CHAPTER - 10

FILE MANAGEMEMT

10.1 Overview

Computers can store the information on different storage media, such as magnetic disks, tapes, compact disks. The physical storage is converted in to a logical storage unit by operating system. The logical storage unit is said to be the 'file'. File s are mapped by the operating system ,onto physical devices. A file is a collection of similar records , with a common name. A record is a collection of related fields that can be treated as a unit by some application program. A field is the some basic element of data. Any individual field contains single value. A data base is a collection of related data. consider the figure 10.1 to illustrate the file.

Student number	Name	Marks in SUB1	Marks in SUB2	Marks in SUB3	Fail/pass
1001	RAM	65	73	34	F
1002	RABERT	56	63	72	P
1003	KUMAR	45	65	53	P
1004	JANI	60	71	81	P

Figure 10.1 A Data File

Student number ,name, marks in SUB1, marks in SUB2, marks in SUB3, fail or pass these are the fields (or) data element. The collection of these data elements is called a record.

For example

1003	KUMAR	45	65	53	P

10.1 Record

Collection of these records called a data files. A data base may contain all of the information related to an organization or project, such as a business or a scientific study. So a data base consisting of different types of data files.

Operations on data base files

Retrieve –all: This command retrieve the all records of a file. It must be useful to perform the operations on the entire file (all records)at a time. For example we want to delete all the records in the student data base file this command is useful at that time.

Retrieve-one: This command retrieves one record at a time, interactive, transaction oriented applications needs this command. Suppose we want to modify a single record or delete a single record at that time we were using.

Retrieve- next: This command retrieves the next record , which is in the next record to the recently retrieved one. Some interactive applications , such as filling in forms, may require such operation.

Retrieve-previous: This command retrieve the previous record , which is the previous record to the recently retrieved one.

Insert-one: Insert a new record to the exiting data base file.

Delete-one: Delete an exiting record, which is specified by the user.

Update-one: Retrieve a record, update one or more of its fields, and rewrites the updated record back into the file.

Retrieve – few: Retrieve a number of records. For example an application or user may wish to retrieve all records that satisfy a certain set of criteria.

Operations on programmable files: The basic operations of the programmable files are create a file, writing a file, reading a file, repositioning with in a file, deleting a file and truncating a file. The operating system provides system calls to create, write, read, reposition, delete and truncate files.

Creating a file: Two steps are needed to create a file. First one is check weather the space is available or not. If the space is available then made an entry for the new file must be made in the directory. The entry includes name of the file, path of the file etc.

Writing a file: To write a file, we have to know two things one is name of the file and second is the information or data to be written on the file, the system searches the entired given locations for the file. If the file is found, the system must keep a write pointer to the location in the file where the next write is to take place.

Reading a file: To read a file , first of all we search the directories for the file, if the file is found , the system needs to keep a 'read' pointer to the location in the file where the next read is to take place.once the read has taken place , the read pointer is updated.

Repositioning with in a file: This file operation is also known as file seek. The directory is searched for the appropriate entry, and current file position is set to a given value.

Deleting files: To delete a file , first of all search the directory for named file, then released the file space and erase the directory entry.

Truncating a file: To truncate a file , remove the file contents only , but the attributes are as it is.

File attributes: Name , type, location , size, protection, time , date , user identification these are the attributes of a file.

Name: A file is named, for the convenience of the user, and is referred to by its name. A name is usually a string of characters. The symbolic file name is the only information kept in human readable form.

Type: Files are so many types, the type is depending on the extension of the file. For example

- .exe : executable file.
- .obj : object file.
- .src : source file.

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

Size: The current size of the file(in bytes, blocks)

Protection: It specifies the access -control information. It controls who can do reading, writing, executing and so on.

Time: It specifies time of creation.

Date: It specifies the file created date.

User identification: This is useful for protection security and usage monitoring.

File types: generally the name of the file spilt in to two parts one is name and second is extension. The file type is depending on extension of the file. For example in MS-DOS a name can allowed up to 8 characters followed by a period and terminated by an up to three characters extension. For example consider the figure 10.2 it displays the file types.

File type	Extension	Purpose/function
Executable	.exe .com .bin or none	Ready to run (or) ready to run machine language program.

Figure 10.2 Common file types

Source code	.c .cpp .pas .asm .f77	Source code in various languages.
Batch	.bat .sh	Commands to the command interpreter.
Text	.txt .doc	Textual data documents
Word processor	.wp .rrf .etc	Various word processor format
Library	.lib .a	Libraries of routines for programmers.
Point (or) view	.ps .dvi .sif	ASCII or binary file in a format for printing or viewing.
Archive	.arc .zip .tor	Grouped files , compressed files for archiving or storage.

Figure 10.2 common file types, continued

10.2 File management system

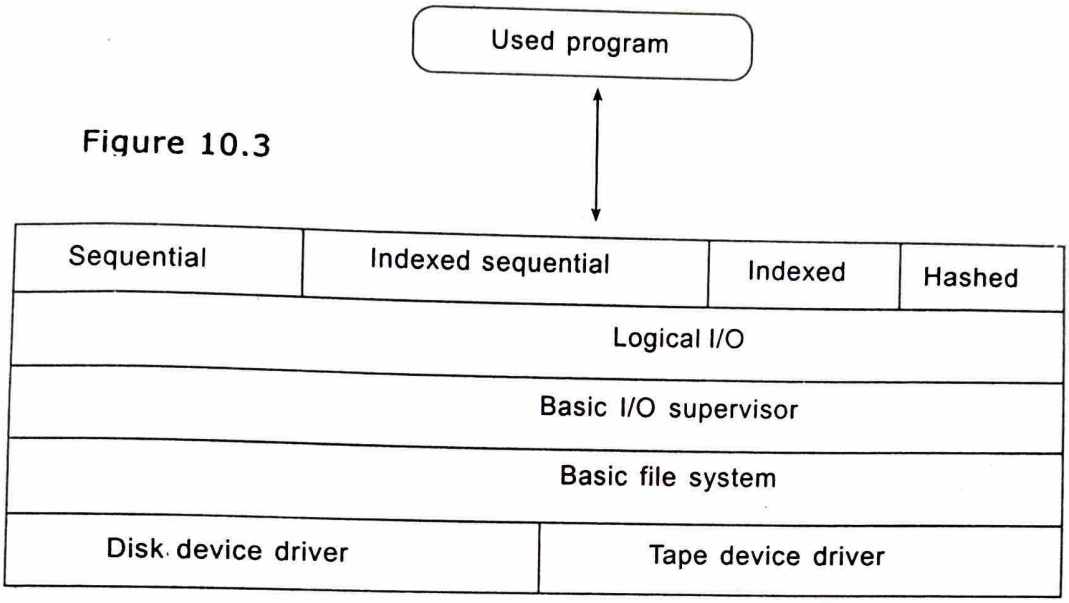
A file management system (FMS) is a collection of system software programs that provides services to the users. For example the user want to create a file ,delete file, it must be done through the FMS. Generally the objectives of the file management system are

- To provide the I/O support for multi users.
- To provide a standardize set of I/O interface routines.
- Storage of data.
- To optimize performance.
- To provide I/O support for a variety of storage device types.
- To guarantee that the data in the file are valid.

File system architecture

Consider the figure 10.3 , it depicts the file system architecture.

Figure 10.3



At the lowest level, device drivers communicate directly with peripheral devices or their controllers or channels. A device driver is responsible for starting I/O operations on a device and processing the completion of I/O request. The next level is the 'Basic file system'. It is the interface with the environment outside of the computer system. The 'basic I/O supervisor' is responsible for all file I/O initiation and termination. ' Logical I/O ' enables users and applications to access records.

10.3 File accessing methods

Files stores information , this information must be accessed and read in to computer memory. There are so many ways that the information in the file can be accessed.

1. sequential access,.
2. Direct access.
3. Indexed sequential access.

10.3.1 Sequential file access

This method is the simplest of all methods. Information in the file is processed in order, one record after the other. Magnetic tapes are supporting this type of file accessing. For example a file consisting of 100 records, the current position of read/write head is 45th record , suppose we want to read the 75th record then it access sequentially from 45,46,47.....73,74,75. So the read/write head traverse all the records between 45 to 75. Consider the figure 10.4 for better understanding.

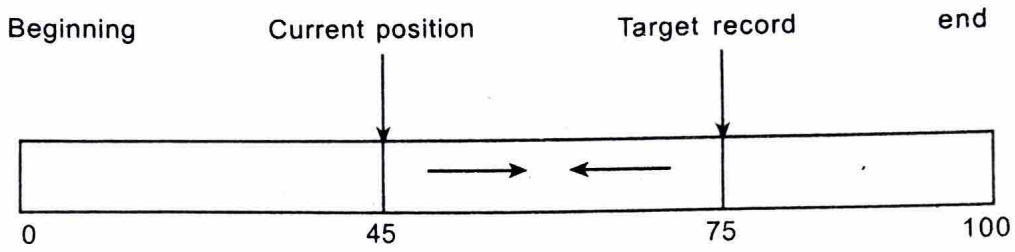


Figure 10.4 Sequential file access

Sequential files are typically used in batch applications and payroll applications.

10.3.2 Direct access

Direct access is also called relative access. In this method records can read /write randomly with out any order. The direct access method is based on a disk model of a file, because disks allow random access to any file block. A direct access file allows arbitrary blocks to be read or written. For example a disk consisting of 256 blocks, the current position of read/write head is at 95th block. The block to be read or write is 250th block. Then we can access the 250th block directly without any restrictions. Best example for direct access is a CD consisting of 10 songs, at present we are listening the song no:3, suppose we want listening the song no:9then we can shift from song no: 3 to 9 with out any restrictions.

10.3.3 Indexed sequential file

The main disadvantage in the sequential file is , it takes more time to access a record, we can overcome this problem in this method. Records are organized in sequence based on a key field. For better understanding consider the figure 10.5.

For suppose a file consisting of 60,000 records, the master index divide the total records in to 6 blocks, each block consisting of a pointer to secondary index. The secondary index divide the 10,000 records in to 10 indexes. Each index consisting of a pointer to its original location. Each record in the index file consisting of two fields. A key field and a pointer field. Suppose we want to access the 55,550th record . the FMS access the index that is 50,000 to 60,000, this block consisting of a pointer , this pointer points to the 6th index in the secondary index. This index points to the original location of the record from 55,000 to 56,000. From this it follows the sequential method. That's why this method is said to be the indexed sequential file. Generally indexed files are used in airline reservation systems and payroll systems.

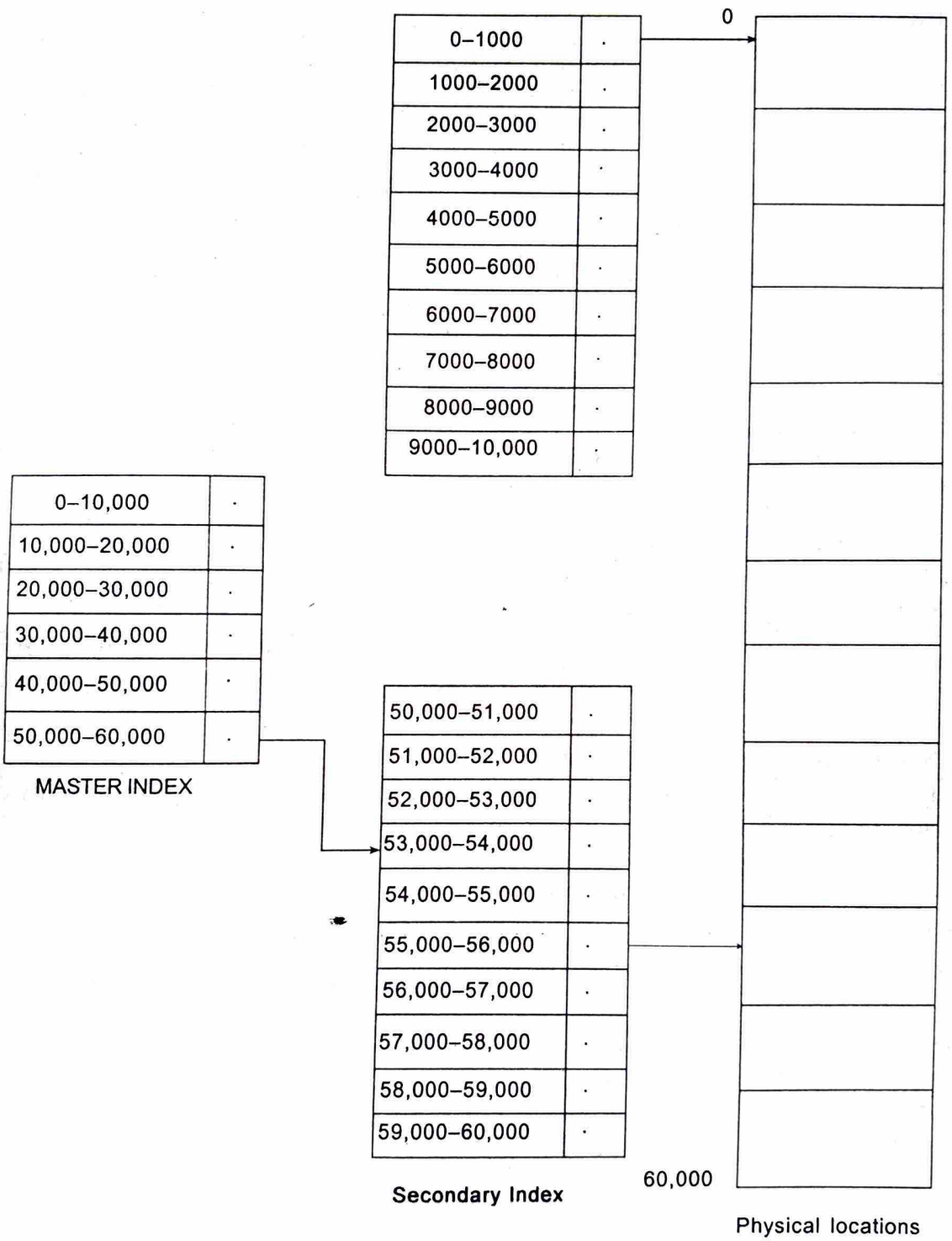


Figure 10.5 Indexed sequential file

10.4 File directories

The directory contains information about the files , including attributes , locations and ownership. Some times the directories consisting of subdirectories also. The directory is it self a file, owned by the operating system and accessible by various file management routines.

Directory structure

Some times the file system consisting of millions of files, at that situation it is very hard to manage the files. To manage these files grouped these files and load one group in to one partition. Each partition is called a directory. The directory can be viewed as a symbol table that translate filenames in to their directory entries. A directory structure provides a mechanism for organizing many files in the file system.

Operations on the file directories:

The operations that can be performed on a directory are as follows.

Search for a file:	Search the directory structure for required file.
Create a file:	Whenever we create a file , should make an entry in the directory.
Delete a file:	When a file is no longer needed , we want to remove it from the directory.
List a directory:	We can know the list of files in the directory.
Rename a file:	whenever we need to change the name of the file we can change the name.
Traverse the file system:	We need to access every directory , and every file with in a directory structure we can traverse the file system.

Single level directory system

It is simplest of all directory structures, in this the directory system having only one directory, it consisting of the all files. Some times it is said to be ' Root directory'. For example consider the figure 10.6. here is directory contains 4 files (A,B,C,D)

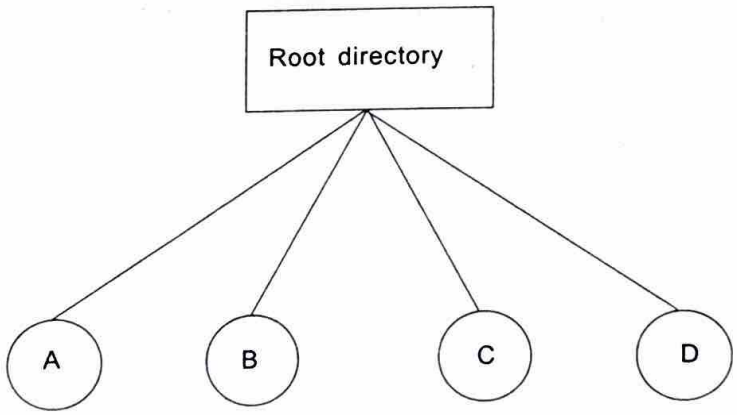


Figure 10.6 Single level directory

The advantage of this scheme are its simplicity and the ability to locate files quickly. The problem with single level directory is different users may accidentally use the same names for their files., for example , if user1 creates a file called sample, and then later user2 creates a file called sample, then user2's file will overwrite A's file. That 's why it is not used in the multi-user system , it is used on small embedded system.

Two-level directory structure

The problem in single level directory is different users may be accidentally use the same names for their files. To avoid this problem each user need a private directory. In this way names chosen by one user don't interfere with names chosen by a different user and there is no problem caused by the same occurring in two or more directories. Consider the figure 10.7 for better understanding.

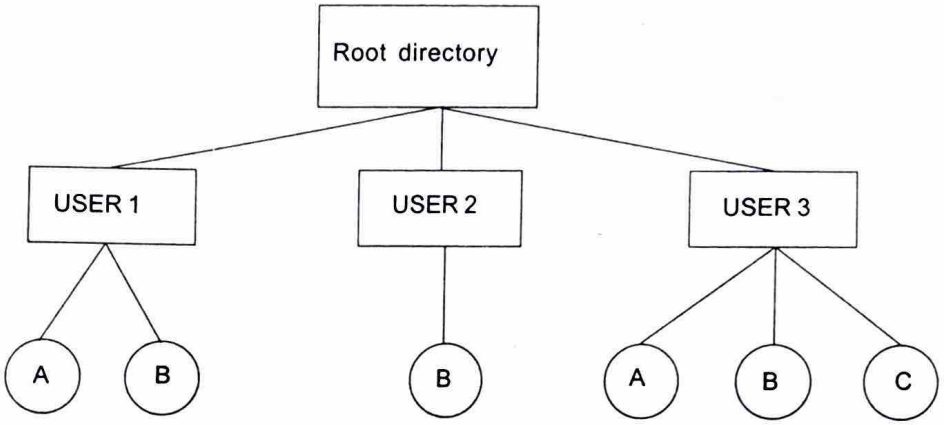


Figure 10.7 A two level directory system

Here root directory is the first level directory . it consisting of entries of ' user directory. User1, user2,user3 are the user levels of directories. A,B,C are the files.

Hierarchical directory system

The two-level directory eliminates name conflicts among users but it is not satisfactory for users with a large number of files. To avoid this creates the subdirectory and load the same type of files in to the subdirectory. So in this method each can have as many directories are needed. Consider the figure 10.8 for better understanding.

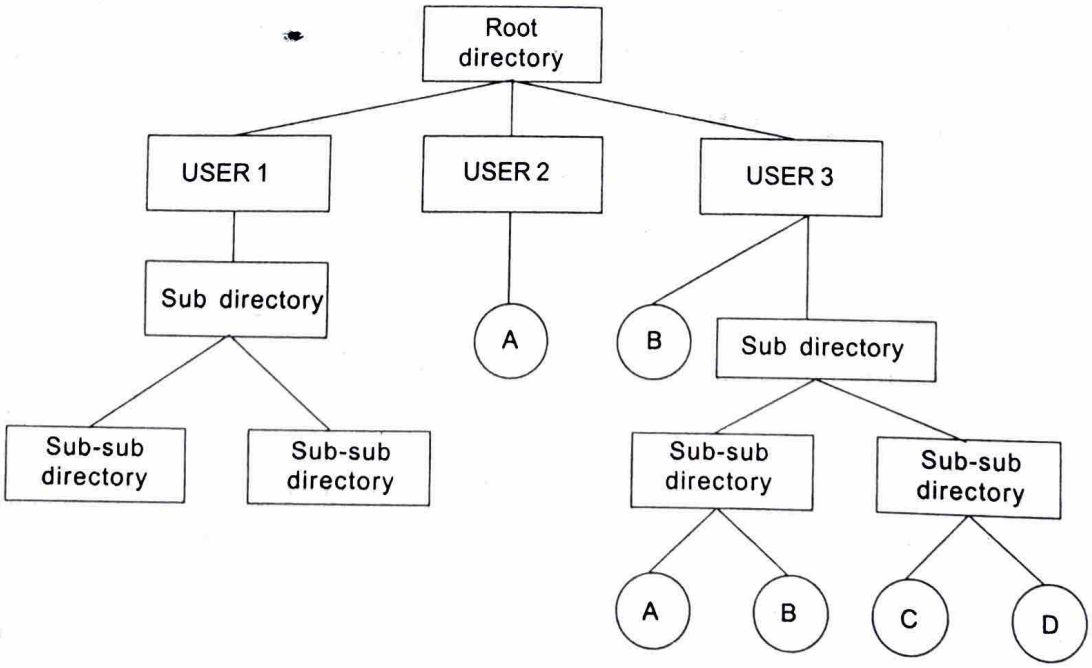


Figure 10.8 A hierarchical directory system

This directory structure looks like tree, that's why it is also said to be tree-level directory structure'.

General graph directory structure

When we add links to an existing tree structured directory , the tree structure is destroyed, resulting in a simple graph structure. Consider the figure 10.9 for better understanding. The primary advantage of this structure is traversing is easy and file sharing also possible.

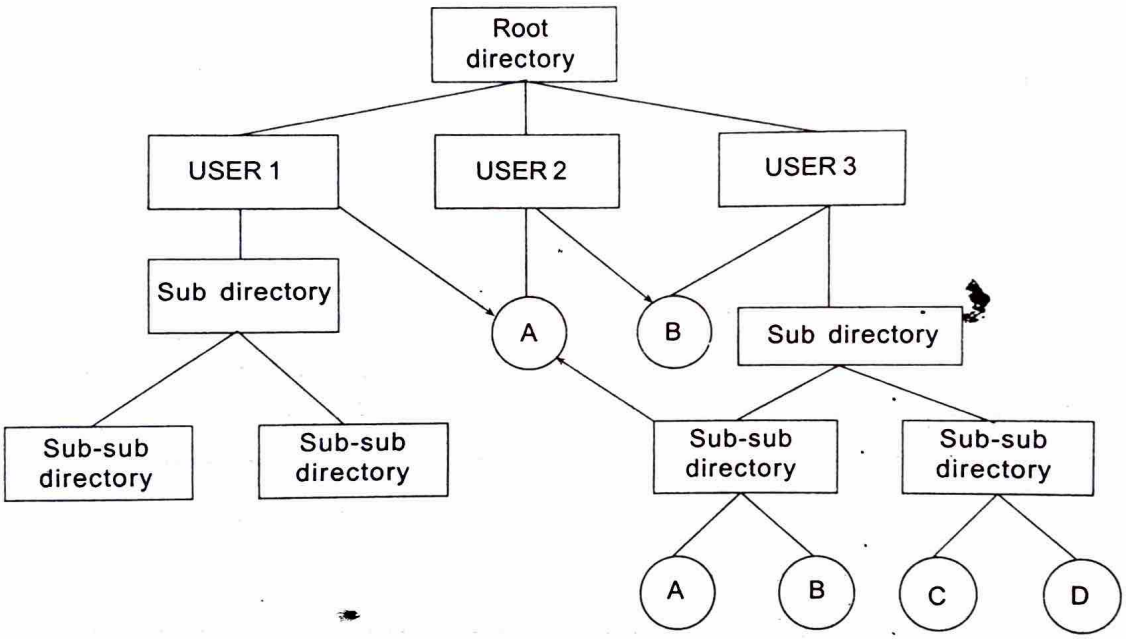


Figure 10.9 Graph directory structure.

10.5 File allocation methods

Files are normally stored on the disks, so the main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly. Three major methods of allocating disk space are in wide use: contiguous, linked and indexed. Each method has its advantages and disadvantages.

10.5.1 Contiguous allocation

In this allocation method each file occupies a set of contiguous blocks on the disk. For example a disk consisting of 1kb blocks. A100 kb file would be allocated 100 consecutive blocks. With 2kb blocks, it would be allocated 50 consecutive blocks. Consider the figure 10.10 for better understanding.

File allocation table

File name	Start name	length
A	21	5
B	3	10
C	35	7

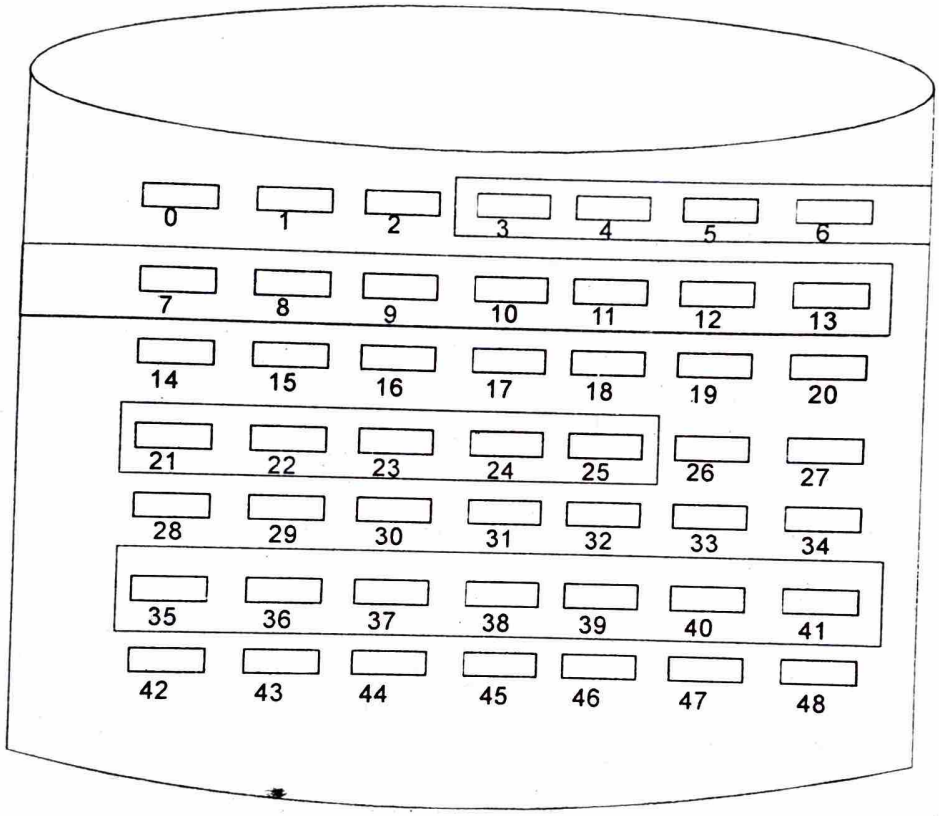


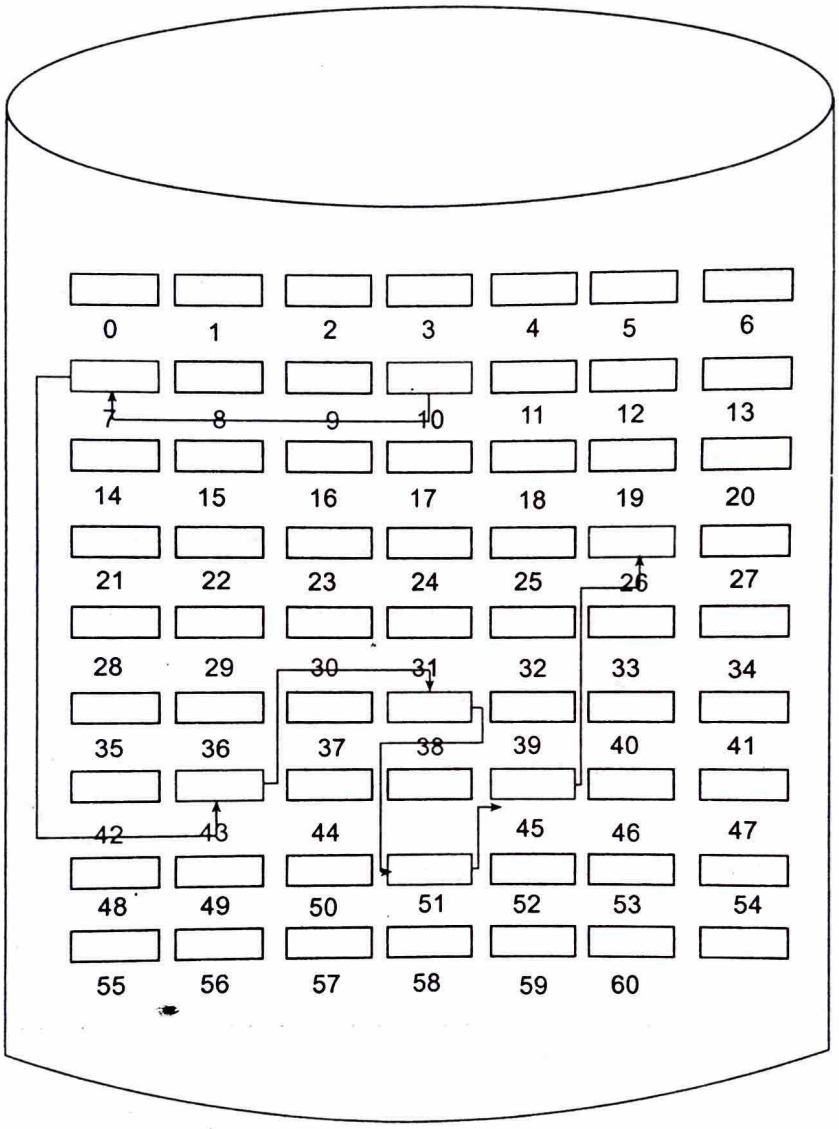
Figure 10.10 : Contigous File allocation

In the figure the right hand side part is the file allocating table' it consisting of a single entry for each file. It shows the file names starting block of the file and size of the file. For example file B size is 10 blocks., the file B stored from 3 to 13 blocks continuously. This method is best suited for sequential files. The main problem in this is it is difficult to find the contiguous free blocks in the disk. Another problem is external fragmentation, it means some free blocks could happen between two files.

10.5.2 Linked allocation

We can avoid the external fragmentation in this scheme. And also it is easy to locate the files , because allocation is on an individual block basis. Each block contains a pointer to the next free block in the chain. Here also the file allocation table consisting of a single entry for each file. Using this method any free block can be added to a

chain very easily. There is a link between one block to another block, that's why it is said to be linked allocation. Consider the figure 10.11 for better understand.



File allocation table

File name	Start block	length
Bubble sort	10	7

Figure 10.11 Linked allocation

Advantages:

1. Avoid the external fragmentation and compaction.
2. Suited for sequential files.

Disadvantages:

1. The pointer itself occupies some memory with in the block.
2. It takes much accessing time.

10.5.3 Grouped allocation (or) indexed allocation

We can overcome the problems using this method which were faced in the previous methods. In this method the file allocation table contains a single entry for each file. The entry consisting of one index block, the index block having the pointers to the other blocks. Which were occupied by the particular file. Consider the figure 10.12 for better understanding.

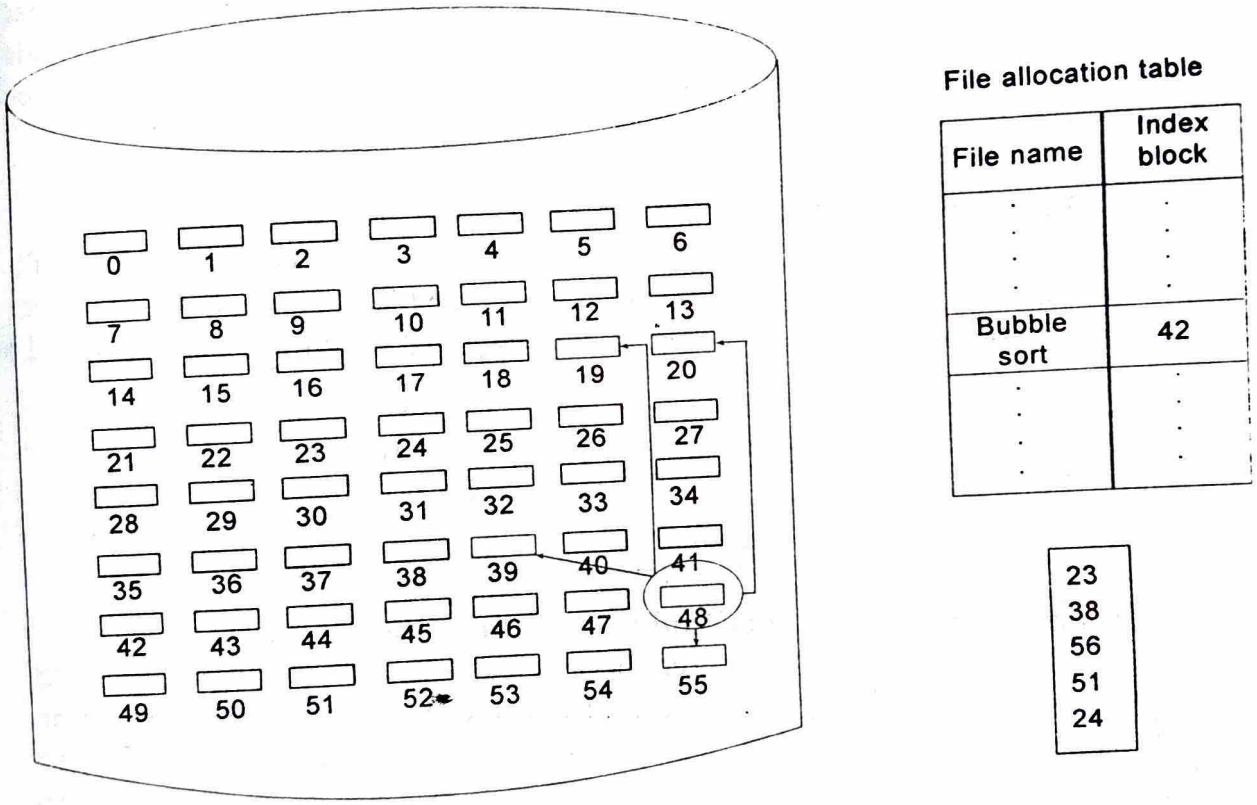


Figure 10.12 : Index allocation

For example bubble sort .c is a file, the entry for the file in the file allocation table points to be indexed block. The index block consisting of pointers to the other blocks. Which were occupied by the particular file. In the figure 23,38,56,51,24 these are the blocks occupied by the bubble sort .c .

Advantages

1. Indexed allocation supports both sequential and direct access files, that's why it is the most popular method.
2. The file indexes are not physically stored as part of the file allocation table.

3. Whenever the file size increases , we can easily add some more blocks to the index.
4. No external fragmentation.

10.6 Free space management (or) disk space management

Generally the files are stored on disk, so management of disk space is a major problem to the designers. If we want to allocate the space for the files , we have to know what blocks on the disk available. Thus we need a disk allocation table in addition to file allocation table. To keep track of free disk space, the file system maintains a free space list. The free space list records all the disk blocks which are free.(That is not allocated some other files). To create a file , we search the free space list for the required amount of space and allocate it to the new file. This space then removed from the free space list.. when a file is deleted , its disk space is added to the free space list. We discuss here a number of techniques that have been implemented.

10.6.1 Bit vector or Bit table

A bit vector is a collection of bits, in which each block is represented by one bit. If the block is free, the bit is 0. If the block is allocated , the bit is 1.for example consider a disk where blocks 5,8,16,35,40,43,48,51 are free, the free space bit vector would be

```
1111011011111110111111111111111110111011101101110

    5  8       16                  35  40  43  48 51
```

10.6.2 Chain free points (or) Linked free space list

Another approach is to link all the free space blocks together, keeping a pointer to the first free block. This block contains a pointer to the next free disk block and so on. In our example we would keep a pointer to block 5, as the first free block 5would contain a pointer to block 8, which would point to block 16, which would block to point to 35 and so on. Consider the figure 10.13 for better understanding.

10.6.3 Index block list

The chained free portion is not very efficient since to traverse the list, we must read each block requiring substantial I/O time. A modification of this approach would store the address of n free blocks in the first free block. The n-1 of these are actually free. The last one is the disk address of another block containing the address of another ' n' free blocks. the main advantage of this method is that the address of large number of free blocks can be found quickly . consider the figure 10.14 for better understanding.

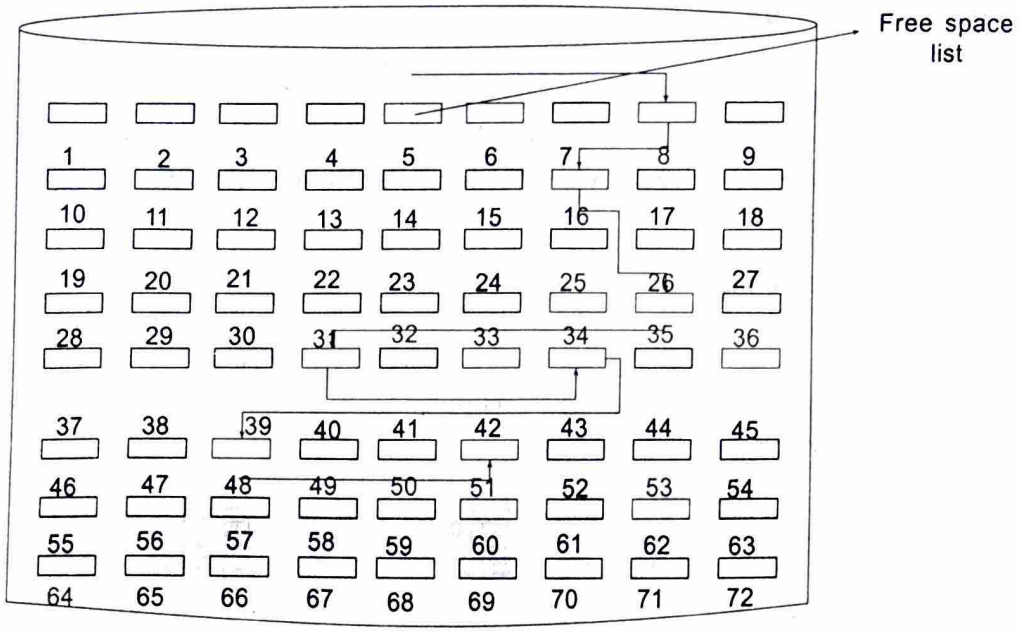


Figure 10.13 linked free space list on the disk

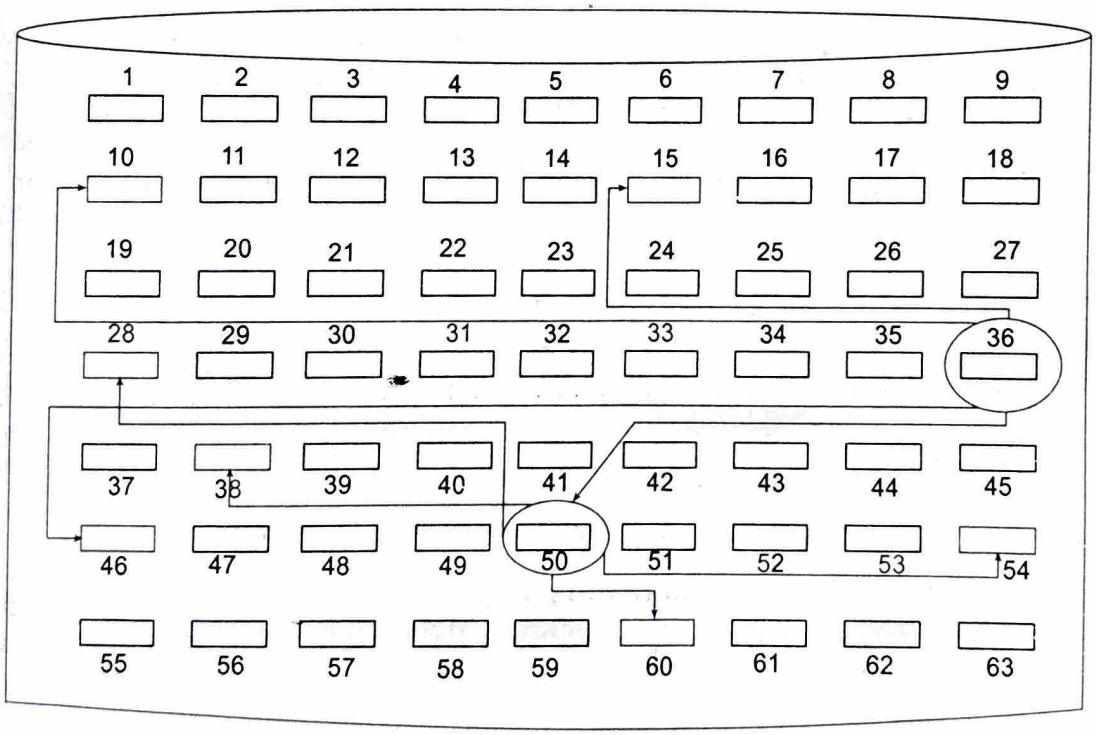


Figure 10.14 Index block list

10.7 Record blocking

Already we discussed that records are the logical unit of access of a file, where as blocks are unit of I/O with secondary storage. For I/O to be performed, records must be organized as blocks. Generally I/O transfer time is reduced by using larger blocks. Larger blocks require larger I/O buffers. Generally blocking can be done in one of the 3 methods.

1. **Fixed blocking**: In this, method, record lengths are fixed. The prescribed number of records are stored in a block. For example the block capacity is 100 records, but the number of records in the block is 90, then the wasted space (100-90 = 10 records area) is called internal fragmentation. Consider the below figure

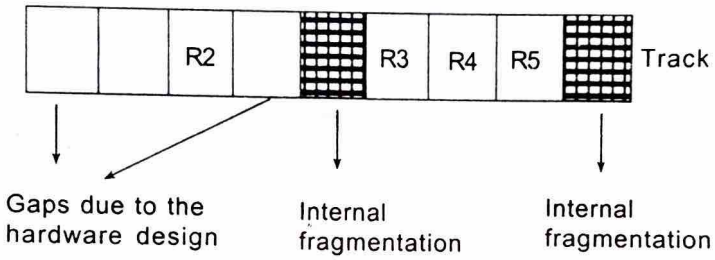


Figure 10.15 : Fixed blocking

2. **Variable length spanned blocking**: In this method record sizes are not same, variable length records are packed into blocks with no unused space. So some records may divide into two blocks, at this type of situation a pointer is passed from one block to another block consider the below figure.

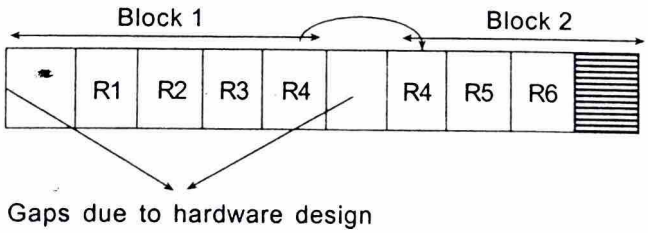


Figure 10.16 : Variable blocking : spanned

3. **Variable length un spanned blocking**: Here records are variable length, but the records are not spanned between blocks. In this method wasted area is serious problem, because of the inability to use the remainder of a block if the next record is larger than the remaining unused space. Consider the below figure

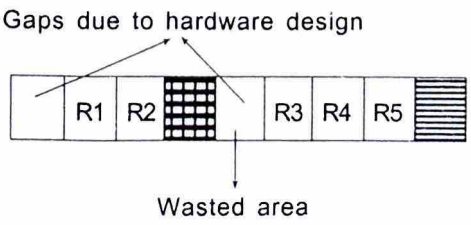


Figure 10.17 : Variable blocking : spanned

10.8 Unix File Management

Every file on a UNIX system has a unique 'inode'. The inode contains the information necessary for a process to access a file. Such as file ownership, access rights, file size, and location of the files data in the file system. Process access files by a well defined set of system calls and specify a file by a path name. An inode consists the following information.

- File owner identifier
- File type
- File access permissions
- File access times
- Number links to the files
- Table of contents for the disk addresses of data in a file.
- File size.

The UNIX KERNEL views all files as streams of bytes. UNIX supports 4 types of files.

I. Ordinary: An ordinary file created by users, these are application specific files.

II. Directory: A directory is a collection of files and sub directories, organized in hierarchical structure. A directory is a sequence of entries, each consisting of an inode number and name of the file.

iii. Special: Special files are used to access peripheral devices, such as terminals and printers. Each I/O device is associated with a special file.

iv. Name files: When user wants to create a file, the kernel must allocate disk blocks from the file system. The file system super block contains an array that is used to cache the number of free disk blocks in the file system. The utility program mkfs (make file system) organizes the data blocks of a file system in a linked list. So each link of the list is a disk block that contains an array of free disk block numbers, and one array entry is the number of the next block of the linked list.

For example consider the below figure.

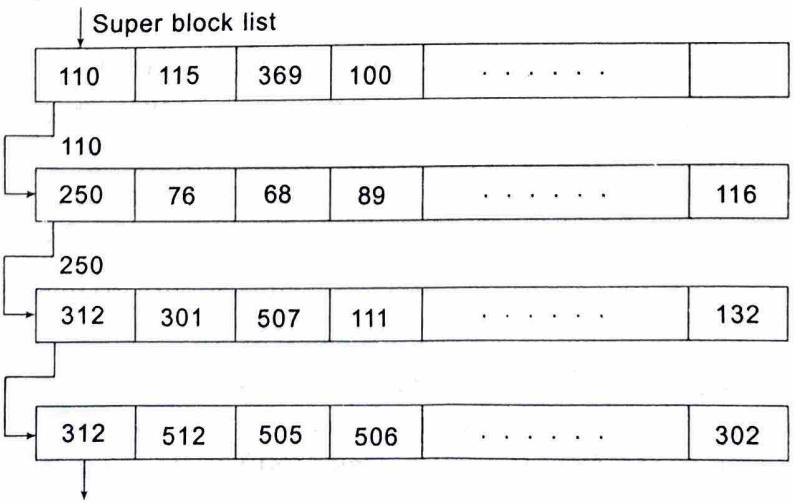


Figure 10.18 :Linked list of free disk block numbers

When the kernel wants to allocate a block from a file system, it allocates the next available block in the super block list. Once allocated, the block cannot be reallocated until it becomes free. Consider the following algorithms for better understand

```
Algorithm alloc /* file system block allocation */
Input : file system number
Output : buffer for new block
{
     while (super block locked)
        sleep (event super block not locked);
     Remove block from super block free list;
     if (removed last block from free list)
     {
        Lock super block;
        Read block just taken from free list (algorithm bread);
        Copy block numbers in block into super block;
        Release block buffer (algorithm brelse);
        Unlock super block;
        Wake up processes (event super block not locked);
     }
     Get buffer for block removed from super block list
        (algorithm getblk);
     zero buffer contents;
     Decrement total count of free blocks;
     Mark super block modified;
     Return buffer;
}
```

If a process writes a lot of data to a file, it repeatedly asks the system for blocks to store the data. But the kernel assigns only one block at a time

10.9 Windows 2000 File System

Windows 2000 supports several file systems, the most important of which are FAT – 16, FAT- 32, and NTFS (NT file system). FAT that runs on windows95, MS-DOS, and OS/2. FAT 16 is the old MS-DOS file system. It uses 16bit disk addresses, so it supports only 2 GB (2^{16}). FAT-32 uses 32 bit disk addresses and supports disk partitions up to 2TB (1TB = 1024 GB, 2^{32}=2TB)

NTFS is a new file system developed specifically for Windows NT and carried over to windows 2000. It uses 64-bit disk addresses and can support disk partitions up to 2^{64} bytes.

Individual filename in NTFS are limited to 255 characters, path names are limited to 32,767 characters. NTFS fully supports case sensitive names. An NTFS file is not just a linear sequence of bytes as FAT-32 and UNIX files are a file consists of multiple attributes.

Each NTFS disk partition contains files directories bitmaps, and other data structures. Each partition organized as a linear sequence of blocks.

10.9.1 Key features of NTFS

Recoverability: NTFS having high recoverability from the system crashes and disk failures.

Security: It provides high security among all the files

Large disks and large files: NTFS supports very large disks and very large files, so that it is more efficient than FAT.

Multiple data Streams: In NTFS a file is treated as a stream of bytes. It is possible to define multiple data streams for a single file.

General indexing facility: In NTFS files can be indexed by any attribute.

10.9.2 Disk storage concepts

Sector: The sector size in NTFS IS 512 bytes. A sector is the smallest physical Storage unit on the disk.

Cluster: collection of sectors is called cluster. The cluster size is power of 2.

Volume: Collection of clusters is called volume. The maximum volume size for NTFS is 2^{64} bytes.

Recoverability: The key elements that NTFS supports recoverability are.

- I/O Manager
- Log file Service
- Cache Manager
- Virtual Memory Manager.

Exercise:

1. What is a file? What are the operations of a file?
2. Explain different file allocation methods?
3. Explain different file accessing methods?
4. Explain free space management?
5. What is a file management system?
6. What are typical operations that may be performed on directory?
7. What is the relationship between a path name and working directory?
8. Explain the purpose of the 'open' and 'close' operations?