**File System:** The Concept of a File- file attributes, file operations, file types, file structures; Access Methods- sequential access, direct access, other access methods, Directory Structure single level directory, two level directories, tree structured directory, general graph directory; File Sharing- multiple users, remote file systems, Protection- types of access, access control.

**Implementing File System**: File System Structure. File System Implementation- overview, partitions and mounting, virtual file systems, Allocation Methods- contiguous allocation, linked allocation, indexed allocation; Free-Space Management- linked list, grouping, counting.

**Disk Scheduling:** FCFS Scheduling, SSTF Scheduling, SCAN Scheduling, C-SCAN scheduling, LOOK Scheduling

## FILE SYSTEM: FILE CONCEPT

A file is a collection of similar records. The data can't be written on to the secondary storage unless they are within a file. Files represent both the program and the data. Data can be numeric, alphanumeric, alphabetic or binary. Many different types of information can be stored on a file ---Source program, object programs, executable programs, numeric data, payroll recorder, graphic images, sound recordings and so on.

A file has a certain defined structures according to its type:

1 Text file:-Text file is a sequence of characters organized in to lines.

2 Object file:-Object file is a sequence of bytes organized in to blocks understandable by the systems linker.

3 Executable file:-Executable file is a series of code section that the loader can bring in to memory and execute.

4 Source File:-Source file is a sequence of subroutine and function, each of which are further organized as declaration followed by executable statements.

**File Attributes:-**File attributes varies from one OS to other. The common file attributes are:

1 Name:-The symbolic file name is the only information kept in human readable form.

2 Identifier:-The unique tag, usually a number, identifies the file within the file system. It is the non- readable name for a file.

3 Type:-This information is needed for those, whose system that supports different types.

4 Location:-This information is a pointer to a device and to the location of the file on that device.

5 Size:-The current size of the file and possibly the maximum allowed size are included in this attribute.

6 Protection:-Access control information determines who can do reading, writing, execute and so on.

7 Time, data and User Identification:-This information must be kept for creation, lastmodification and last use. These data are useful for protection, security and usage monitoring. **File Operations:-** File is an abstract data type. To define a file we need to consider the operationthat can be performed on the file.

Basic operations of files are:

1. Creating a file:-Two steps are necessary to create a file. First space in the file system for file is found. Second an entry for the new file must be made in the directory. The directory entry records the name of the file and the location in the file system.
2. Writing a file:-System call is mainly used for writing in to the file. System call specify the name of the file and the information i.e., to be written on to the file. Given the name the system search the entire directory for the file. The system must keep a write pointer to the location in thefile where the next write to be taken place.
3. Reading a file:-To read a file system call is used. It requires the name of the file and the memory

address. Again the directory is searched for the associated directory and system must maintain a read pointer to the location in the file where next read is to take place.

4. Delete a file:-System will search for the directory for which file to be deleted. If entry is found it releases all free space. That free space can be reused by another file.

5. Truncating a file:-User may want to erase the contents of the file but keep its attributes. Rather than forcing the user to delete a file and then recreate it, truncation allows all attributes to remain unchanged except for file length

6. Repositioning within a file:-The directory is searched for appropriate entry and the current file position is set to a given value. Repositioning within a file does not need to involve actual i/o. The file operation is also known as file seeks.

In addition to this basis 6 operations the other two operations include appending new information to the end of the file and renaming the existing file. These primitives can be combined to perform other two operations. Most of the file operation involves searching the entire directory for the entry associated with the file. To avoid this OS keeps a small table containing information about an open file (the open table). When a file operation is requested, the file is specified via index in to this table. So searching is not required. Several piece of information are associated with an open file:

- File pointer:-on systems that does not include offset an a part of the read and write system calls, the system must track the last read-write location as current file position pointer. This pointer is unique to each process operating on a file.
- File open count:-As the files are closed, the OS must reuse its open file table entries, or it could run out of space in the table. Because multiple processes may open a file, the system must wait for the last file to close before removing the open file table entry. The counter tracks the number of copies of open and closes and reaches zero to last close.
- Disk location of the file:-The information needed to locate the file on the disk is kept in memory to avoid having to read it from the disk for each operation.
- Access rights:-Each process opens a file in an access mode. This information is stored on per-process table the

In addition to this basis 6 operations the other two operations include appending new information to the end of the file and renaming the existing file. These primitives can be combined to perform other two operations. Most of the file operation involves searching the entire directory for the entry associated with the file.

To avoid this OS keeps a small table containing information about an open file (the open table). When a file operation is requested, the file is specified via index in to this table. So searching is not required.

**File Types**

When we design a file system—indeed, an entire operating system—we always consider whether the operating system should recognize and support file types. 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 as shown in Figure 1. In this way, the user and the operating system can tell from the name alone what the type of a file is.

| 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, perl, asm | source code in various languages |
| batch | bat, sh | commands to the command interpreter |
| markup | xml, html, tex | textual data, documents |
| word processor | xml, rtf, docx | various word-processor formats |
| library | lib, a, so, dll | libraries of routines for programmers |
| print or view | gif, pdf, jpg | ASCII or binary file in a format for printing or viewing |
| archive | rar, zip, tar | related files grouped into one file, sometimes compressed, for archiving or storage |
| multimedia | mpeg, mov, mp3, mp4, avi | binary file containing audio or A/V information |

Figure 1: Common File types

**File Structure**

File types also can be used to indicate the internal structure of the file. Source and object files have structures that match the expectations of the programs that read them. Further, certain files must conform to a required structure that is understood by the operating system. For example, the operating system requires that an executable file have a specific structure so that it can determine where in memory to load the file and what the location of the first instruction is.

This point brings us to one of the disadvantages of having the operating system support multiple file structures: it makes the operating system large and cumbersome. If the operating system defines five different file structures, it needs to contain the code to support these file structures. In addition, it may be necessary to define every file as one of the file types supported by the operating system.

## 2. ACCESS METHODS

The information in the file can be accessed in several ways. Different file access methods are:

**1. Sequential Access:** Sequential access is the simplest access method. Information in the file is processed in order, one record after another. Editors and compilers access the files in this fashion. Normally read and write operations are done on the files. A read operation reads the next portion of the file and automatically advances a file pointer, which track next i/o location. Write operation appends to the end of the file and such a file can be next to the beginning. Sequential access depends on a tape model of a file as shown in Figure 2.

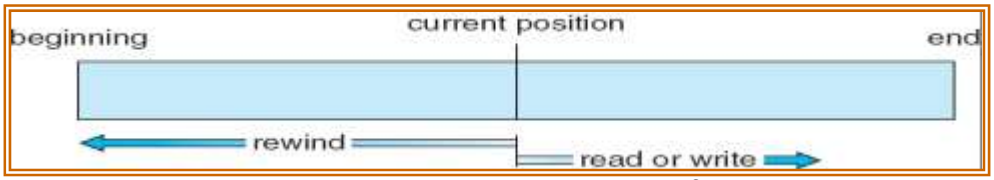


Figure 2: Sequential-access file

**2. Direct Access:** Direct access allows random access to any file block. This method is based on disk model of a file. A file is made up of fixed length logical records. It allows the program to read and write records rapidly in any order. A direct access file allows arbitrary blocks to be read or written.

Eg:-User may need block 13, then read block 99 then write block 12. For searching the records in large amount of information with immediate result, the direct access method is suitable. Not all OS support sequential and direct access. Few OS use sequential access and some OS uses direct access. It is easy to simulate sequential access on a direct access but the reverse is extremely inefficient. Simulation of sequential access on a direct-access file is shown in figure 3.

| sequential access | implementation for direct access |
|---|---|
| reset | cp = 0; |
| read_next | read cp ;<br>cp = cp + 1; |
| write_next | write cp;<br>cp = cp + 1; |

Figure 3: Simulation of sequential access on a direct-access file.

**3. Indexing Method:** The index is like an index at the end of a book which contains pointers to various blocks as shown in Figure 4. To find a record in a file, we search the index and then use the pointerto access the file directly and to find the desired record. With large files index file itself can be very large to be kept in memory. One solution to create an index to the index files itself. The primary index file would contain pointer to secondary index files which would point to the actualdata items.

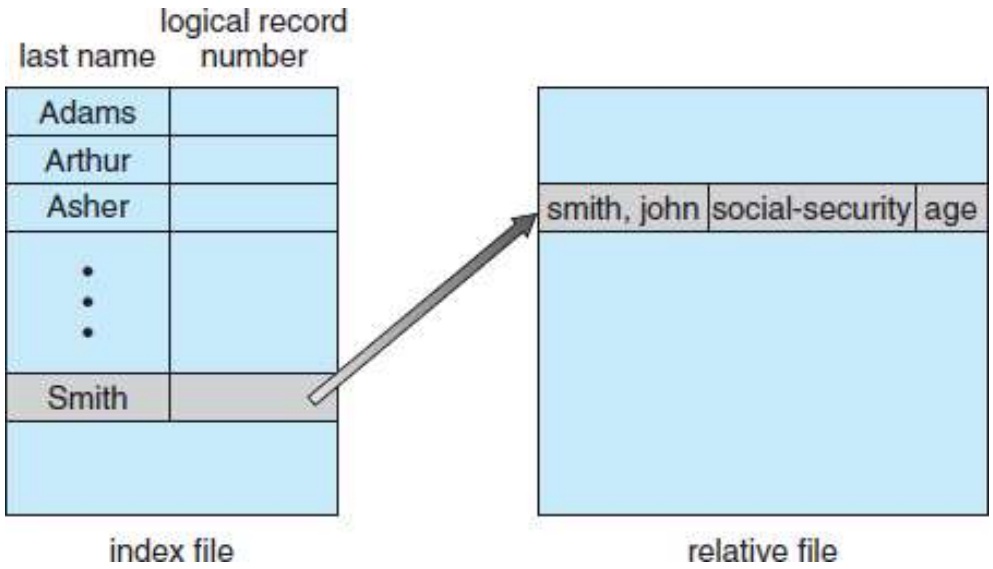


Figure 4: Example of index and relative files.

Two types of indexes can be used:

a. Exhaustive index:-Contain one entry for each of record in the main file. An index itself is organized as a sequential file.

b. Partial index:-Contains entries to records where the field of interest exists with records of variable length, some record will not contain any fields. When a new record is added to the main file, all index files must be updated.

## 3. DIRECTORY STRUCTURE

The files systems can be very large. Some systems stores millions of files on the disk. To manage all this data we need to organize them.

This organization is done in two parts:

1. Disks are split in to one or more partition also known as minidisks.

2. Each partition contains information about files within it. This information is kept in entries in a device directory or volume table of contents. The device directory or simple directory records information as name, location, size, type for all files on the partition. The directory can be viewed as a symbol table that translates the file names in to the directory entries. The directory itself can be organized in many ways.

When considering a particular directory structure, we need to keep in mind the operations that are to be performed on a directory.

- Search for a file:-Directory structure is searched for finding particular file in the directory. Files have symbolic name and similar name may indicate a relationship between files, we may want to be able to find all the files whose name match a particular pattern.
- Create a file:-New files can be created and added to the directory.
- Delete a file:-when a file is no longer needed, we can remove it from the directory.
- List a directory:-We need to be able to list the files in directory and the contents of the directory entry for each file in the list.
- Rename a file:-Name of the file must be changeable when the contents or use of the file is changed. Renaming allows the position within the directory structure to be changed.
- Traverse the file:-it is always good to keep the backup copy of the file so that or it can be used when the system gets fail or when the file system is not in use.

**I. Single-level directory:** This is the simplest directory structure. All the files are contained in the same directory which is easy to support and understand as shown in Figure 5.

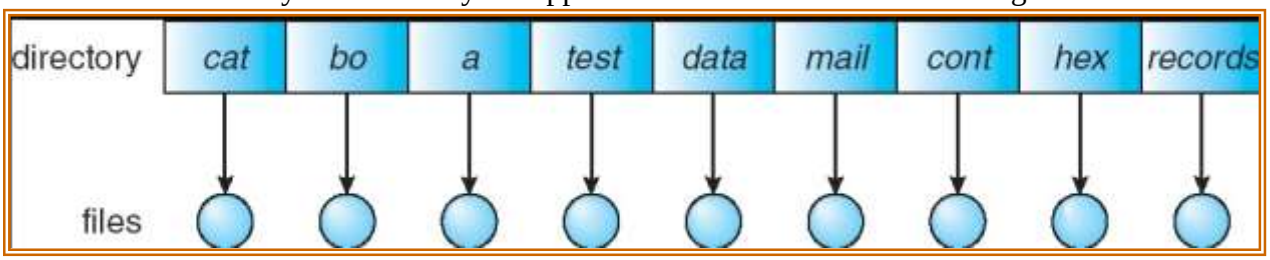


Figure 5: Single-level directory

***Disadvantage:***

- Not suitable for a large number of files and more than one user.
- Because of single directory files, files require unique file names.
- Difficult to remember names of all the files as the number of files increases. MS-DOS OS allows only 11 character file name where as UNIX allows 255 character.

**II. Two-level directory:** A single level directory often leads to the confusion of file names between different users. The solution here is to create separate directory or each user as shown in Figure 6.

In two level directories each user has its own directory. It is called User File Directory (UFD). Each UFD has a similar structure, but lists only the files of a single user. x When a user job starts or users logs in, the systems Master File Directory (MFD) is searched. The MFD is indexed by the user name or account number and each entry points to the UFD for that user. x When a user refers to a particular file, only his own UFD is searched. Thus different users may have files with the same name.

To create a file for a user, OS searches only those users UFD to ascertain whether anotherfile of that name exists.

To delete a file checks in the local UFD so that accidentally delete another user's file with the same name

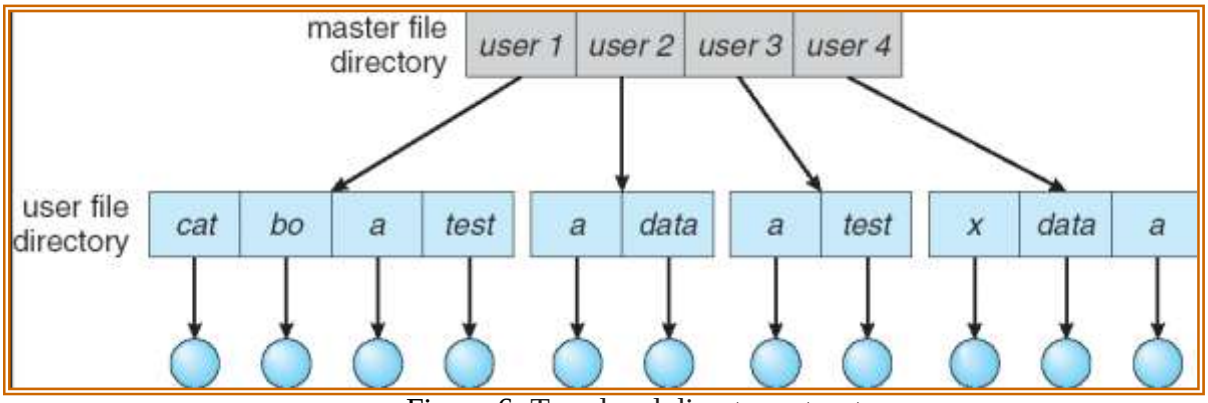


Figure 6: Two-level directory structure.

Although two-level directories solve the name collision problem but it still has some disadvantage. This structure isolates one user from another. This isolation is an advantage. When the users are independent but disadvantage, when some users want to co-operate on some table and to access one another file.

**III. Tree-structured directories:** MS-DOS use Tree structure directory as shown in Figure 7. It allows users to createtheir own subdirectory and to organize their files accordingly. A subdirectory contains a set of files or subdirectories. A directory is simply another file, but it is treated in a special way. The entire directory will have the same internal format. One bit in each entry defines the entry as a file (0) and as a subdirectory (1). Special system calls are used to create and delete directories. Innormal use each uses has a current directory. Current directory should contain most of the files that are of the current interest of users. When a reference to a file is needed the current directory is searched. If file is needed i.e., not in the current directory to be the directory currently holding that file.

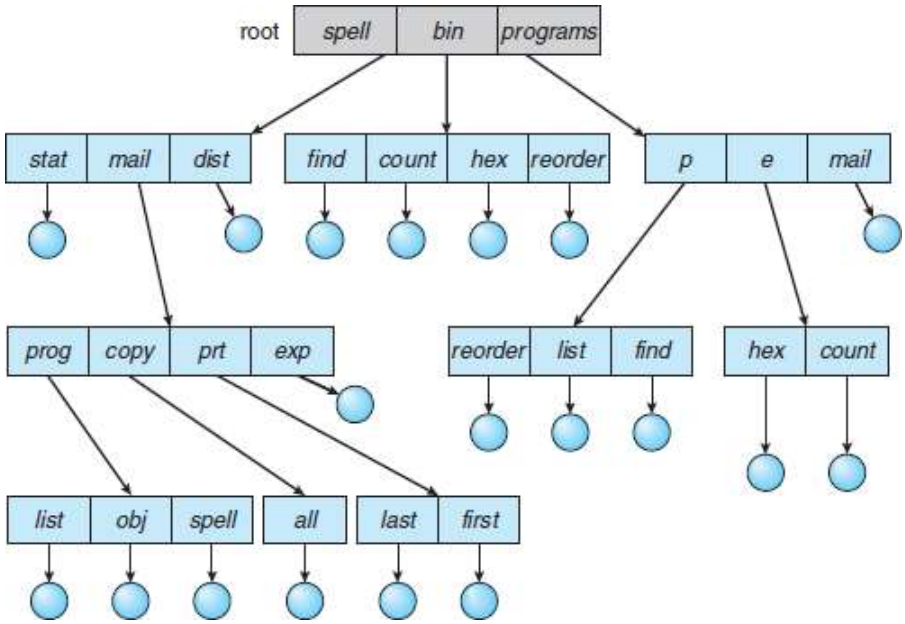


Figure 7: Tree-structured directory structure

Path name can be of two types: a. Absolute path name:-Begins at the root and follows a path down to the specified file, giving the directory names on the path. b. Relative path name:-Defines a path from the current directory. One important policy in this structure is low to handle the deletion of a directory.

a. If a directory is empty, its entry can simply be deleted.

b. If a directory is not empty, one of the two approaches can be used.

i. In MS-DOS, the directory is not deleted until it becomes empty.

ii. In UNIX, RM command is used with some options for deleting directory.

**IV. Acyclic graph directories:** It allows directories to have shared subdirectories and files as shown in Figure 8. Same file or directory may be in two different directories. A graph with no cycles is a generalization of the tree structure subdirectories scheme. Shared files and subdirectories can be implemented by using links. A link is a pointer to another file or a subdirectory. A link is implemented as absolute or relative path. An acyclic graph directory structure is more flexible then is a simple tree structure but sometimes it is more complex.

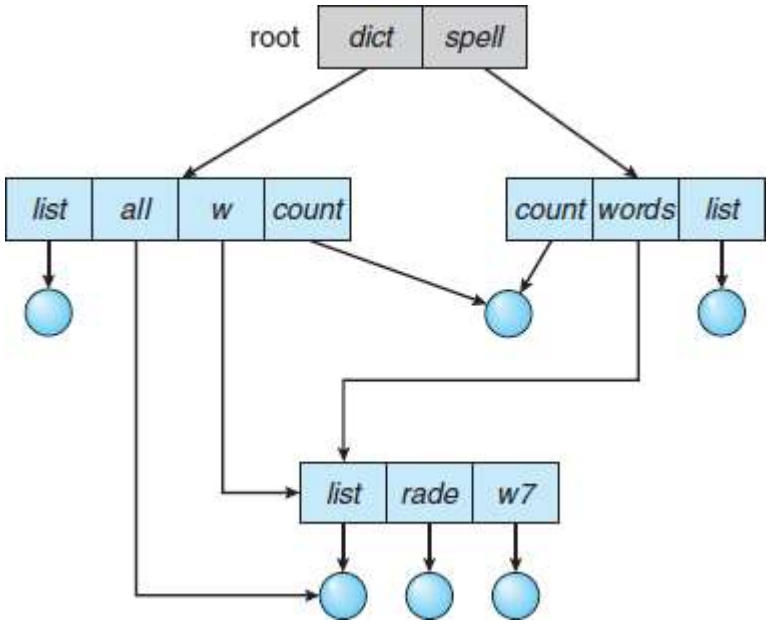


Figure 8: Acyclic-graph directory structure.

## 5. FILE SHARING and PROTECTION

### FILE SHARING

The ability to share files is very desirable for users who want to collaborate and to reduce the effort required to achieve a computing goal.

### Multiple Users

When an operating system accommodates multiple users, the issues of file sharing, file naming, and file protection become preeminent. Given a directory structure that allows files to be shared by users, the system must mediate the file sharing. The system can either allow a user to access the files of other users by default or require that a user specifically grant access to the files. These are the issues of access control and protection.

### Remote File Systems

### The Client –Server Model

Remote file systems allow a computer to mount one or more file systems from one or more remote machines. In this case, the machine containing the files is the server, and the machine seeking access to the files is the client. The client–server relationship is common with networked machines.

Generally, the server declares that a resource is available to clients and specifies exactly which resource (in this case, which files) and exactly which clients. A server can serve multiple clients, and a client can use multiple servers, depending on the implementation details of a given client–server facility.

**Distributed Information Systems** To make client–server systems easier to manage, distributed information systems, also known as distributed naming services, provide unified access to the information needed for remote computing. The domain name system (DNS) provides host-name-to-network-address translations for the entire Internet. Before DNS became widespread, files containing the same information were sent via e-mail or ftp between all networked hosts. Obviously, this methodology was not scalable!

**Failure Modes** Local file systems can fail for a variety of reasons, including failure of the drive containing the file system, corruption of the directory structure or other disk-management information (collectively called metadata), disk-controller failure, cable failure, and host-adapter failure. Remote file systems have even more failure modes. Because of the complexity of network systems and the required interactions between remote machines, many more problems can interfere with the proper operation of remote file systems. In the case of networks, the network can be interrupted between two hosts. Such interruptions can result from hardware failure, poor hardware configuration, or networking implementation issues.

**Protection**

Protection mechanisms provide controlled access by limiting the types of file access that can be made. Access is permitted or denied depending on several factors, one of which is the type of access requested. Several different types of operations 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 for possible reuse.
- List. List the name and attributes of the file.
- Attribute change. Changing the attributes of the file.

Other operations, such as renaming, copying, and editing the file, may also be controlled.

**IMPLEMENTING FILE SYSTEM**

## ALLOCATION METHODS

The space allocation strategy is closely related to the efficiency of the file accessing andof logical to physical mapping of disk addresses.

A good space allocation strategy must take in to consideration several factors such as:

1. Processing speed of sequential access to files, random access to files and allocation and de-allocation of blocks.
2. Disk space utilization.
3. Ability to make multi-sector and multi-track transfers.
4. Main memory requirement of a given algorithm. Three major methods of allocating disk space is used.

1. **Contiguous Allocation:** A single set of blocks is allocated to a file at the time of file creation. This is a pre-allocation strategy that uses portion of variable size. The file allocation table needs just a single entry for each file, showing the starting block and the length of the blocks as shown in Figure 9. If the file is n blocks long and starts at location b, then it occupies blocks b, b+1, b+2….b+n

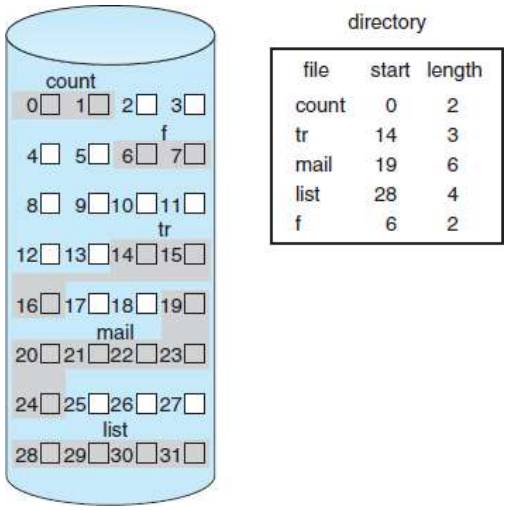


Figure 9: Contiguous allocation of disk space

The file allocation table entry for each file indicates the address of starting block and the length of the area allocated for this file. Contiguous allocation is the best from the point of view of individual sequential file. It is easy to retrieve a single block. Multiple blocks can be broughtin one at a time to improve I/O performance for sequential processing. Sequential and direct access can be supported by contiguous allocation. Contiguous allocation algorithm suffers from external fragmentation. Depending on the amount of disk storage the external fragmentation can be a major or minor problem. Compaction is used to solve the problem of external fragmentation. The following figure shows the contiguous allocation of space after compaction. The original disk was then freed completely creating one large contiguous space. If the file is n blocks long and starts at location b, then it occupies blocks b, b+1, b+2…..b+n

Advantages:

- Supports variable size problem.
- Easy to retrieve single block.
- Accessing a file is easy.
- It provides good performance.

Disadvantage:

- Pre-allocation is required.
- It suffers from external fragmentation.

2. **Linked Allocation:** It solves the problem of contiguous allocation. This allocation is on the basis of an individual block. Each block contains a pointer to the next block in the chain as shown in Figure 10. The disk block can be scattered anywhere on the disk. The directory contains a pointer to the first andthe last blocks of the file.

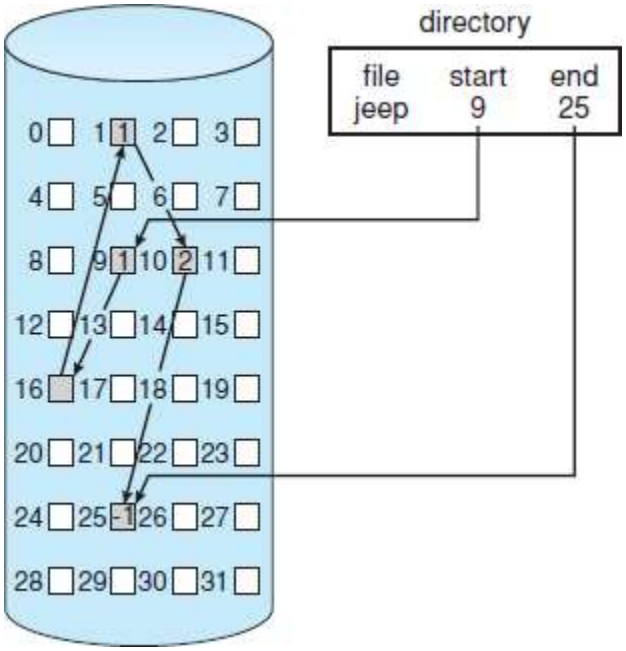


Figure 10: Linked allocation of disk space.

The following figure shows the linked allocation. To create a new file, simply create a new entry in the directory. There is no external fragmentation since only one block is needed at atime. The size of a file need not be declared when it is created. A file can continue to grow as long as free blocks are available.

Advantages:

- No external fragmentation.
- Compaction is never required.
- Pre-allocation is not required.

Disadvantages:

- Files are accessed sequentially.
- Space required for pointers.
- Reliability is not good.
- Cannot support direct access.

3. **Indexed Allocation:** The file allocation table contains a separate one level index for each file. The index has one entry for each portion allocated to the file. The $i^{th}$ entry in the index block points to the $i^{th}$ block of the file as shown in Fig.11.

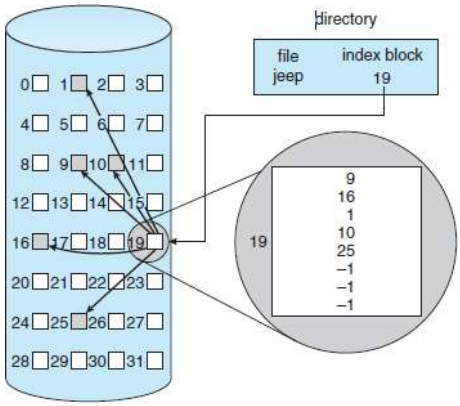


Figure 11: Indexed allocation of disk space.

The indexes are not stored as a part of file allocation table rather than the index is kept as a separate block and the entry in the file allocation table points to that block. Allocation can be made on either fixed size blocks or variable size blocks. When the file is created all pointers in the index block are set to nil. When an entry is made a block is obtained from free space manager. Allocation by fixed size blocks eliminates external fragmentation whereas allocation by variable size blocks improves locality. Indexed allocation supports both direct access and sequential access to the file.

Advantages:-

- Supports both sequential and direct access.
- No external fragmentation.
- Faster than other two methods.
- Supports fixed size and variable sized blocks.

Disadvantage:-

- Suffers from wasted space.
- Pointer overhead is generally greater

**Free-Space Management**

Since storage space is limited, we need to reuse the space from deleted files for new files, if possible. (Write-once optical disks allow only one write to any given sector, and thus reuse is not physically possible.) To keep track of free disk space, the system maintains a free-space list. The free-space list records all free device blocks— those not allocated to some file or directory

**1 Bit Vector**

Frequently, the free-space list is implemented as a bitmap 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 bitmap would be 00111100111111000110000001110000 0 .

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.

**2 Linked List**

Another approach to free-space management is to link together all the free blocks, keeping a pointer to the first free block in a special location in the file system and caching it in memory. This first block contains a pointer to the next free block, and so on. Recall our earlier example in bit vector, 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 as shown in following figure 12.

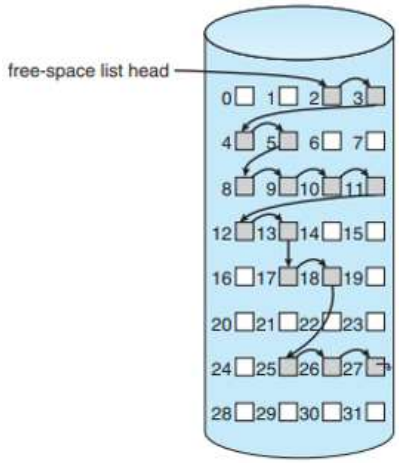


Figure 12: Linked free space list on disk

This scheme is not efficient; to traverse the list, we must read each block, which requires substantial I/O time on HDDs. Fortunately, however, traversing the free list is not a frequent action. Usually the operating system simply needs a free block so that it can allocate that block to a file, so the first block in the free list is used. The FAT method incorporates free-block accounting into the allocation data structure. No separate method is needed.

**3 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.

**4 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 block 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 device address and a count. These entries can be stored in a balanced tree, rather than a linked list, for efficient lookup, insertion, and deletion.

## DISK SCHEDULING ALGORITHMS

- The operating system tries to use hardware efficiently for disk drives ⇒ having fast access time, disk bandwidth
- Access time has two major components
  - *Seek time* is time to move the heads to the cylinder containing the desired sector
  - *Rotational latency* is additional time waiting to rotate the desired sector to the disk head.
- Want to minimize seek time
- Seek time ≈ seek distance
- Disk bandwidth is total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

### i. FCFS Scheduling:

The simplest form of disk scheduling is, the first-come first-served (FCFS) algorithm. This algorithm intrinsically fair, but it generally does not provide the fastest service. Consider, an example a disk queue with requests for I/O to block on cylinders

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

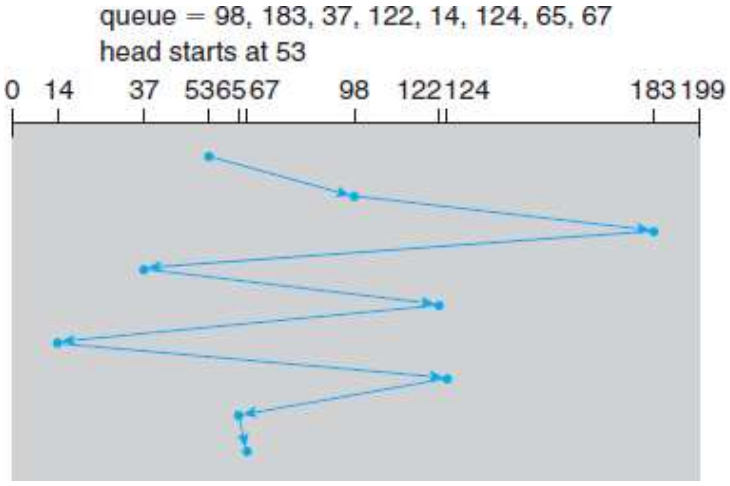


Fig: FCFS disk scheduling

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. The wild swing from 122 to 14 and then backto 124 illustrates the problem with this schedule. If the requests for cylinders 37 and 14 could be services

together, before or after the requests at 122 and 124, the total head movement could be decrease substantially, and performed could be thereby improved.

**ii. SSTF Scheduling:**

It seems reasonable to service all the requests close to the current head position before moving the head far away to service other requests. This assumption is the basis for the **Shortest-Seek-Time-First algorithm.** The SSTF algorithm selects the request with the minimum seek time from the current head position.

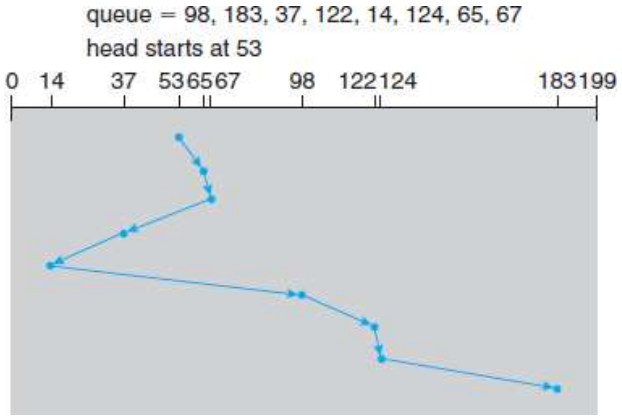


Fig: SSTF disk scheduling

Consider the disk queue with requests 98, 183, 37, 122, 14, 124, 65, 67. For this 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 the service for the request at cylinder 14, after that 98, 122, 124 and finally 183.

SSTF scheduling is essentially a form of shortest-job-first (SJF) scheduling and like SJF scheduling; it may cause starvation of some requests. Suppose that we have two requests in the queue, for cylinders 14 and 186, and while servicing the request from 14 a new request near 14 arrives. This new request will be serviced next, making the request at 186 wait. If any other request near 14 arrives it makes 186 in wait. This becomes increasingly likely if the pending- request queue grows long.

**iii. SCAN Scheduling:**

In the SCAN algorithm, the disk arm starts at one end of the disk and moves towards 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.

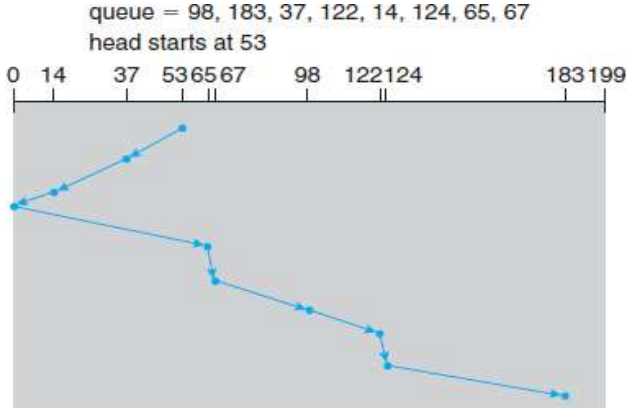

Fig: SCAN disk scheduling

Let's take a example 98, 183, 37, 122, 14, 124, 65, 67. Before applying scan to schedule the requests on cylinders we need to know the direction of head movement in addition to the head's current position (53).

If the disk arm is moving towards 0, the head will 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. 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 end of the disk, reverses direction, and comes back.

**iv. 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 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.

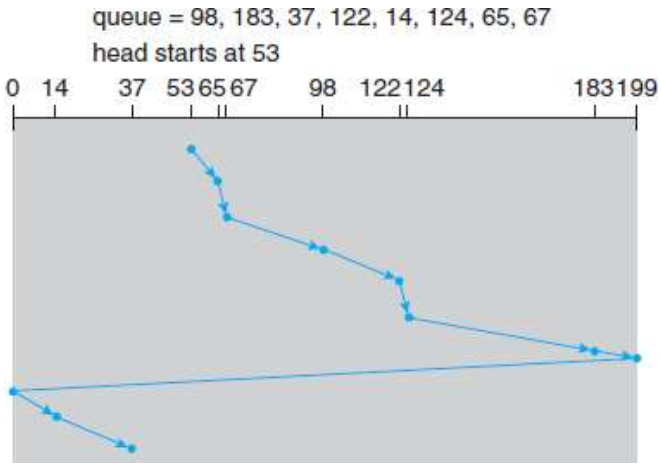


Fig: C-SCAN disk scheduling

**v. LOOK Scheduling:**

As we described that both SCAN and C-SCAN move the disk arm across the full widthof 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 called LOOK and C-LOOK scheduling, because they look for a request before continuing to move in a given direction.

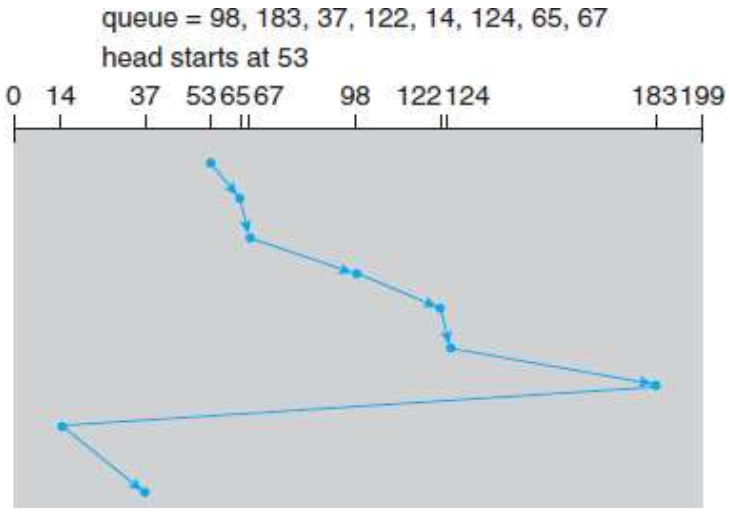


Fig: C-LOOK disk scheduling.

**Selection of a Disk-Scheduling Algorithm:**

Given so many disk-scheduling algorithms, how do we choose the best one? 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. For any particular list of requests, we can define an optimal order of retrieval, but the computation needed to find an optimal schedule may not justify the savings over SSTF or SCAN. With any scheduling algorithm, however, performance depends heavily on the number and types of requests. For instance, suppose that the queue is usually has just one outstanding request. Then, all scheduling algorithms behave the same, because they only one choice for where to move the disk head: They all behave like FCFS scheduling.

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.