Chapter – 9
File System

*** File Concept –**

Computers can store information on various storage media, such as magnetic disks, magnetic tapes, and optical disks. These storage devices are usually nonvolatile.

A file is a named collection of related information that is recorded on secondary storage. Many different types of information may be stored in a file – source programs, object programs, numeric data, text, graphic images, sound recordings, and so on.

a) File Attributes –

A file is named, for the useful of its human users, and is referred to by its name. A name is usually a string of characters, such as sample.txt. 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:

1) **Name –** The symbolic file name of the file.
2) **Identifier –** It is a unique number that identifies the file within the file system.
3) **Type –** This information is needed for systems that support different types of files.
4) **Location –** Location of the file on the device.
5) **Size –** The current size of the file.
6) **Protection –** Access control information determines who can do reading, writing or both.
7) **Time and date –** This information may be kept for creation and last modification.

b) File Operations –

The operating system can provide system calls to create, write, read, and delete files.

1) **Creating a file –** Two steps are necessary to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.

2) **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.

3) **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.

4) **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.

c) File Types –

The name is split into two parts – a name and an extension, usually separated by a period character.

Most operating systems allow users to specify file names as a sequence of characters followed by a period and terminated by an extension of additional characters. File name examples include resume.doc, server.java, and sample.txt.

The system uses the extension to indicate the type of the file. Only a file with a .com, .exe, or .bat extension can be executed.

The .com and .exe files are two forms of binary executable files, whereas a .bat file is a batch file containing, in ASCII format, commands to the operating system.

*** Access Methods –**

Files store information. When it is used, this information must be accessed and read into computer memory. The information in the file can be accessed in several ways.

Some systems provide only one access method for files. Other systems, support many access methods.

a) Sequential Access –

The simplest access method is sequential access. 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 – 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.

Such a file can be reset to the beginning; and on some systems, a program may be able to skip forward or backward ‘n’ records.

Sequential access file, which is shown in following figure, is based on a tape model of a file and works as well on sequential access devices as it does on random access ones.

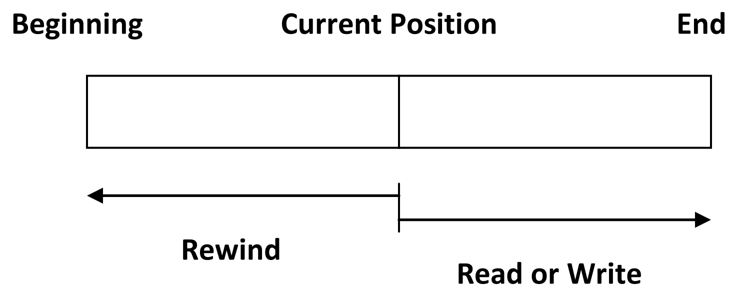


Figure – Sequential access file

b) Direct Access –

Another method is direct access (or relative access). A file is made up of fixed length logical records 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.

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. Direct access files are of great use for immediate access to large amounts of 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', where 'n' is the block number, rather than read next, and write 'n' rather than write next.

Not all operating systems support both sequential and direct access for files. Some systems allow only sequential file access; others allow only direct access.

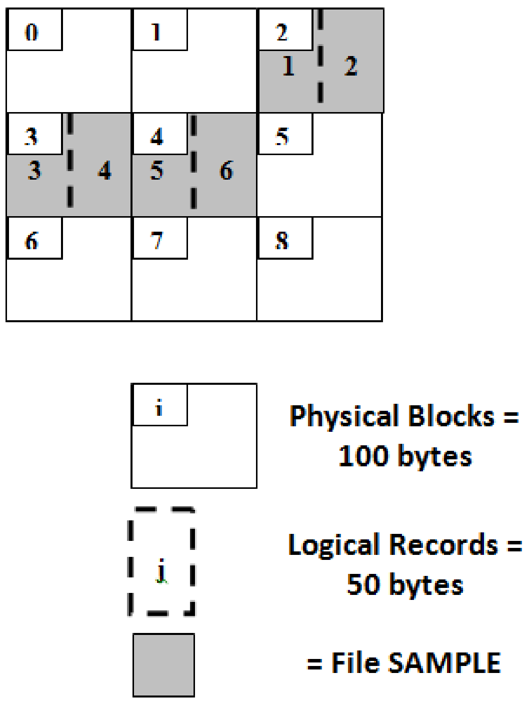


Figure – Physical Block Storage

Suppose, system wants to access record number 5 from the SAMPLE file. System identifies associated block number for that record using following formulas and then directly accessed the block and then accesses required record,

Logical byte address = (Record Number - 1) * Record Length
= (5 - 1) * 50
= 200

Physical Block Number = Logical Byte Address /
Physical Block Size +
Address of First Physical Block

E.g. - Physical Block Number = 200 / 100 + 2
= 2 + 2
= 4

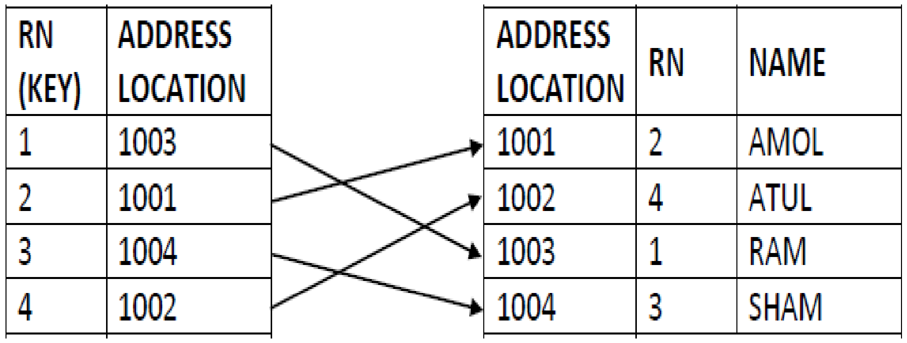
This is the block (4) containing record 5 of file SAMPLE. There is no need to search in sequential order like 2, 3, and then 4. So, seeking time for direct access method is less as compared to the sequential access method

c) Other Access Methods –

If we want to read specific section in book, we would not begin from page 1 & read every page until we across this section. Rather, we would search this section in the table of contents at the beginning of the book called as, 'INDEX'. Index file use same principle.

Following figure shows, in indexed access method, there are two files for every data file i.e. MASTER file & INDEX file. MASTER file contains actual records in a file & INDEX file contains the index key & disk address of each record in the MASTER file.

Records in the MASTER file can be stored in random sequence, but index keys in the index file are stored in sorted sequence on index key value.



RN (KEY)	ADDRESS LOCATION
1	1003
2	1001
3	1004
4	1002

Table - Index File

ADDRESS LOCATION	RN	NAME
1001	2	AMOL
1002	4	ATUL
1003	1	RAM
1004	3	SHAM

Table - Master File

For processing a search request for a particular record, the computer first searches the index file to determine physical location of the record and then accesses the corresponding record from the data file.

For example, to locate a student record who's RN (ROLL NUMBER) is 4. The computer searches index file first for this STUDENT – RN key & obtains the corresponding address value 1002. It then directly accesses the record stored at address location 1002 in the MASTER file.

Advantage –

1) Records are accessed in very quickly.

Disadvantages –

1) It requires extra memory space to store index file.
2) If an index fails to operate, the whole system fails.

*** Directory and Disk Structure –**

A disk can be used in its entirety for a file system. A file system can be created on each of the parts of the disk.

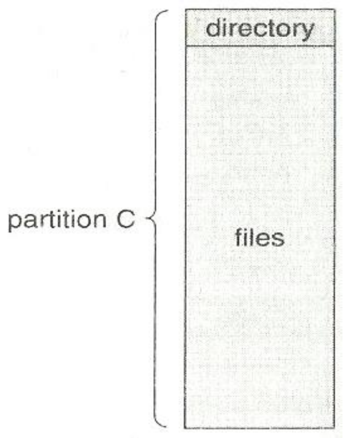


Figure – A typical file system organization

Each volume that contains a file system must also contain information about the files in the system. This information is kept in entries in a device directory or volume table of contents. The device directory records information – such as name, location, size, and type for all files on that volume.

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

1. **Search for a file –** To find a particular file in the directory.
2. **Create a file –** After a new file is created, its entry will be added to the directory.
3. **Delete a file –** Remove the entry of deleted file from the directory.
4. **List a directory –** To see the list of available files.
5. **Rename a file –** We must be able to change the name as per requirement.

a) 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 as shown in following figure,

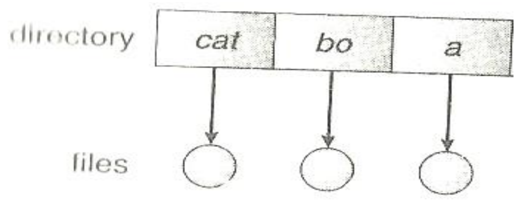


Figure – Single level directory

A single level directory has significant limitations, however, 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.

Even a single user on a single level directory may find it difficult to remember the names of all the files as the number of files increases.

b) Two Level Directory –

A single level directory often leads to confusion of file names among different users. The standard solution is to create a separate directory for each user.

In the two level directory structures, each user has own user file directory (UFD). The UFDs have similar structures, but each lists only the files of a single user.

When a user logs in, the system's master file directory (MFD) is searched. The MFD is indexed by user name or account number, and each entry points to the UFD for that user as shown in following figure,

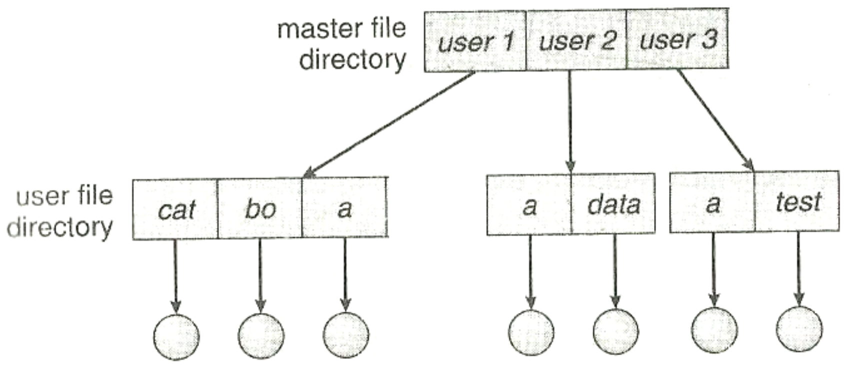


Figure – Two level directory structure

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, the operating system searches only that user's UFD to check whether another file of that name exists. To delete a file, the operating system confines its search to the local UFD.

A two level directory can be thought of as a tree, of height 2. The root of the tree is the MFD. Its direct descendants are the UFDs. The descendants of the UFDs are the files themselves.

c) Tree Structured Directories –

Tree structured directories allows users to create their own subdirectories and to organize their files accordingly. A tree is the most common directory structure.

The tree has a root directory, and every file in the system has a unique path name. A directory (or subdirectory) contains a set of files or subdirectories. A directory is simply another file, but it is treated in a special way.

All directories have the same internal format. One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1).

Path names can be of two types: absolute and relative. An absolute path name begins at the root and follows a path down to the specified file, giving the directory names on the path. A relative path name defines a path from the current directory.

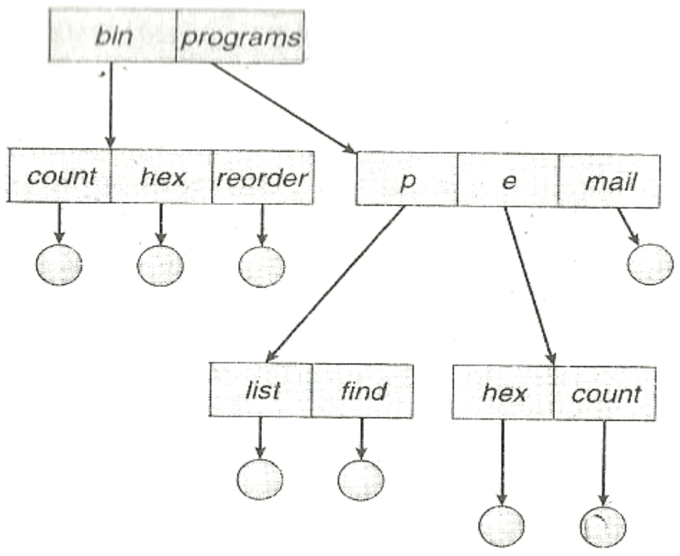


Figure – Tree structured directory structure

For example, in the tree structured file system of above figure, the relative path is p/list and the absolute path is programs/p/list for same file i.e. list file.

If the directory to be deleted is not empty then one of two approaches can be taken. Some systems, such as MS-DOS, will not delete a directory unless it is empty. Thus, to delete a directory, the user must first delete all the files in that directory.

An alternative approach, such as that taken by the UNIX is to provide an option: When a request is made to delete a directory that entire directory's files and sub directories are also to be deleted.

d) Acyclic Graph Directories –

A tree structure prohibits the sharing of files or directories. An acyclic graph – that is, a graph with no cycles – allows directories to share subdirectories and files as shown in following figure.

The common subdirectory should be shared. A shared directory or file will exist in the file system in two (or more) places at once.

It is important to note that a shared file is not the same as two copies of the file. With two copies, each programmer can view the original copy. But if one programmer changes the file, the changes will not appear in the other's copy.

With a shared file, only one actual file exists, so any changes made by one person are immediately visible to the other.

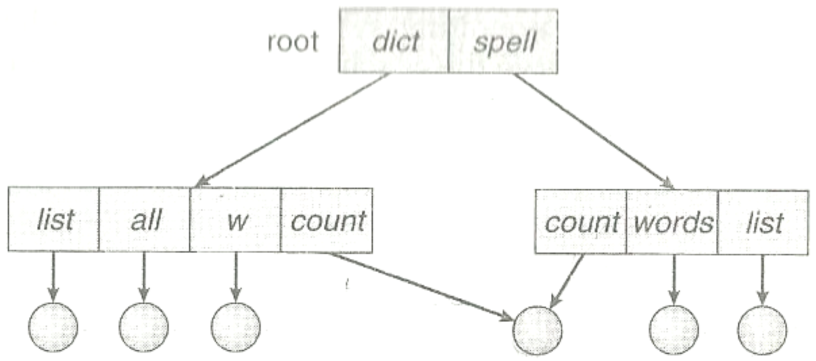


Figure – Acyclic graph directory structure

Sharing is particularly important for subdirectories; a new file created by one person will automatically appear in all the shared subdirectories. An acyclic graph directory structure is more flexible than is a simple tree structure, but it is also more complex.

*** Allocation Methods –**

Three major methods of allocating disk space are in wide use: contiguous, linked, and indexed. Each method has advantages and disadvantages.

a) Contiguous Allocation –

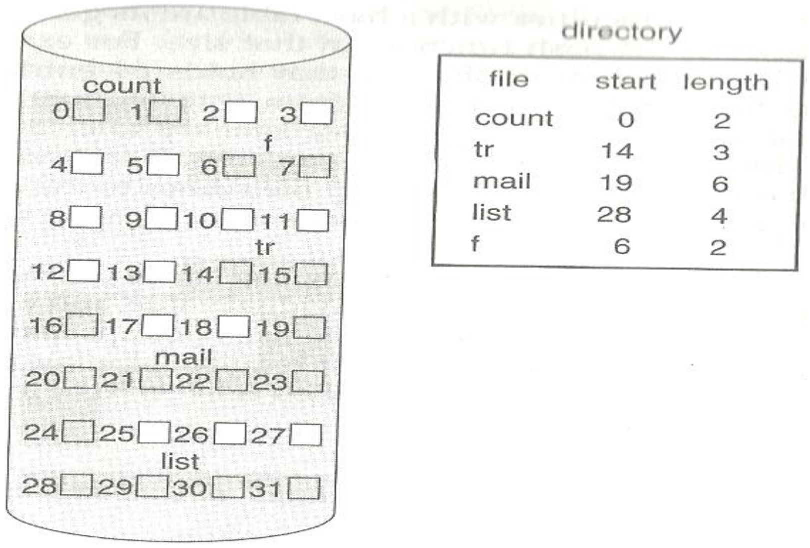
In this method, contiguous blocks are allocated to a file. The directory entry contains name of the file, address of the starting block and the length of the area allocated for this file.

If the file is 'n' blocks long and starts at location 'b', then it occupies blocks b, b + 1, b + 2, ..., b + n - 1.

For sequential access of records, the file system must remember the disk address of the last block read and next block is easily read. For direct access of block "i" of a file, we can directly access $(b + i)^{th}$ block, if block start with b.

First fit and best fit are the most common strategies used to select a free block from the set of available blocks.

Following figure shows the contiguous allocation of files,



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

Figure – Contiguous allocation of disk space

Advantages –

1. It supports both sequential and direct access methods
2. Accessing a file that has been allocated contiguously is easy

Disadvantages –

1. Difficulty in searching space for new files
2. Need to know size of the file in advance
3. External fragmentation
4. Problem is there in the cases where files grow

b) 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 entry contains name of the file, starting address of a pointer and last address of a pointer of the file.

Each block contains a pointer to the next block. These pointers are not made available to the user.

If each block is 512 bytes in size, and a disk address (the pointer) requires 4 bytes, then the user sees blocks of 508 bytes.

Advantages –

1. There is no problem even if size of files grow
2. No need to know size of the file in advance
3. No external fragmentation

Disadvantages –

1. Only sequential access is possible
2. Extra space is required for pointers
3. Reliability is big problem due to pointer lost or damage

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 as shown in following figure,

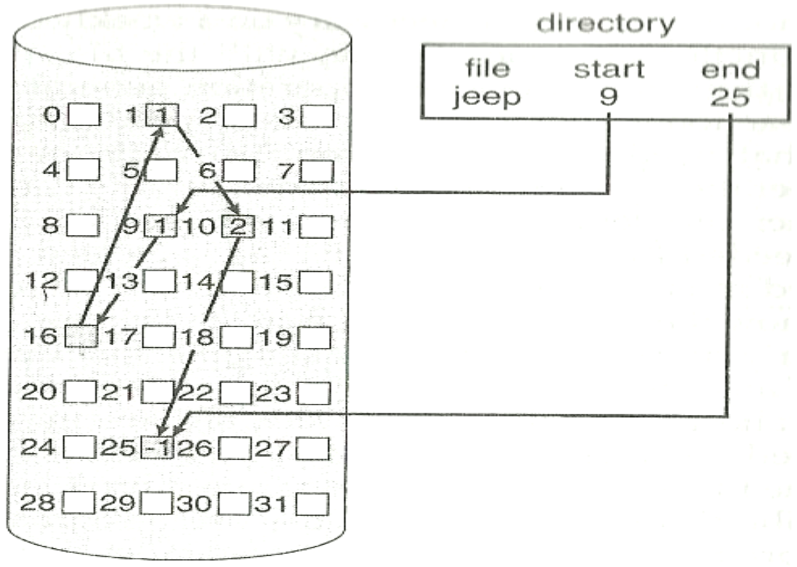


Figure – Linked allocation of disk space

c) Indexed Allocation –

In this method, 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 name of the file and address of the index block as shown in following figure,

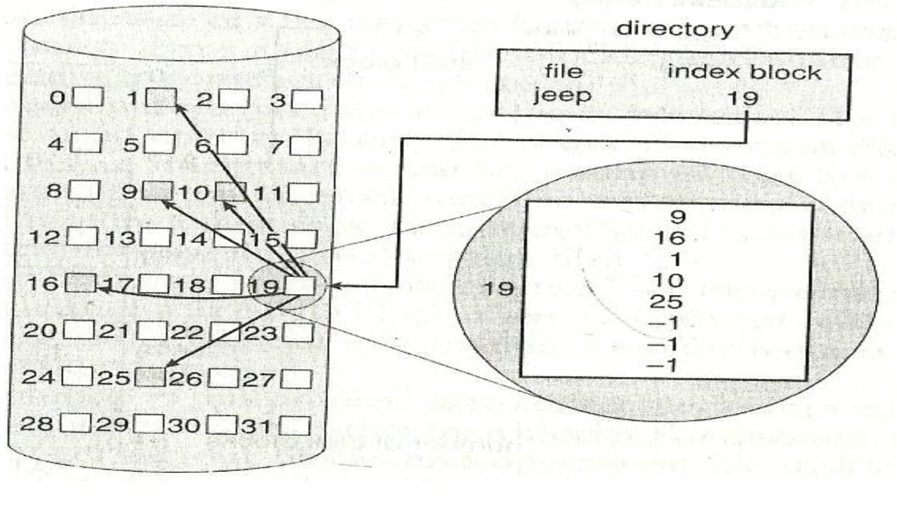


Figure – Indexed allocation of disk space

When the file is created, index block is set to nil. When the ith block is first written, a block is obtained from the free space manager, and its address is put in the ith index block entry.

Advantages –

1. Direct access is possible
2. There is no problem even if size of files grow
3. No need to know size of the file in advance
4. No external fragmentation

Disadvantages –

1. Use of index block is not feasible for very small files
2. File access time is increased because for any operation on the file, index block has to be accessed
3. Extra space is required to maintain index block

*** Free Space Management –**

Disk space is limited; we need to reuse the space from deleted files for new files. 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.

There are several different ways are available to maintain free space list.

1) Bit Vector –

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 9 blocks and blocks 0, 1, 4, 5 and 6 are free and the rest of the blocks are allocated. The free space bit map would be 110011100.

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 disk blocks, keeping a pointer to the first free block in a special location on the disk. This first block contains a pointer to the next free disk block, and so on.

In following 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 are allocated.

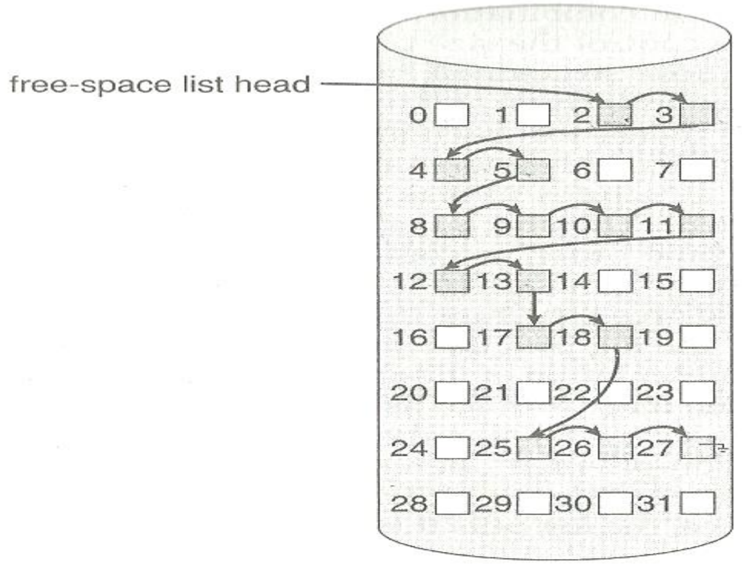


Figure – Linked free space list on disk.

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 above figure.

However; this scheme is not efficient; to traverse the list, we must read each block, which requires substantial I/O time.

3) Grouping –

A modification of the free list approach is to store 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 other 'n' free blocks, and so on.

The addresses of a large number of free blocks can now be found quickly.

Consider following example, there are 12 blocks available. Out of these block no 4, 5, 6, 9, and 10 are allocated and remaining are free. So, there are three groups are possible.

Group 1 – Block no 0, 1, 2, and 3

Group 2 – Block no 7 and 8

Group 3 – Block no 11

In this method, first free block of first group i.e. block 0 contains address(es) of next free block(s) i.e. 1, 2, and 3 as well as last block of first group i.e. block 3 contains address(es) of next free block(s) i.e. 7 and 8. Last block of second group i.e. block 8 contains address(es) of next block(s) i.e. 11 and so on.

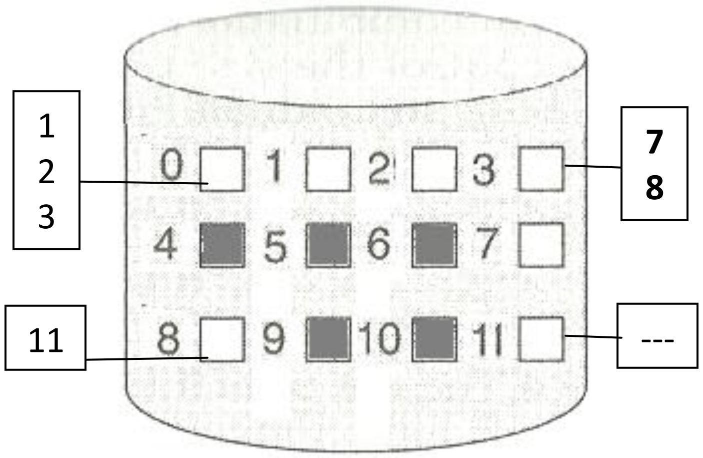


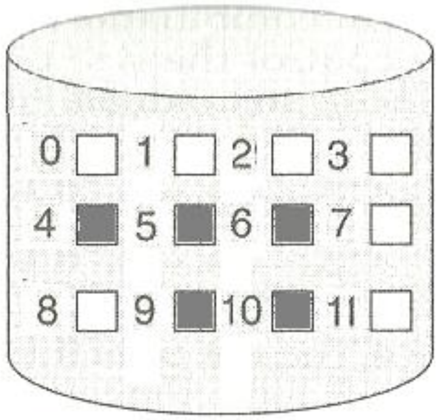
Figure – Grouping

So, in this method out of seven free blocks only three blocks are useful to store user's data and remaining four blocks are used to store addresses of free blocks.

4) Counting –

Generally, several contiguous blocks may be allocated or freed simultaneously, particularly when space is allocated with the contiguous allocation algorithm.

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. So, in this method all seven free blocks are useful to store user's data.



Disk Address	Count
0	4
7	2
11	1

Figure – Counting