Unit 9 File System Interface and Implementation

Structure:

9.1 Introduction

In the last unit, we have discussed about virtual memory and various page replacement algorithms. The operating system is a resource manager. Secondary resources like the disk are also to be managed. Information is stored in secondary storage because it costs less, is non-volatile and provides large storage space. Processes access data / information present on secondary storage while in execution. Thus, the operating system has to properly organize data / information in secondary storage for efficient access.

The file system is the most visible part of an operating system. It is a way for on-line storage and access of both data and code of the operating system and the users. It resides on the secondary storage because of the two main characteristics of secondary storage, namely, large storage capacity and non-volatile nature. This unit will give you an overview of file system interface and its implementation.

Objectives:

After studying this unit, you should be able to:

- explain various file concepts
- discuss different file access methods
- describe various directory structures
- list out and explain various disk space allocation methods
- manage free space on the disk effectively
- implement directory

9.2 Concept of a File

Users use different storage media such as magnetic disks, tapes, optical disks and so on. All these different storage media have their own way of storing information. The operating system provides a uniform logical view of information stored in these different media. The operating system abstracts from the physical properties of its storage devices to define a logical storage unit called a file. These files are then mapped on to physical devices by the operating system during use. The storage devices are usually non-volatile, meaning the contents stored in these devices persist through power failures and system reboots.

The concept of a file is extremely general. A **file** is a collection of related information recorded on the secondary storage. For example, file containing student information, file containing employee information, files containing C source code and so on. A file is thus the smallest allotment of logical secondary storage, that is any information to be stored on the secondary storage need to be written on to a file and the file is to be stored. Information in files could be program code or data in numeric, alphanumeric, alphabetic or binary form either formatted or in free form. A file is therefore a collection of records if it is a data file or a collection of bits / bytes / lines if it is code. Program code stored in files could be source code, object code or executable code whereas data stored in files may consist of plain text, records pertaining to an application, images, sound and so on. Depending on the contents of a file, each file has a pre-defined structure. For example, a file containing text is a collection of characters organized as lines, paragraphs and pages whereas a file containing source code is an organized collection of segments which in turn are organized into declaration and executable statements.

9.2.1 Attributes of a file

A file has a name. The file name is a string of characters. For example, test.c, pay.cob, master.dat, os.doc. In addition to a name, a file has certain other attributes. Important attributes among them are:

- *Type*: Information on the type of file.
- *Location*: The location of the file on the device.
- *Size*: The current size of the file in bytes.
- *Protection*: Control information for user access.
- *Time, date and user id*: Information regarding when the file was created, last modified and last used. This information is useful for protection, security and usage monitoring.

All these attributes of files are stored in a centralized place called the **directory**. The directory is big if the numbers of files are many and also requires permanent storage.

9.2.2 Operations on files

A file is an abstract data type. Six basic operations are possible on files. They are:

1) Creating a file: The two steps in file creation include space allocation for the file and an entry to be made in the directory to record the name and location of the file.
2) Writing a file: The parameters required to write into a file are the name of the file and the contents to be written into it. Given the name of the file the operating system makes a search in the directory to find the location of the file. An updated write pointer enables to write the contents at a proper location in the file.
3) Reading a file: To read information stored in a file the name of the file specified as a parameter is searched by the operating system in the directory to locate the file. An updated read pointer helps read information from a particular location in the file.
4) Repositioning within a file: A file is searched in the directory and a given new value replaces the current file position. No I/O takes place. It is also known as files seek.
5) Deleting a file: The directory is searched for the particular file. If it is found, file space and other resources associated with that file are released and the corresponding directory entry is erased.
6) Truncating a file: In this the file attributes remain the same, but the file has a reduced size because the user deletes information in the file. The end of file pointer is reset.

Other common operations are combinations of these basic operations. They include append, rename and copy. A file on the system is very similar to a manual file. An operation on a file is possible only if the file is open. After performing the operation, the file is closed. All the above basic operations together with the open and close are provided by the operating system as **system calls**.

9.2.3 Types of files

The operating system recognizes and supports different file types. The most common way of implementing file types is to include the type of the file as part of the file name. The attribute 'name' of the file consists of two parts: a name and an extension separated by a period. The extension is the part of a file name that identifies the type of the file. For example, in MS-DOS a file name can be up to eight characters long followed by a period and then a three-character extension. Executable files have a .com / .exe / .bat extension, C source code files have a .c extension, COBOL source code files have a .cob extension and so on.

If an operating system can recognize the type of a file then it can operate on the file quite well. For example, an attempt to print an executable file should be aborted since it will produce only garbage. Another use of file types is the capability of the operating system to automatically recompile the latest version of source code to execute the latest modified program. This is observed in the Turbo / Borland integrated program development environment.

9.2.4 Structure of file

File types are an indication of the internal structure of a file. Some files even need to have a structure that need to be understood by the operating system. For example, the structure of executable files need to be known to the operating system so that it can be loaded in memory and control transferred to the first instruction for execution to begin. Some operating systems also support multiple file structures.

Operating system support for multiple file structures makes the operating system more complex. Hence some operating systems support only a minimal number of files structures. A very good example of this type of operating system is the UNIX operating system. UNIX treats each file as a sequence of bytes. It is up to the application program to interpret a file. Here maximum flexibility is present but support from operating system point of view is minimal. Irrespective of any file structure support, every operating system must support at least an executable file structure to load and execute programs.

Disk I/O is always in terms of blocks. A **block** is a physical unit of storage. Usually all blocks are of same size. For example, each block = 512 bytes. Logical records have their own structure that is very rarely an exact multiple of the physical block size. Therefore a number of logical records are packed into one physical block. This helps the operating system to easily locate an offset within a file. For example, as discussed above, UNIX treats files as a sequence of bytes. If each physical block is say 512 bytes, then the operating system packs and unpacks 512 bytes of logical records into physical blocks.

File access is always in terms of blocks. The logical size, physical size and packing technique determine the number of logical records that can be packed into one physical block. The mapping is usually done by the operating system. But since the total file size is not always an exact multiple of the block size, the last physical block containing logical records is not full. Some part of this last block is always wasted. On an average half a block is wasted. This is termed **internal fragmentation**. Larger the physical block size, greater is the internal fragmentation. All file systems do suffer from internal fragmentation. This is the penalty paid for easy file access by the operating system in terms of blocks instead of bits or bytes.

Self-Assessment Questions

1. The operating system provides a uniform logical view of information stored in different storage media. (True / False)
2. A _______________ is a collection of related information recorded on the secondary storage.
3. Usually a block size will be _______________ bytes. (Pick the right option)
   a) 512
   b) 256
   c) 128
   d) 64

9.3 File Access Methods

Information is stored in files. Files reside on secondary storage. When this information is to be used, it has to be accessed and brought into primary main memory. Information in files could be accessed in many ways. It is usually dependent on an application. Access methods could be:-

- Sequential access
- Direct access
- Indexed sequential access

9.3.1 Sequential access

In this simple access method, information in a file is accessed sequentially one record after another. To process the i^{th} record all the i-1 records previous to i must be accessed. Sequential access is based on the tape model that is inherently a sequential access device. Sequential access is best suited where most of the records in a file are to be processed. For example, transaction files.

9.3.2 Direct access

Sometimes it is not necessary to process every record in a file. It may not be necessary to process records in the order in which they are present. Information present in a record of a file is to be accessed only if some key value in that record is known. In all such cases, direct access is used. Direct access is based on the disk that is a direct access device and allows random access of any file block. Since a file is a collection of physical blocks, any block and hence the records in that block are accessed. For example, master files. Databases are often of this type since they allow query processing that involves immediate access to large amounts of information. All reservation systems fall into this category. Not all operating systems support direct access files. Usually files are to be defined as sequential or direct at the time of creation and accessed accordingly later. Sequential access of a direct access file is possible but direct access of a sequential file is not.

9.3.3 Indexed sequential access

This access method is a slight modification of the direct access method. It is in fact a combination of both the sequential access as well as direct access. The main concept is to access a file direct first and then sequentially from that point onwards. This access method involves maintaining an index. The index is a pointer to a block. To access a record in a file, a direct access of the index is made. The information obtained from this access is used to access the file. For example, the direct access to a file will give the block address and within the block the record is accessed sequentially. Sometimes indexes may be big. So a hierarchy of indexes is built in which one direct access of an index leads to info to access another index directly and so on till the actual file is accessed sequentially for the particular record. The main advantage in this type of access is that both direct and sequential access of files is possible.

9.4 Directory Structure

Files systems are very large. Files have to be organized. Usually a two level organization is done:

- The file system is divided into partitions. By default there is at least one partition. Partitions are nothing but virtual disks with each partition considered as a separate storage device.
- Each partition has information about the files in it. This information is nothing but a table of contents. It is known as a directory.

The directory maintains information about the name, location, size and type of all files in the partition. A directory has a logical structure. This is dependent on many factors including operations that are to be performed on the directory like search for file/s, create a file, delete a file, list a directory, rename a file and traverse a file system. For example, the *dir*, *del*, *ren* commands in MS-DOS.

9.4.1 Single-level directory

This is a simple directory structure that is very easy to support. All files reside in one and the same directory (See figure 9.1).

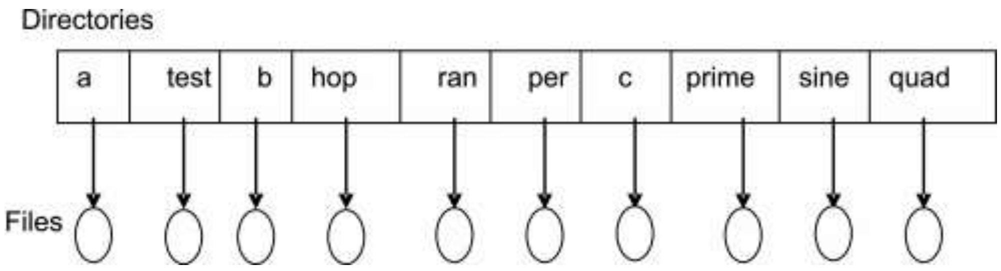


Figure 9.1: Single-level directory structure

A single-level directory has limitations as the number of files and users increase. Since there is only one directory to list all the files, no two files can have the same name i.e., file names must be unique in order to identify one file from another. Even with one user, it is difficult to maintain files with unique names when the number of files becomes large.

9.4.2 Two-level directory

The main limitation of single-level directory is to have unique file names by different users. One solution to the problem could be to create separate directories for each user.

A two-level directory structure has one directory exclusively for each user. The directory structure of each user is similar in structure and maintains file information about files present in that directory only. The operating system has one master directory for a partition. This directory has entries for each of the user directories (Refer figure 9.2).

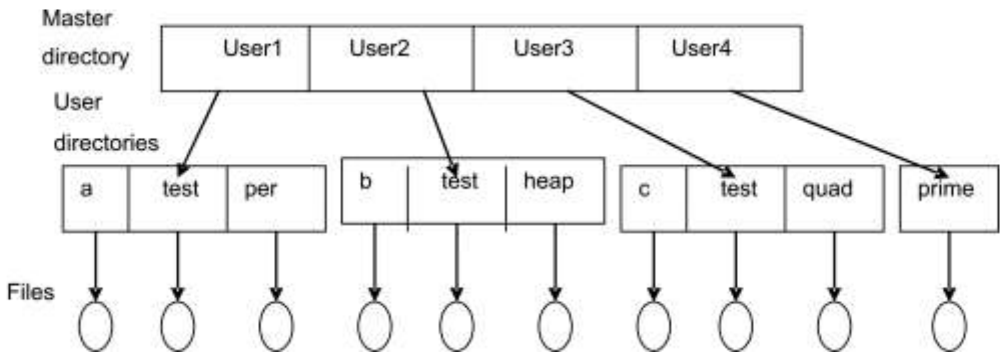


Figure 9.2: Two-level directory structure

Files with same names exist across user directories but not in the same user directory. File maintenance is easy. Users are isolated from one another. But when users work in a group and each wants to access files in another user's directory, it may not be possible.

Access to a file is through user name and file name. This is known as a path. Thus a path uniquely defines a file. For example, in MS-DOS if 'C' is the partitions then C:\USER1\TEST, C:\USER2\TEST, C:\USER3\C are all files in user directories. Files could be created, deleted, searched and renamed in the user directories only.

9.4.3 Tree-structured directories

A tree structured directory is a tree of height two with the master file directory at the root having user directories as descendants that in turn have the files themselves as descendants (Refer figure 9.3). This generalization allows users to organize files within user directories into sub directories. Every file has a unique path. Here the path is from the root through all the sub directories to the specific file.

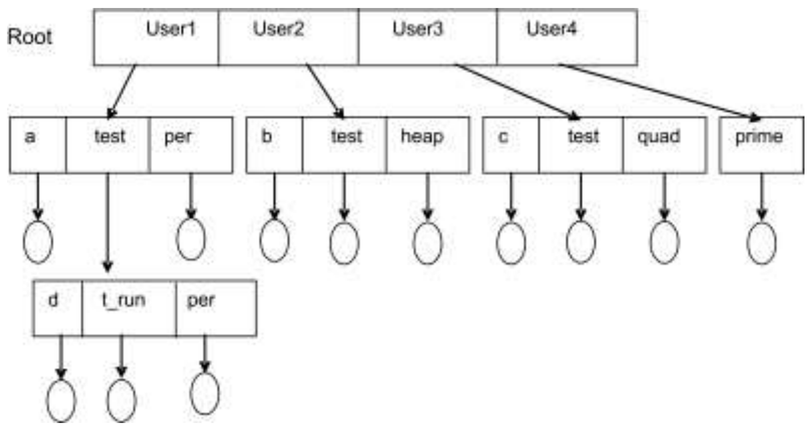


Fig. 9.3: Tree-structured directory structure

Usually the user has a current directory. User created sub directories could be traversed. Files are usually accessed by giving their path names. Path names could be either absolute or relative. Absolute path names begin with the root and give the complete path down to the file. Relative path names begin with the current directory. Allowing users to define sub directories allows for organizing user files based on topics. A directory is treated as yet another file in the directory, higher up in the hierarchy. To delete a directory it must be empty. Two options exist: delete all files and then delete the directory or delete all entries in the directory when the directory is deleted. Deletion may be a recursive process since directory to be deleted may contain sub directories.

Self Assessment Questions

4. The CPU can directly process the information stored in secondary memory without transferring it to main memory. (True / False)
5. In _______________ method, information in a file is accessed sequentially one record after another.
6. A _______________ is a tree of height two with the master file directory at the root having user directories as descendants that in turn have the files themselves as descendants. (Pick the right option)
   a) Single level directory
   b) Tree structured directory
   c) Two level directory
   d) Directory

9.5 Allocation Methods

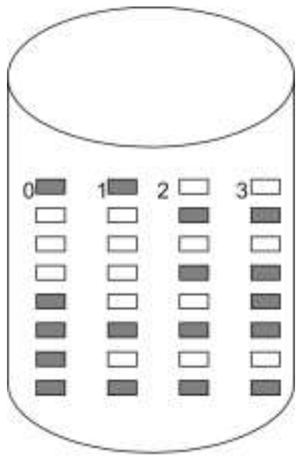
Allocation of disk space to files is a problem that looks at how effectively disk space is utilized and quickly files can be accessed. The three major methods of disk space allocation are:

- Contiguous allocation
- Linked allocation
- Indexed allocation

9.5.1 Contiguous allocation

Contiguous allocation requires a file to occupy contiguous blocks on the disk. Because of this constraint disk access time is reduced, as disk head movement is usually restricted to only one track. Number of seeks for accessing contiguously allocated files is minimal and so also seek times.

A file that is 'n' blocks long starting at a location 'b' on the disk occupies blocks b, b+1, b+2, ….., b+(n-1). The directory entry for each contiguously allocated file gives the address of the starting block and the length of the file in blocks as illustrated below (See figure 9.4).



Directory:

File	Start	Length
a.	0	2
per	14	3
hop	19	6
b	28	4
f	6	2

Figure 9.4: Contiguous allocation

Accessing a contiguously allocated file is easy. Both sequential and random access of a file is possible. If a sequential access of a file is made then the next block after the current is accessed, whereas if a direct access is made then a direct block address to the ith block is calculated as b+i where b is the starting block address.

A major disadvantage with contiguous allocation is to find contiguous space enough for the file. From a set of free blocks, a first-fit or best-fit strategy is adopted to find ‘n’ contiguous holes for a file of size ‘n’. But these algorithms suffer from external fragmentation. As disk space is allocated and released, a single large hole of disk space is fragmented into smaller holes. Sometimes the total size of all the holes put together is larger than the size of the file size that is to be allocated space. But the file cannot be allocated in that space because there is no contiguous hole of size equal to that of the file. This is when external fragmentation has occurred. Compaction of disk space is a solution to external fragmentation. But it has a very large overhead.

Another problem with contiguous allocation is to determine the space needed for a file. The file is a dynamic entity that grows and shrinks. If allocated space is just enough (a best-fit allocation strategy is adopted) and if the file grows, there may not be space on either side of the file to expand. The solution to this problem is to again reallocate the file into a bigger space and release the existing space. Another solution that could be possible if the file size is known in advance is to make an allocation for the known file size. But in this case there is always a possibility of a large amount of internal fragmentation because initially the file may not occupy the entire space and also grow very slowly.

9.5.2 Linked allocation

Linked allocation overcomes all problems of contiguous allocation. A file is allocated blocks of physical storage in any order. A file is thus a list of blocks that are linked together. The directory contains the address of the starting block and the ending block of the file. The first block contains a pointer to the second, the second a pointer to the third and so on till the last block (Refer figure 9.5)

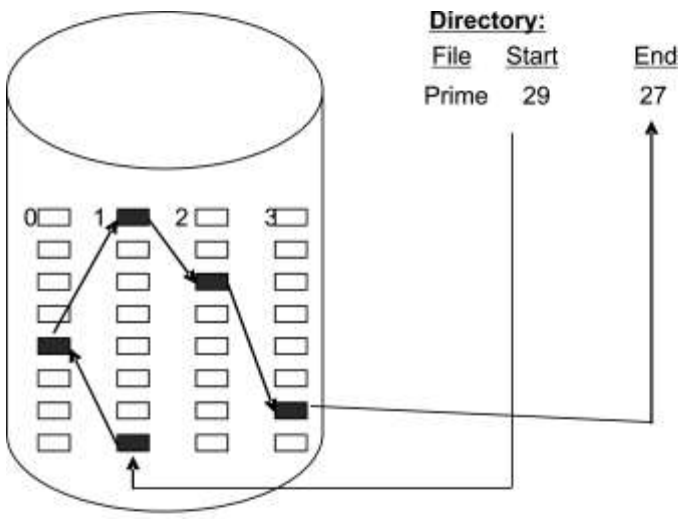


Fig. 9.5: Linked allocation

Initially a block is allocated to a file, with the directory having this block as the start and end. As the file grows, additional blocks are allocated with the current block containing a pointer to the next and the end block being updated in the directory.

This allocation method does not suffer from external fragmentation because any free block can satisfy a request. Hence there is no need for compaction. Moreover a file can grow and shrink without problems of allocation.

Linked allocation has some disadvantages. Random access of files is not possible. To access the i^{th} block access begins at the beginning of the file and follows the pointers in all the blocks till the i^{th} block is accessed. Therefore access is always sequential. Also some space in all the allocated blocks is used for storing pointers. This is clearly an overhead as a fixed percentage from every block is wasted. This problem is overcome by allocating blocks in clusters that are nothing but groups of blocks. But this tends to increase internal fragmentation. Another problem in this allocation scheme is that of scattered pointers. If for any reason a pointer is lost, then the file after that block is inaccessible. A doubly linked block structure may solve the problem at the cost of additional pointers to be maintained.

MS-DOS uses a variation of the linked allocation called a **File Allocation Table** (FAT). The FAT resides on the disk and contains entry for each disk block and is indexed by block number. The directory contains the starting block address of the file. This block in the FAT has a pointer to the next block and so on till the last block (Refer figure 9.6). Random access of files is possible because the FAT can be scanned for a direct block address.

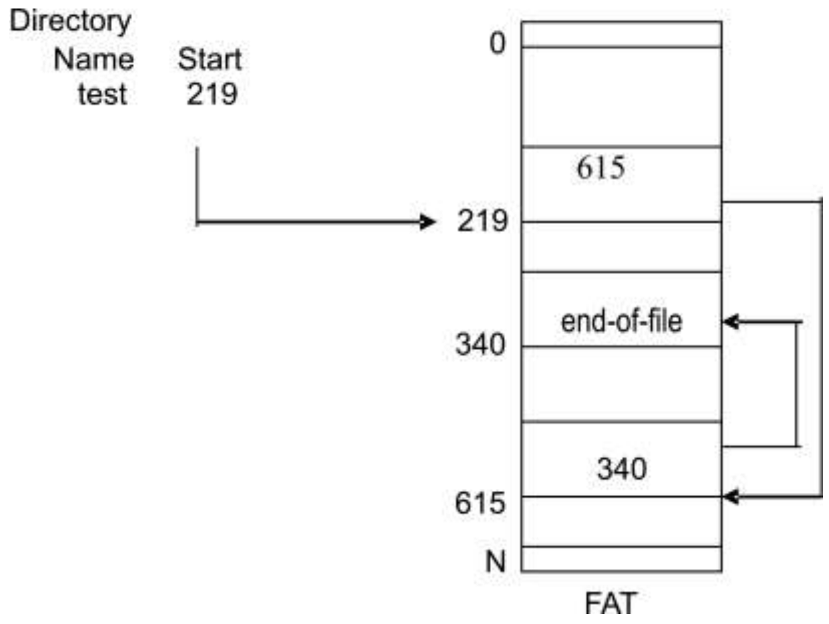


Fig. 9.6: File allocation table

9.5.3 Indexed allocation

Problems of external fragmentation and size declaration present in contiguous allocation are overcome in linked allocation. But in the absence of FAT, linked allocation does not support random access of files since pointers hidden in blocks need to be accessed sequentially. Indexed allocation solves this problem by bringing all pointers together into an index block. This also solves the problem of scattered pointers in linked allocation. Each file has an index block. The address of this index block finds an entry in the directory and contains only block addresses in the order in which they are allocated to the file. The i^{th} address in the index block is the i^{th} block of the file (Refer figure 9.7). Here both sequential and direct access of a file is possible. Also it does not suffer from external fragmentation.

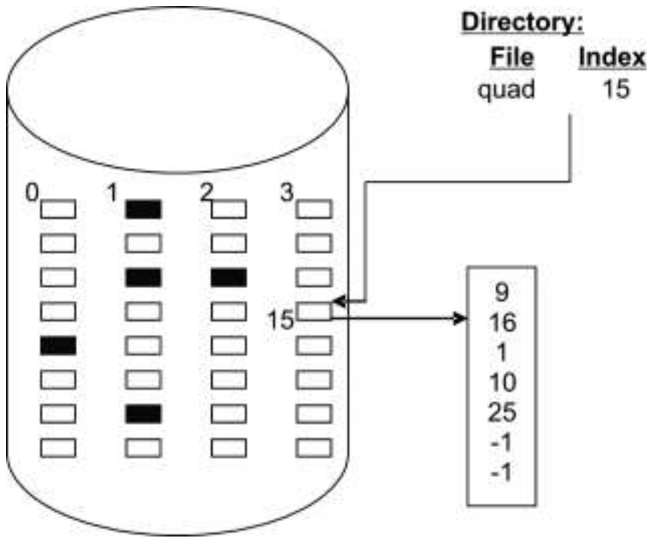


Fig. 9.7: Indexed Allocation

Indexed allocation does suffer from wasted block space. Pointer overhead is more in indexed allocation than in linked allocation. Every file needs an index block. Then what should be the size of the index block? If it is too big, space is wasted. If it is too small, large files cannot be stored. More than one index blocks are linked so that large files can be stored. Multilevel index blocks are also used. A combined scheme having direct index blocks as well as linked index blocks has been implemented in the UNIX operating system.

9.5.4 Performance comparison

All the three allocation methods differ in storage efficiency and block access time. Contiguous allocation requires only one disk access to get a block, whether it is the next block (sequential) or the ith block (direct). In the case of linked allocation, the address of the next block is available in the current block being accessed and so is very much suited for sequential access. Hence direct access files could use contiguous allocation and sequential access files could use linked allocation. But if this is fixed then the type of access on a file needs to be declared at the time of file creation. Thus a sequential access file will be linked and cannot support direct access. On the other hand a direct access file will have contiguous allocation and can also support sequential access; the constraint in this case is making known the file length at the time of file creation. The operating system will then have to support algorithms and data structures for both allocation methods.

Conversion of one file type to another needs a copy operation to the desired file type.

Some systems support both contiguous and linked allocation. Initially all files have contiguous allocation. As they grow a switch to indexed allocation takes place. If on an average files are small, than contiguous file allocation is advantageous and provides good performance.

9.6 Free Space Management

The disk is a scarce resource. Also disk space can be reused. Free space present on the disk is maintained by the operating system. Physical blocks that are free are listed in a free-space list. When a file is created or a file grows, requests for blocks of disk space are checked in the free-space list and then allocated. The list is updated accordingly. Similarly, freed blocks are added to the free-space list. The free-space list could be implemented in many ways as follows:

9.6.1 Bit vector

A bit map or a bit vector is a very common way of implementing a free-space list. This vector 'n' number of bits where 'n' is the total number of available disk blocks. A free block has its corresponding bit set (1) in the bit vector whereas an allocated block has its bit reset (0).

Illustration: If blocks 2, 4, 5, 9, 10, 12, 15, 18, 20, 22, 23, 24, 25, 29 are free and the rest are allocated, then a free-space list implemented as a bit vector would look as shown below:

001011000110100100101011110001000000………

The advantage of this approach is that it is very simple to implement and efficient to access. If only one free block is needed then a search for the first '1' in the vector is necessary. If a contiguous allocation for 'b' blocks is required, then a contiguous run of 'b' number of 1's is searched. And if the first-fit scheme is used then the first such run is chosen and the best of such runs is chosen if best-fit scheme is used.

Bit vectors are inefficient if they are not in memory. Also the size of the vector has to be updated if the size of the disk changes.

9.6.2 Linked list

All free blocks are linked together. The free-space list head contains the address of the first free block. This block in turn contains the address of the next free block and so on. But this scheme works well for linked allocation. If contiguous allocation is used then to search for ‘b’ contiguous free blocks calls for traversal of the free-space list which is not efficient. The FAT in MS-DOS builds in free block accounting into the allocation data structure itself where free blocks have an entry say –1 in the FAT.

9.6.3 Grouping

Another approach is to store ‘n’ free block addresses in the first free block. Here (n-1) blocks are actually free. The last nth address is the address of a block that contains the next set of free block addresses. This method has the advantage that a large number of free block addresses are available at a single place unlike in the previous linked approach where free block addresses are scattered.

9.6.4 Counting

If contiguous allocation is used and a file has freed its disk space then a contiguous set of ‘n’ blocks is free. Instead of storing the addresses of all these ‘n’ blocks in the free-space list, only the starting free block address and a count of the number of blocks free from that address can be stored. This is exactly what is done in this scheme where each entry in the free-space list is a disk address followed by a count.

9.7 Directory Implementation

The two main methods of implementing a directory are:

- Linear list
- Hash table

9.7.1 Linear list

A linear list of file names with pointers to the data blocks is one way to implement a directory. A linear search is necessary to find a particular file. The method is simple but the search is time consuming. To create a file, a linear search is made to look for the existence of a file with the same file name and if no such file is found the new file created is added to the directory at the end. To delete a file, a linear search for the file name is made and if found allocated space is released. Every time making a linear search consumes time and increases access time that is not desirable since directory information is frequently used. A sorted list allows for a binary search that is time efficient compared to the linear search. But maintaining a sorted list is an overhead especially because of file creations and deletions.

9.7.2 Hash table

Another data structure for directory implementation is the hash table. A linear list is used to store directory entries. A **hash table** takes a value computed from the file name and returns a pointer to the file name in the linear list. Thus search time is greatly reduced. Insertions that are prone to collisions are resolved. The main problem is the hash function that is dependent on the hash table size. A solution to the problem is to allow for chained overflow with each hash entry being a linked list. Directory lookups in a hash table are faster than in a linear list.

Self Assessment Questions

7. In contiguous allocation the disk access time is reduced, as disk head movement is usually restricted to only one track. (True / False)
8. FAT stands for _______________.
9. If the file size is small, then _______________ is advantageous and provides good performance. (Pick the right option)
   a) Linked allocation
   b) Indexed allocation
   c) Contiguous file allocation
   d) Linked with indexed allocation

9.8 Summary

Let's recapitulate important points:

- The operating system is a secondary resource manager.
- Data / information stored in secondary storage have to be managed and efficiently accessed by executing processes. To do this the operating system uses the concept of a file.
- A file is the smallest allotment of secondary storage. Any information to be stored needs to be written on to a file.
- There are three methods of file access viz. Sequential, Direct and Indexed sequential.
- Contiguous allocation, linked allocation and indexed allocation are various space allocation methods.

9.9 Terminal Questions

1. Explain different file access methods in detail.
2. What are the different directory structures available? Explain.
3. Discuss various space allocation methods for files.

9.10 Answers

Self-Assessment Questions

1. True
2. File
3. a) 512
4. False
5. Sequential Access
6. b) Tree Structured Directory
7. True
8. File Allocation Table
9. c) Contiguous File Allocation

Terminal Questions

1. Information in files could be accessed in many ways. It is usually dependent on an application. Access methods could be:-
   a. Sequential access
   b. Direct access
   c. Indexed sequential access (Refer Section 9.3)
2. The directory maintains information about the name, location, size and type of all files in the partition. A directory has a logical structure. This is dependent on many factors including operations that are to be performed on the directory like search for file/s, create a file, delete a file, list a directory, rename a file and traverse a file system. (Refer Section 9.4)
3. Allocation of disk space to files is a problem that looks at how effectively disk space is utilized and quickly files can be accessed. The three major methods of disk space allocation are:
   a. Contiguous allocation
   b. Linked allocation
   c. Indexed allocation (Refer Section 9.5)