addition to the speed and cost of the various storage systems, there is also the issue of storage volatility. **Volatile storage** loses its contents when the power to the device is removed. In the hierarchy shown in Figure, the storage systems from main memory up are volatile, whereas the storage systems below main memory are nonvolatile. In the absence of expensive battery and generator backup systems, data must be written to **nonvolatile storage** for safekeeping.

Magnetic Disks

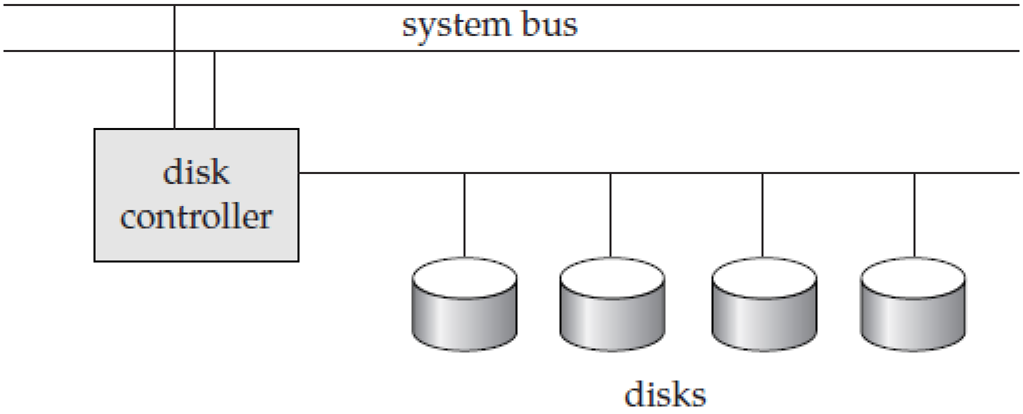
Magnetic disks provide the bulk of secondary storage for modern computer systems. Disk capacities have been growing at over 50 percent per year, but the storage requirements of large applications have also been growing very fast, in some cases even faster than the growth rate of disk capacities. A large database may require hundreds of disks.



- Physically, disks are relatively simple. Each disk platter has a flat circular shape. Its two surfaces are covered with a magnetic material, and information is recorded on the surfaces. Platters are made from rigid metal or glass and are covered (usually on both sides) with magnetic recording material.
- When the disk is in use, a drive motor spins it at a constant high speed (usually 60, 90, or 120 revolutions per second, but disks running at 250 revolutions per second are available). There is a read–write head positioned just above the surface of the platter.
- The disk surface is logically divided into **tracks**, which are subdivided into **sectors**. A sector is the smallest unit of information that can be read from or written to the disk.
- The inner tracks (closer to the spindle) are of smaller length, and in current-generation disks, the outer tracks contain more sectors than the inner tracks.
- The read–write head stores information on a sector magnetically as reversals of the direction of magnetization of the magnetic material. There may be hundreds of concentric tracks on a disk surface, containing thousands of sectors.
- Each side of a platter of a disk has a read–write head, which moves across the platter to access different tracks. A disk typically contains many platters, and the read –write heads of all the tracks are mounted on a single assembly called a disk arm, and move together.
- The disk platters mounted on a spindle and the heads mounted on a disk arm are together known as head–disk assemblies. Since the heads on all the platters move together, when the head on one platter is on the ith track, the heads on all other platters are also on the ith track of their respective platters. Hence, the ith tracks of all the platters together are called the ith **cylinder.**
- A **disk controller** interfaces between the computer system and the actual hardware of the disk drive. It accepts high-level commands to read or write a sector, and initiates actions, such as moving the disk arm to the right track and actually reading or writing the data.

- Disk controllers also attach **checksums** to each sector that is written; the checksum is computed from the data written to the sector. When the sector is read back, the controller computes the checksum again from the retrieved data and compares it with the stored checksum; if the data are corrupted, with a high probability the newly computed checksum will not match the stored checksum.

Given Figure shows how disks are connected to a computer system. Like other storage units, disks are connected to a computer system or to a controller through a high speed interconnection.



The AT attachment (ATA) interface (which is a faster version of the integrated drive electronics (IDE) interface used earlier in IBM PCs) and a small-computer system interconnect (SCSI; pronounced “scuzzy”) are commonly used to connect disks to personal computers and workstations.

IMPORTANT:

While disks are usually connected directly by cables to the disk controller, they can be situated remotely and connected by a high-speed network to the disk controller. In the **storage area network** (**SAN**) architecture, large numbers of disks are connected by a high-speed network to a number of server computers. The disks are usually organized locally using **redundant arrays of independent disks** (**RAID**) storage organizations, but the RAID organization may be hidden from the server computers: the disk subsystems pretend each RAID system is a very large and very reliable disk. The controller and the disk continue to use SCSI or Fiber Channel interfaces to talk with each other, although they may be separated by a network. Remote access to disks across a storage area network means that disks can be shared by multiple computers, which could run different parts of an application in parallel. Remote access also means that disks containing important data can be kept in a central server room where they can be monitored and maintained by system administrators, instead of being scattered in different parts of an organization.

Disk Scheduling:

One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives, meeting this responsibility entails having 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 heads to the cylinder containing the desired sector.

The **Rotation latency** is the additional time for the disk to rotate the desired sector to the disk head.

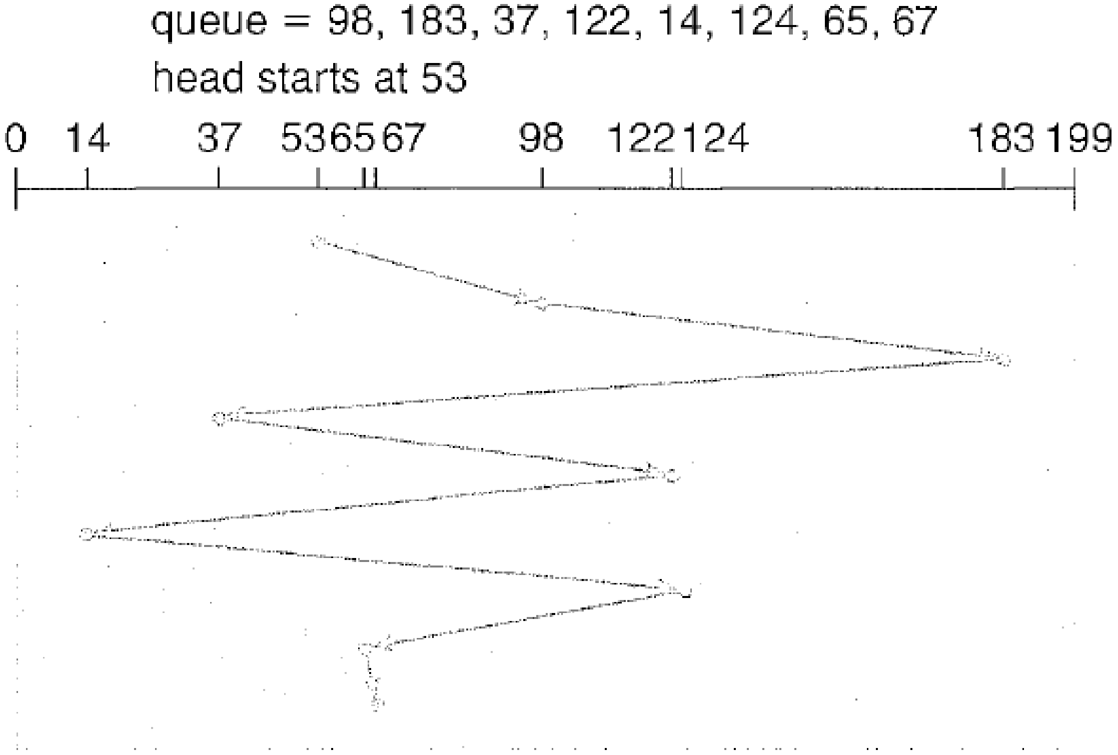
The disk **Bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

We can improve both the access time and the bandwidth by managing the order in which disk I/O requests are serviced.

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/0 to blocks on cylinders in that 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 movement of 640 cylinders. This schedule is diagrammed in Figure

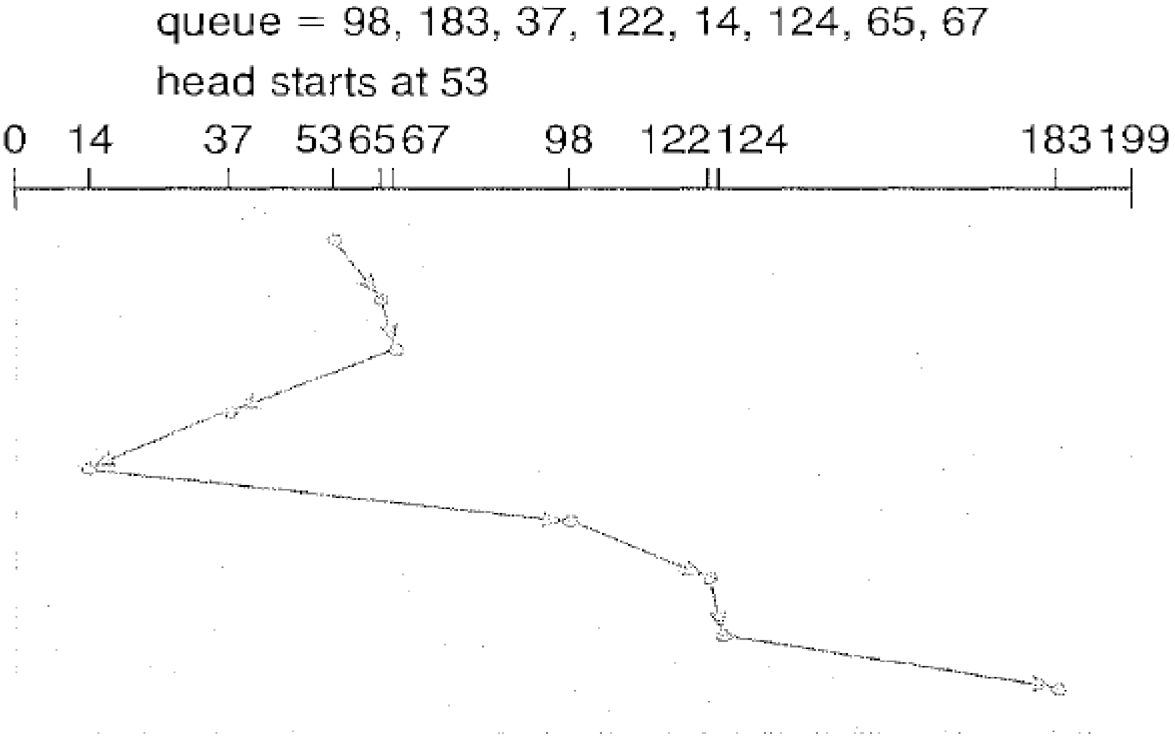
98, 183, 37, 122, 14, 124, 65, 67



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 request. This assumption is the basis for the **Shortest Seek Time First**. The SSTF algorithm 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 current head position.

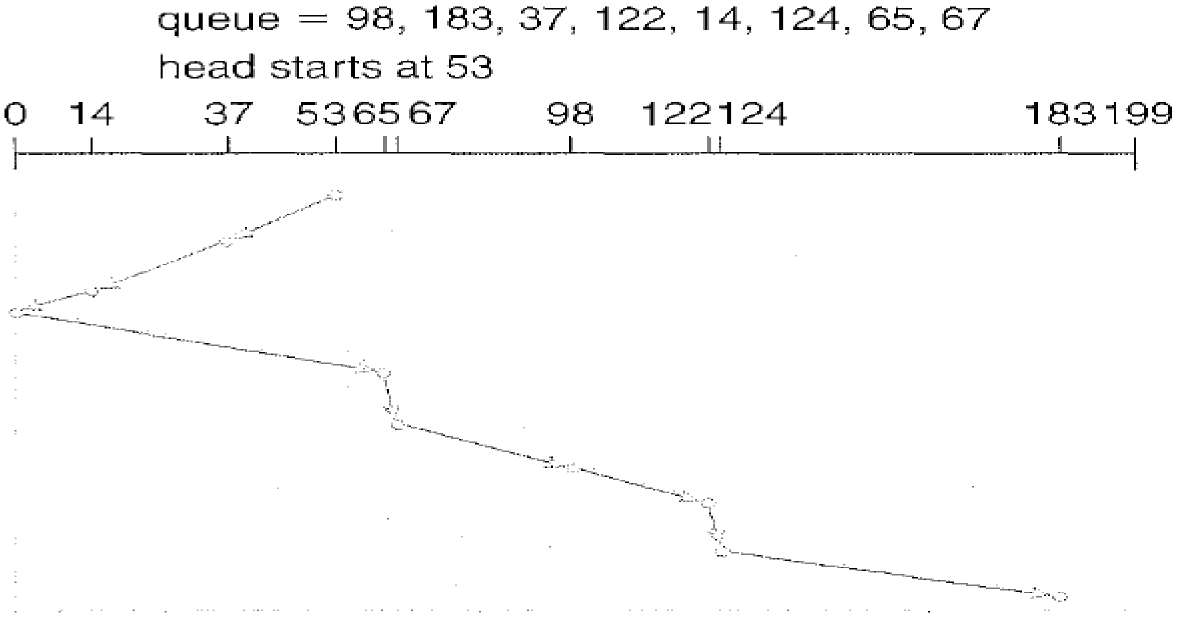


SCAN Scheduling

In this algorithm 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.

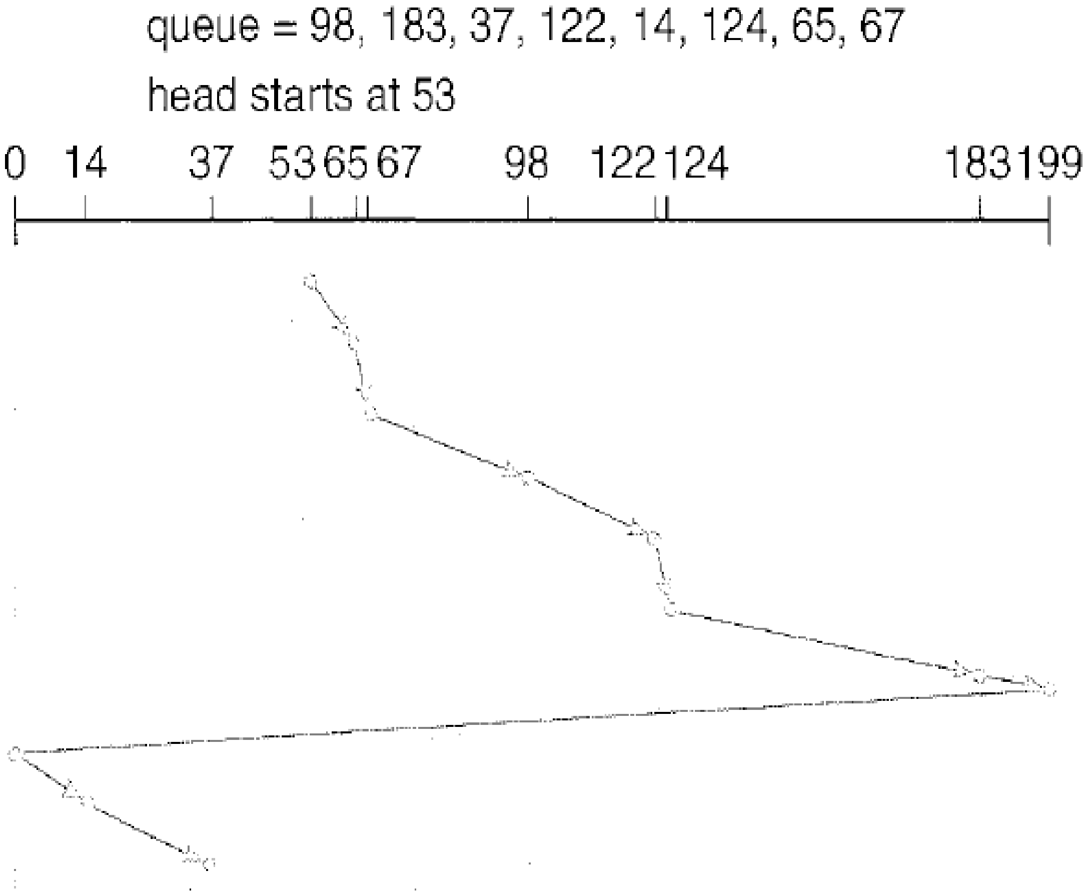
At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk.

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.



C-SCAN Scheduling

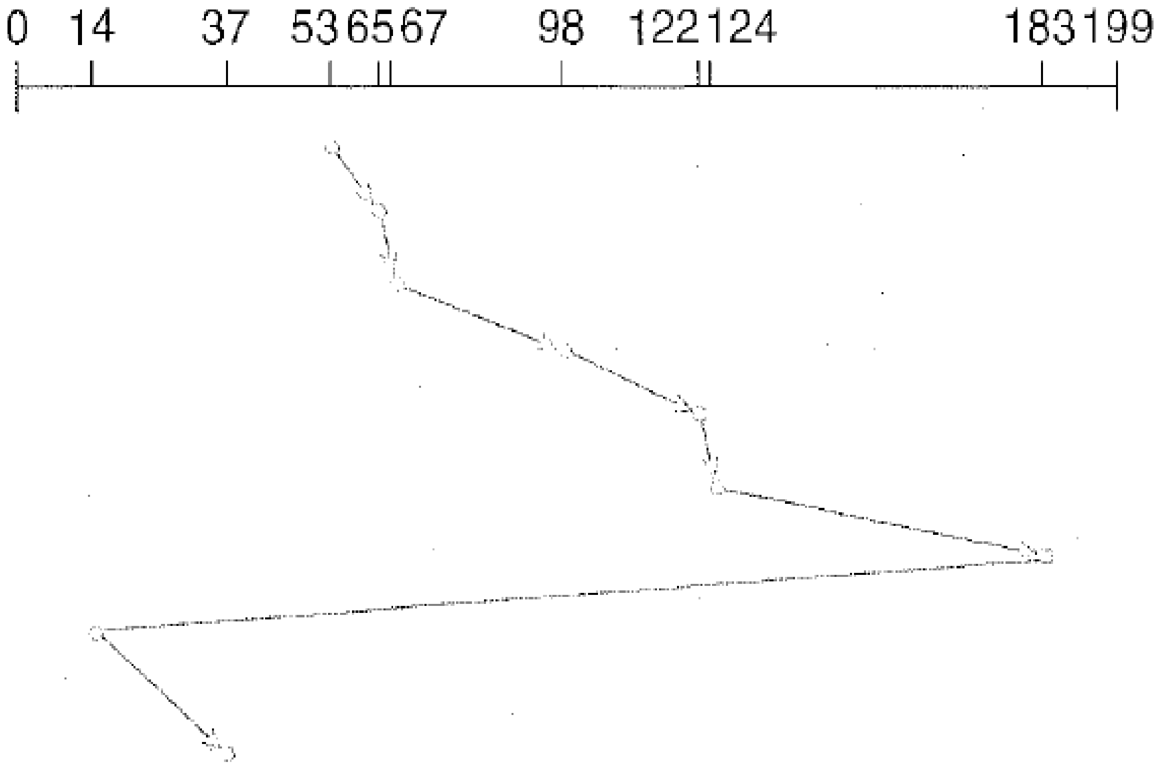
C-SCAN Scheduling is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing 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.



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 or C-LOOK** and because they *look* for a request before continuing to move in a given direction queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53

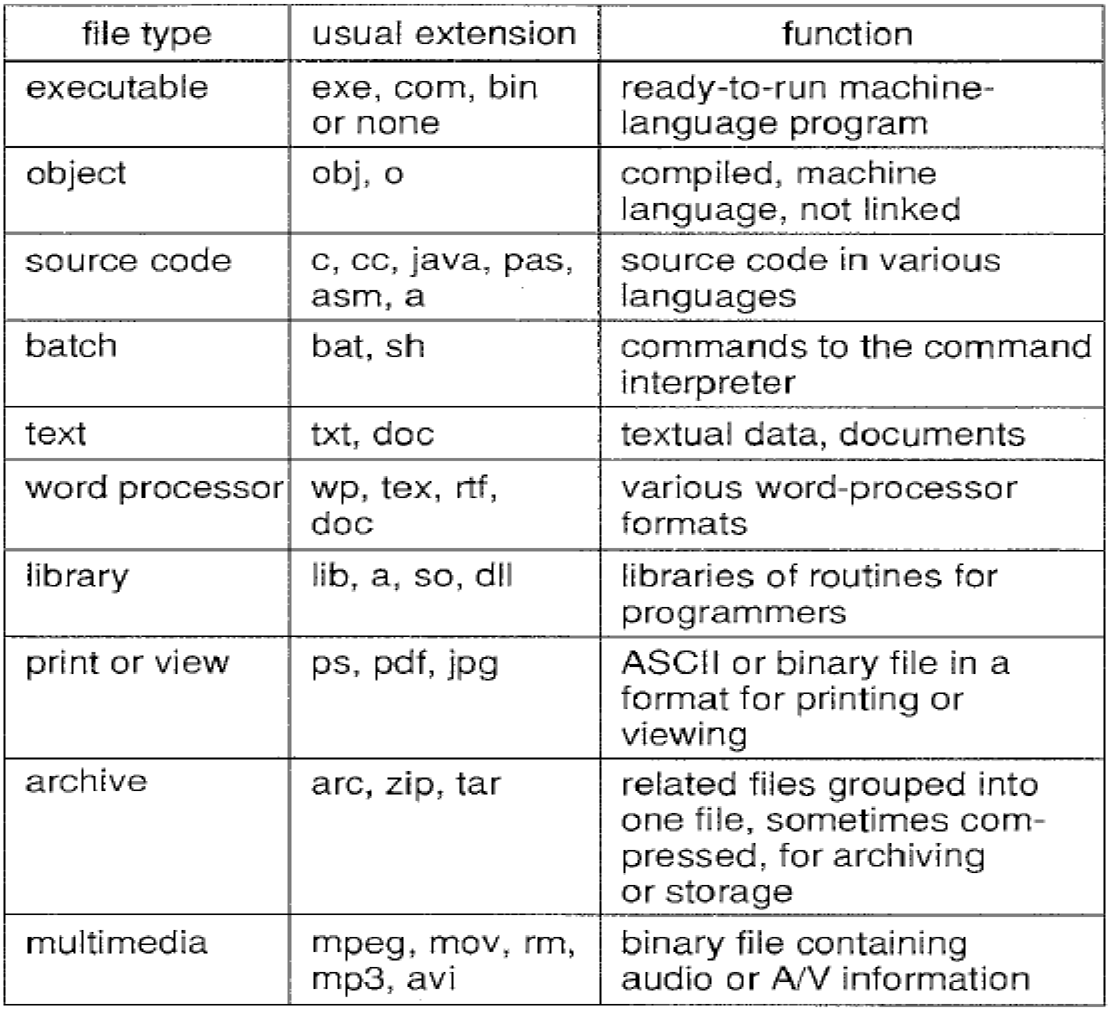


File System

The file system is the most visible aspect of an operating system. It provides the mechanism for on-line storage of and access to both data and programs of the operating system and all the users of the computer system. The file system consists of two distinct parts: a collection of *files*, each storing related data, and a *directory structure*, which organizes and provides information about all the files in the system.

File:

- A file is a named collection of related information that is recorded on secondary storage. From a 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.
- A common technique for implementing file types is to include the type as part of the file name. The name is split into two parts-a name and an extension, usually separated by a period character.



file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

File Attributes

A file is named, for the convenience of its human users, and is referred to by its name. A name is usually a string of characters, such as **example.c.** Some systems differentiate between uppercase and lowercase characters in names, whereas other systems do not.

A file's attributes vary from one operating system to another but typically consist of these:

- **Name**. The symbolic file name is the only information kept in human readable form.
- **Identifier**. This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.

- **Type.** This information is needed for systems that support 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, words, or blocks) and possibly the maximum allowed size are included in this attribute.
- **Protection**. Access-control information determines who can do reading, writing, executing, and so on.
- **Time, date, and user identification**. This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

File Operations

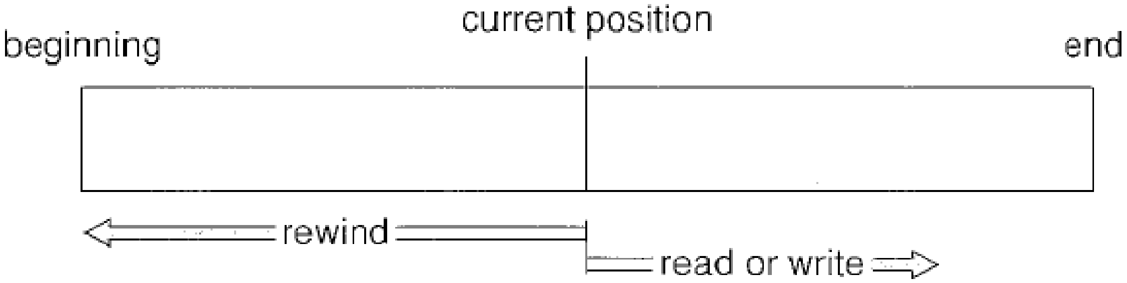
- **Creating a file.** Two steps are necessary to create a file. First, space in the file system must be found for the file. We discuss how to allocate space for the file in Chapter 11. Second, an entry for the new file must be made in the directory.
- **Writing a file.** To write a file, we make a system call specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the file's location. The system must keep 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 where (in memory) the next block of the file should be put. Again, the directory is searched for the associated entry, and the system needs to keep a read pointer to the location in the file where the next read is to take place. Once the read has taken place, the read pointer is updated. Because a process is usually either reading from or writing to a file, the current operation location can be kept as a per-process. Both the read and write operations use this same pointer, saving space and reducing system complexity.
- **Repositioning within a file**. The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value. Repositioning within a file need not involve any actual I/0. This file operation is also known as a file seek.
- **Deleting a file**. To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase the directory entry.
- **Truncating a file**. The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged –except for file length-but lets the file be reset to length zero and its file space released.

Access Methods:

1. **Sequential Access:** The simplest access method is Sequential Access. Information in the file is processed in order, one record after the other. This mode of access is by far the most common; for example, editors and compilers usually access files in this fashion.

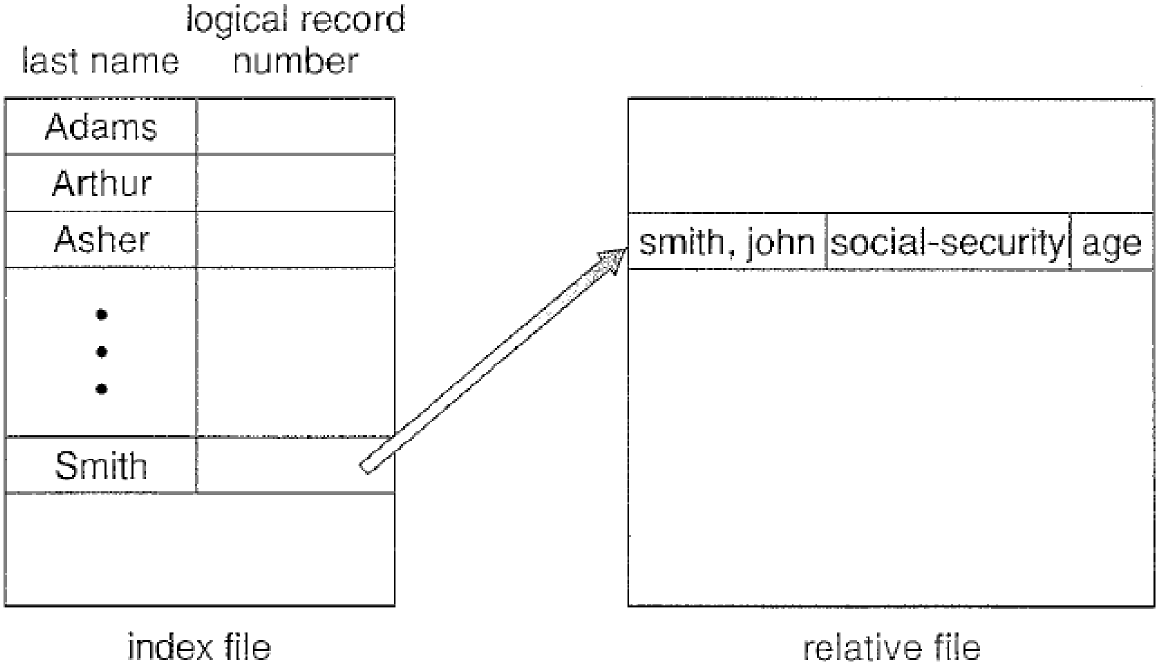
   Reads and writes make up the bulk of the operations on a file. A read operation-**read next**-reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location. Similarly, the write operation-**write next-**appends to the end of the file and advances to the end of the newly written material (the new end of file).

   Sequential access, which is depicted in Figure, is based on a tape model of a file and works as well on sequential-access devices.



2. **Direct Access:** Another method of file access is Direct Access method. A file is made up of fixed length logical address that allow programs to read and write records rapidly in no particular order. The direct-access method is based on a disk model of a file, since disks allow random access to any file block. For direct access, the file is viewed as a numbered sequence of blocks or records. Thus, we may read block 14, then read block 53, and then write block 7. There are no restrictions on the order of reading or writing for a direct-access file. For the direct-access method, the file operations must be modified to include the block number as a parameter. Thus, we have read n, where n is the block number, rather than read next, and ·write n rather than write next.

3. **Indexed Access:** This methods generally involve the construction of an index for the file. The index like an index in the back of a book contains pointers to the various blocks. To find a record in the file, we first search the index and then use the pointer to access the file directly and to find the desired record.



Free Space Management:

Since disk space is limited, we need to reuse the space from deleted files for new files, if possible. To keep track of free disk space, the system maintains a free space list. The free-space list records all *free* disk blocks-those not allocated to some file or directory. To create a file, we search the free-space list for the required amount of space and allocate that space to the new file. This space is then removed from the free-space list. When a file is deleted, its disk space is added to the free-space list.

Bit Vector

Frequently, the free-space list is implemented as a bit map or bit vector. Each block is represented by 1 bit. If the block is free, the bit is 1; if the block is allocated, the bit is 0.

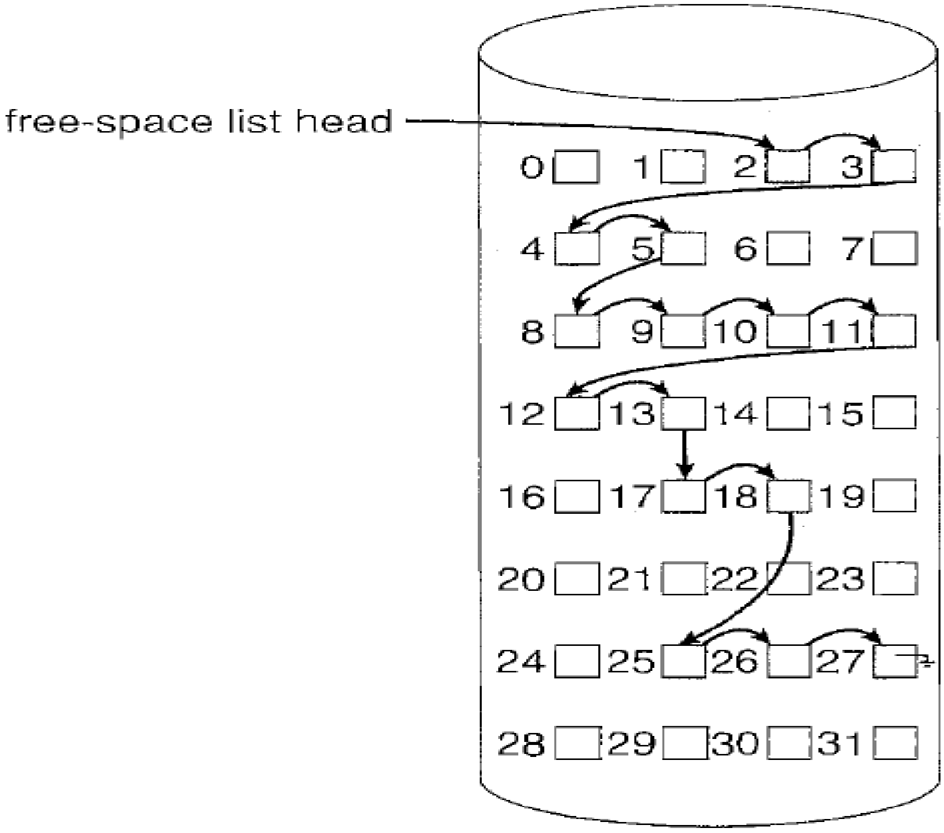
For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17,18, 25, 26, and 27 are free and the rest of the blocks are allocated. The free-space bit map would be

001111001111110001100000011100000 ...

The main advantage of this approach is its relative simplicity and its efficiency in finding the first free block or n consecutive free blocks on the disk.

Linked List

Another approach to free-space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory. This first block contains a pointer to the next free disk block, and so on. Recall our earlier example, in which blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 were free and the rest of the blocks were allocated. In this situation, we would keep a pointer to block 2 as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on



Grouping

A modification of the free-list approach stores the addresses of n free blocks in the first free block. The first n-1 of these blocks are actually free. The last block contains the addresses of another n free blocks, and so on. The addresses of a large number of free blocks can now be found quickly, unlike the situation when the standard linked-list approach is used.

Counting

Another approach takes advantage of the fact that, generally, several contiguous blocks may be allocated or freed simultaneously, particularly when space is allocated with the contiguous-allocation algorithm or through clustering. Thus, rather than keeping a list of n free disk addresses, we can keep the address of the first free block and the number (n) of free contiguous blocks that follow the first block. Each entry in the free-space list then consists of a disk address and a count.

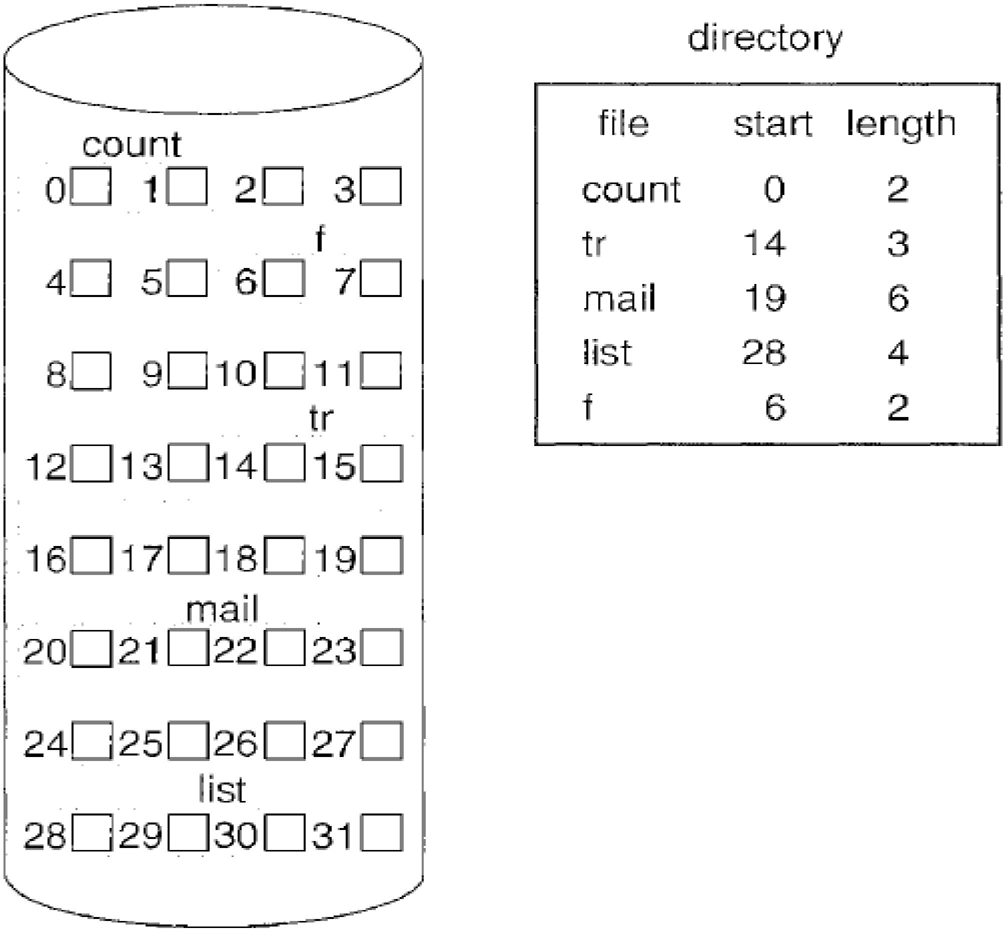
Allocation Method:

The direct-access nature of disks allows us flexibility in the implementation of files. In almost every case, many files are stored on the same disk. The main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly. Three major methods of allocating disk space are in wide use: contiguous, linked, and indexed.

Contiguous Allocation

Contiguous Allocation addresses define a linear ordering on the disk. With this ordering, assuming that only one job is accessil1.g the disk, accessing block b + 1 after block b normally requires no head movement. When head movement is needed the head need only move from one track to the next. Thus, the number of disk seeks required for accessing contiguously allocated files is minimal, as is seek time when a seek is finally needed.

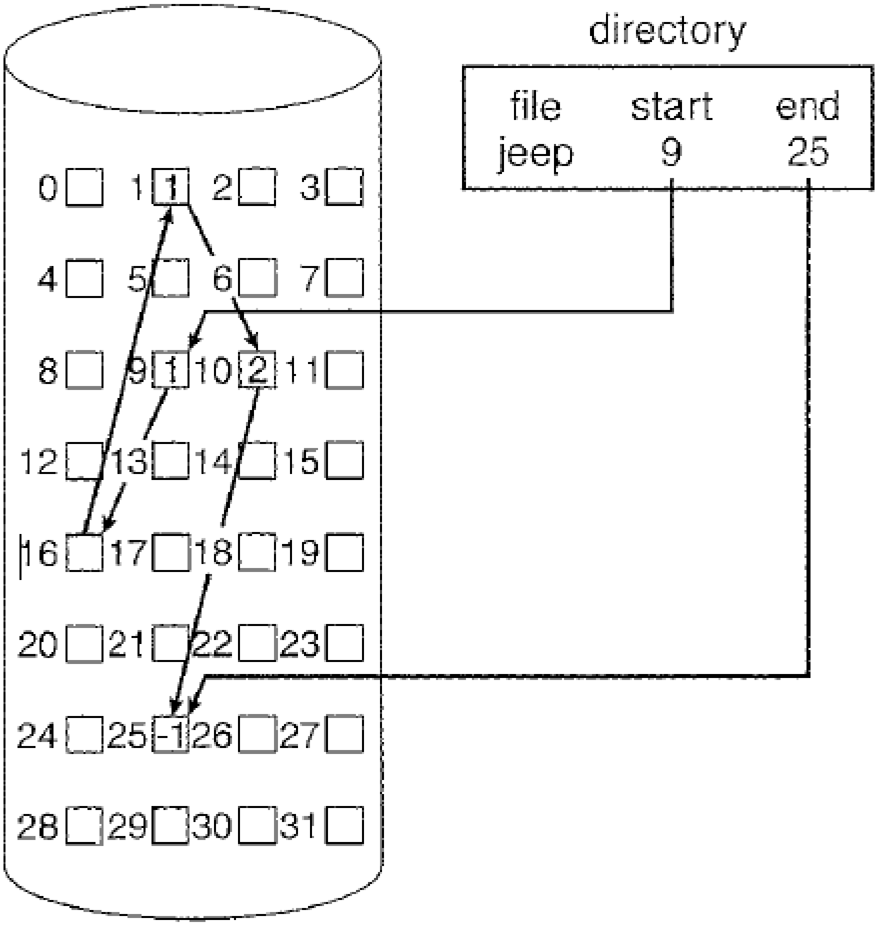
Accessing a file that has been allocated contiguously is easy. For sequential access, the file system remembers the disk address of the last block referenced and, when necessary, reads the next block. For direct access to block i of a file that starts at block b, we can immediately access block b + i. Thus, both sequential and direct access can be supported by contiguous allocation.



file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

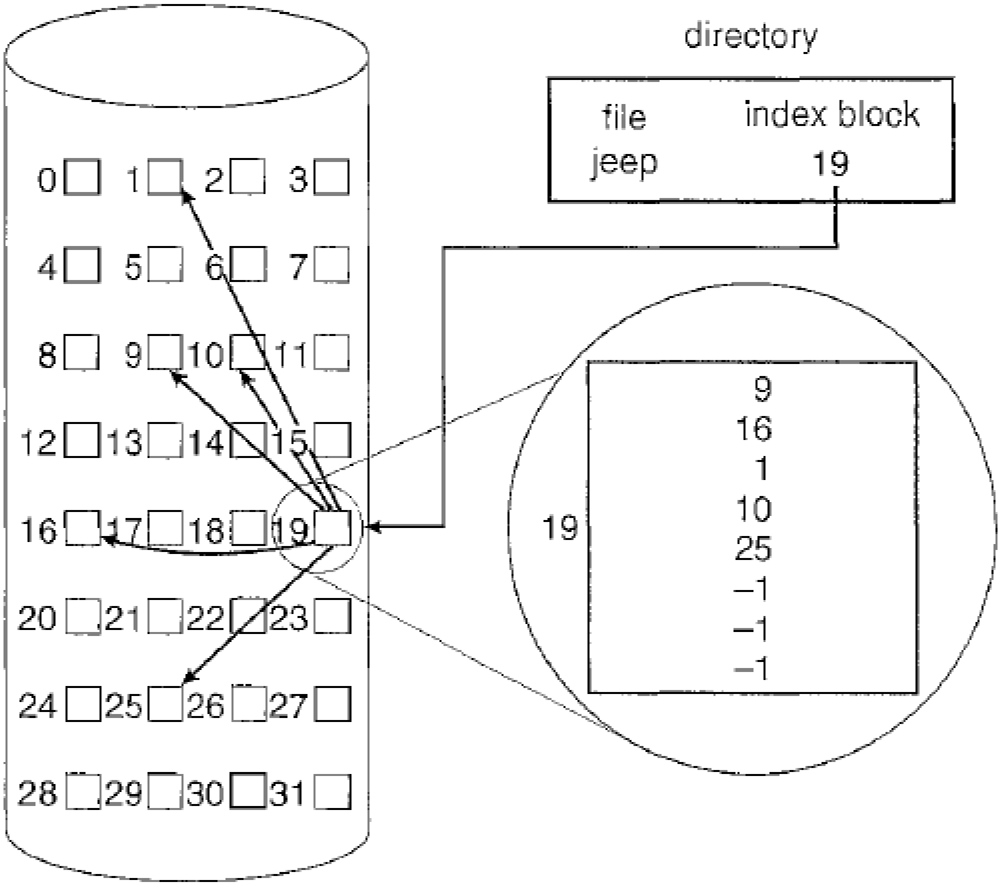
Linked Allocation

Linked Allocation solves all problems of contiguous allocation. With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. For example, a file of five blocks might start at block 9 and continue at block 16, then block 1, then block 10, and finally block 25. Each block contains a pointer to the next block. These pointers are not made available to the user.



Indexed Allocation

Indexed Allocation bring all the pointers together into one location: the Index block.
Each file has its own index block, which is an array of disk-block addresses. The ith entry in the index block points to the ith block of the file. The directory contains the address of the index block. To find and read the ith block, we use the pointer in the ith index-block entry. This scheme is similar to the paging scheme.


directory
file
index block
jeep
19
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
19
9
16
1
10
25
−1
−1
−1