Unit 4

File System Mgmt.

- A file is a collection of related information defined by its creator. Commonly files represent program & data.
- A file is a sequence of bits & bytes, lines or records whose meaning is defined by its user.
- A file is a named collection of related info that is recorded on secondary storage.
- Data can't be written to sec. storage unless they are within a file.

File Attributes :-

- Name - It is only info that is kept in human readable form.
- Type :- Info about file which is reqd for system having diff^{n} types of files [.doc, .txt., .c, .cpp, .mpeg etc].
- Location :- It is a pointer to a device & to the location of the file on that device.
- Size :- Current size of file (in bytes, words)
- Protection :- Access control info determines who can perform various op^{n} on files
- Time, Date & User Identification :- Info contains when created, last modified & last usage.

This info about files is kept in the directory structure that resides on sec. storage.

Opⁿ on files

1. Creating a file :- space is needed
   An entry for the new file must be made.

2. Writing a file :- To write a file, a system call specifying both the name of the file & the info to be written to the file. The system must keep a write pointer to the location in file where the next write is to take place. The write pointer must be updated whenever a write occurs.

3. Reading a file :- To read from a file, a system call specifies the file name & when the next block of the file should be put in memory.

4. Deleting a file :- On this, we release all file space & invalidate the directory entry.

5. Truncating a file :- User may want to erase the contents of file but keep its attributes. only the file length change.

File Access Mechanism

1. Sequential Access
2. Direct "
3. Index "

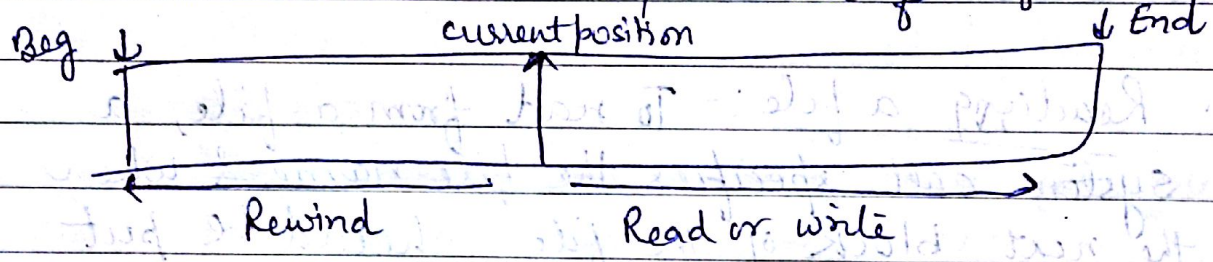
Sequential:- In this, one record is processed after the another. Eg Editors & compilers usually access files in this manner.

2 opn on files :- Read & write

✓ A read opn reads the next portion of the file & automatically advances a file pointer, which tracks the I/o location.

✓ A write opn appends the end of file & advances the end of newly written material.

This method is based on tape model of a file.

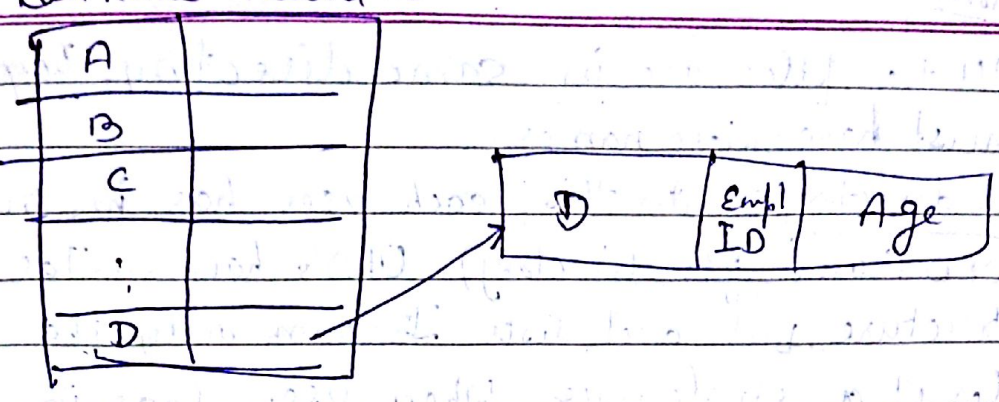


Direct :- Based on disk model, allows random access to any block or record of a file.

A file is viewed as a numbered sequence of blocks or records which are read or written in an arbitary manner.

It is well suited for DBMS in which we access large amount of info.

Indexed :- In this an index is created which contains a key field & pointers to various blocks. To find an entry in the file for a key value, we first search the index then use pointer to directly access a file & find the desried entry.



~~File Allocation Methods~~ :-

Logical Separation of files :-

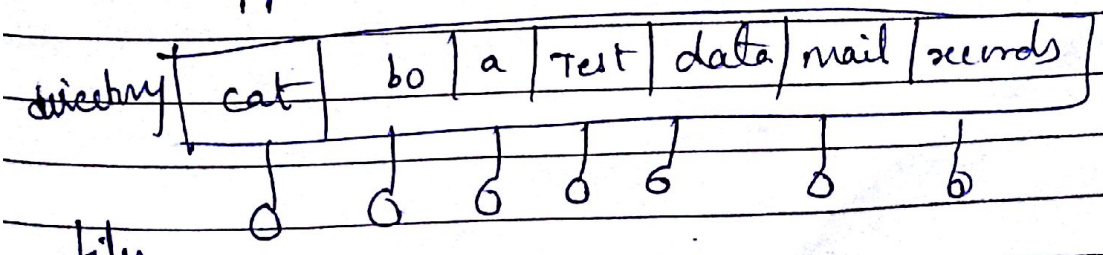
1) Partitioning
2) Directory Structure

Partitioning :- Logical division of secondary mem. & every partition is considered as independent drive (disk) i.e every drive can have its own file system & directory structure which is created by O.S

Directory str :- It is the data structure maintained by O.S to keep track of files.

File Directory (Directory structure)

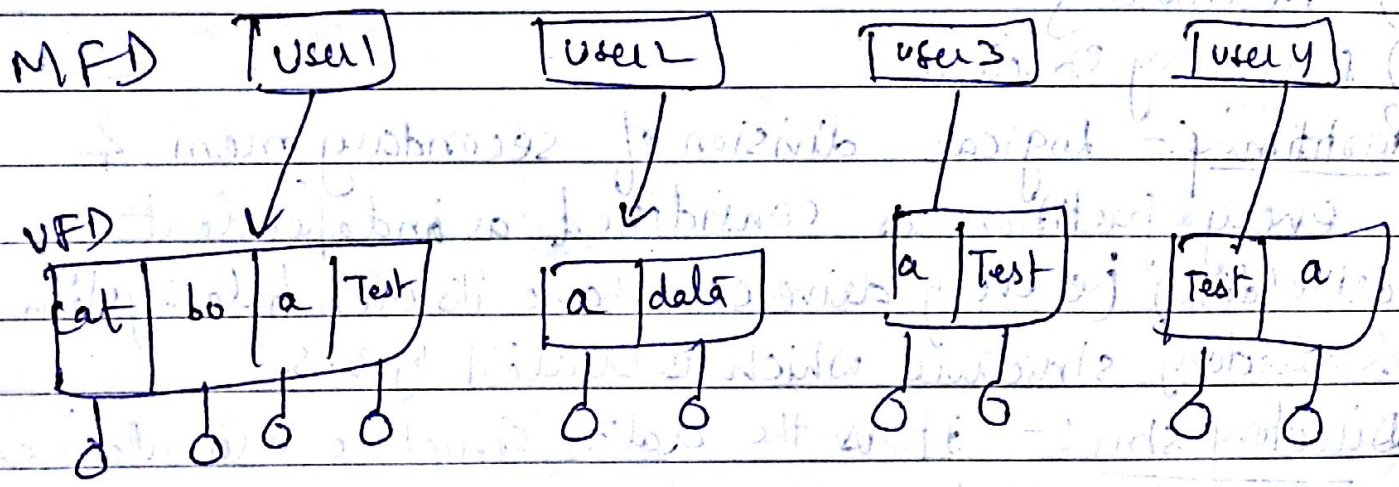
a) Single level Dir :- It is simplest, all files are contained in same directory which is easy to support & understand.



Limitations

1. All the files are in same directory, they must have unique names.

2. 2 level directory :- In this, each user has his own UFD (user file directory). UFD's have similar structure, but each lists ~~its own~~ only the files of a single user. When user logs in, the system's MFD (Master file directory) is searched. The MFD is indexed by user name or account no. & each entry points to the UFD for that user.



2 Level directory

3) Tree structured Directory :-

Implementation of file systems

(1) Implementation of directory structure
(2) Mapping of file in blocks. [for file Allocation Methods.)
(3) Free Space Management.

The key issues in implementing file storage is keeping track of which disk blocks go with which file.

a) Contiguous Allocation :- It is the simplest allocation scheme & we store each file as contiguous block of data on the disk. Thus, on a disk having block of size 1K, a 25K file would be allocated 25 consecutive blocks.

Logical separation of files:-

1. partitioning
2. Directory structure

1. Partitioning:- logical division of secondary memory & every partition is considered as independent drive (disk) (i.e. every drive can have its own file system & directory structure which is created by O.S.)

2. Directory Structure:-

- Directory structure is the data structure maintained by O.S. to keep track of files

File ⇒ A file is a named collection of data or collection of logically related entities (records)

- A file will have various attributes that may vary from one operating system to another but typically consists of these:-

(a) Name:- The symbolic file name is the only information kept in human readable form

(b) Size :- the amount of data stored in the file

(c) location :- the location of the file (in a storage device or in the system's logical file organization)

(d) Protection or Accessibility :- Access-control information determines who can do reading, writing, executing and so on.

(e) Type :- how the file data is used. For ex, an executable file contains machine instructions for a process. A data file may specify the applicatio that is used to access its data.

(f) Volatility :- the frequency with which additions and deletions are made to a file

(g) Activity :- the percentage of a file's records accessed during a given period of time

Operations on files :-

A file is a named collection of data that may be manipulated as a unit by operation such as

~~• Open :- Prepare a file to be referenced.~~
~~• Close :- Prevent further reference.~~

- Creating a file :- Create a new file
- 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.
- 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. (Copy data from a file to a process's memory)
- Repositioning within a file :- The directory is searched for the appropriate entry and the current-file-position pointer is repositioned to a given value.
- 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.
- Renaming a file :- change the name of the file.
- Truncating a file :- This function allows all attributes to remain unchanged - except for file length - but lets the file be reset to length zero and its file space released.

⇒

File structure :-

Four terms are in common use when discussing files :-

- Field :- A field is the basic element of data such as an employee's last name, a date etc
- Record :- A record is a collection of related fields that can be treated as a unit by some application program. For ex:- an employee record would contain such fields as name, SSN, Job classification, date of hire & so on.
- file :- A file is a collection of similar record
- database :- It is a collection of related data. The database itself consists of one or more types of files.

Access methods :-

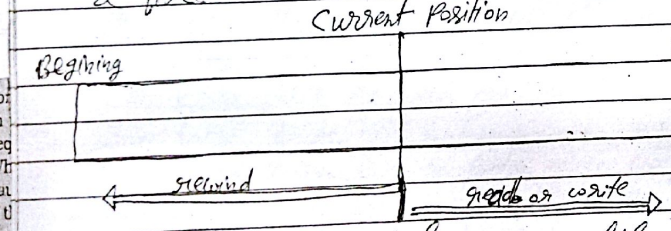
File store information when it is used, this information must be accessed and read into Computer memory. There are various access methods as following :-

(a) Sequential Access :-

Information in the file is processed in order, one record after the other. For ex:- editors and compilers usually access files in this fashion.

The main operations that we perform on a file are read and write. A read operation - read next - reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location. Similarly, the write operation - write next - appends to the end of the file and advances to the end of the newly written material.

Sequential access is based on Tape model of a file.



Sequential - access file

(b) Direct Access :-

A file is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order. For direct access, the file is viewed as a numbered sequence of blocks or records. Thus, we may read block 14, then read block 53 and then write block 7. There are no restrictions on the order of reading or writing for a direct-access file.

- Direct access mechanism is well suited for DBMS in which we have to access a large amount of information. When a query arrives then we compute the block containing the answer and then read that block directly which provides the desired information.

As a simple ex, on an airline - reservation system, we might store all the information about a particular flight (for ex. flight 713) in the block identified by the flight number. Thus the number of available seats for flight 713 is stored in block 713 of the reservation file. To store information about a larger set, such as people, we might compute a hash function on the people's names or search a small in-memory index to determine a block to read and search.

- The block numbers provided by the user to the O.S. is normally a relative block number.

In reality, systems may have zero or more file systems.

A relative block number is an index relative to the beginning of the file. Thus, the first relative block of the file is 0.

(ii) Indexed Access :- In this method an index is

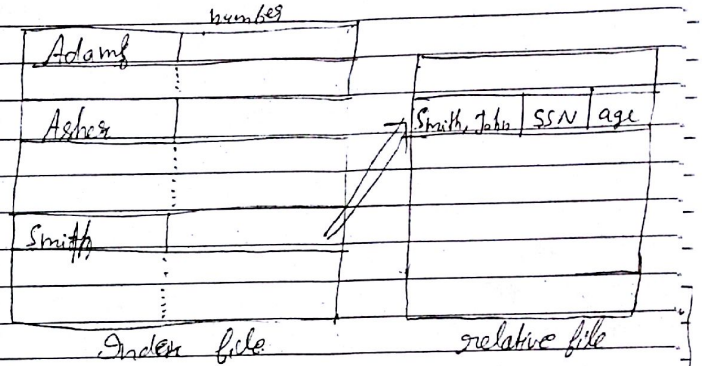
Simulation of Sequential Access on a Direct-Access File:-

Sequential Access	Implementation for direct Access
reset	cp = 0;
read next	read cp; cp = cp+1;
write next	write cp; cp = cp+1;

(iii) Indexed Access :-

In this method an index is created which contains a key field and pointers to the various blocks. To find an entry in the file for a key value, we first search the index and then use the pointer to directly access a file and find the desired entry.
With large files, the index file itself may become too large to be kept in memory. One solution is to create an index for the index file. The primary index file would contain pointers to the secondary index files, which would point to the actual data items.

Logical separation of drives :-
1. partitioning
2. Directory structure



File Directories (Directory Structure) :-

Some systems store millions of files on disk. To manage all these data, we need to organize them. This organization involve the use of directories.

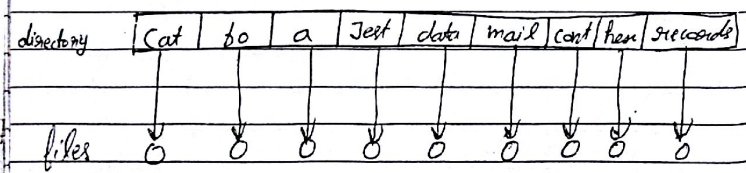
- A directory is a location for storing files on your computer.
The directories can be organized in many ways.
The various operations that can be performed on directories are:-
- Search for a file
- Create a file - New files need to be created and added to the directory.
- Delete a file.

- List a directory :- We need to be able to list the files in a directory and the contents of the directory entry for each file in the list.
- Rename a file
- Traverse the file System :- We may wish to access every directory and every file within directory structure.

In the following Section we describe the most common Schemes for defining the logical structure of a directory or logical view of directory structure.

1. Single - level Directory :-

The Simplest - level directory structure is the Single - level directory. All files are contained in the same directory, which is easy to support and understand.

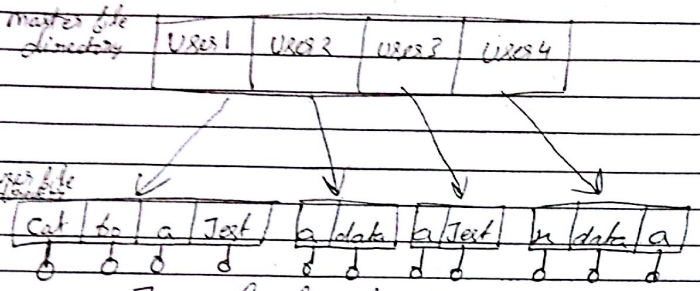


Single - level directory

- Single - level directory has certain 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. if Two users call their data file 'Jest', then unique - name rule is violated for ex:- in one programming class 23 students called the program for their second assignment prog-2, another 19 called it assgn. Although file names are generally selected to reflect the content of the file, they are often limited in length, complicating the task of making file names unique.

2. Two - Level Directory :-

In the Two - level directory structure, each user has his own user file directory (UFD). The UFDs have similar structures, but each lists only the files of a single user. when a user's job starts or 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. when a user refers to a particular file, only his own UFD is searched. Thus, different users may have files with the same name, as long as all the file names within each UFD are unique.



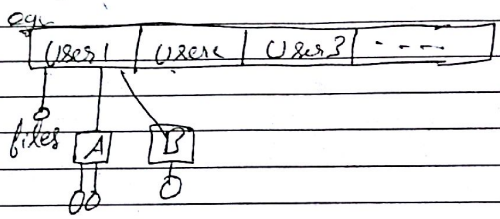
Two-level directory

Although the Two-level directory structure solves the name-collision problem, but it still has disadvantages. This structure effectively isolates one user from another. Isolation is an advantage when the users are completely independent but is a disadvantage when the users want to cooperate on some task and to access one another's files. Some systems simply do not allow local user files to be accessed by other users.

If access is to be permitted, one user must have the ability to name a file in another user's directory. To name a particular file uniquely in a Two-level directory, we must give both the user name and the file name. For ex- if user A wishes to access her own test file named test, they can simply refer to test. To access the file named test of user B (with directory-entry name user), however, they might have to refer to /user6/test.

3. Tree-structured Directories :-

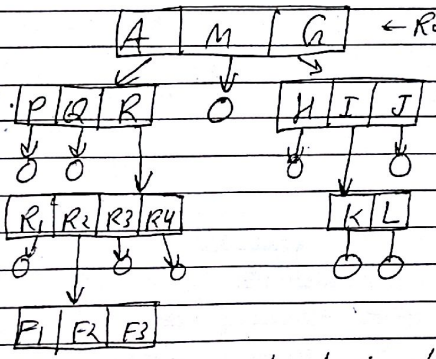
- So, here dir. within dir. is possible
- user can store files within at first level directory or can make directory in the first level



(II) Tree-structured Directories :-

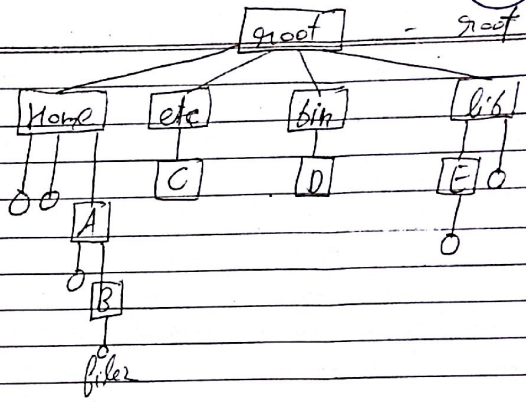
- This structure allows users to create their own subdirectories and to organize their files accordingly. The tree has a root directory, and every file in the system has a unique path name.
- A directory (or subdirectory) contains a set of files or subdirectories. One bit in each directory entry defines 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.
e.g. ex:- In the Tree-structured file system, if the current directory is root/spell/mail then the relative path name prt/first refers to the same file as does the absolute path name root/spell/mail/prt/first.



Tree-structured directory

or



- With a Tree-structured directory system, users can be allowed to access, in addition to their files, the files of other users. For ex, user B can access a file of user A by specifying its path names. User B can specify either an absolute or relative path name.

- In Tree dir structure, every file has single path from root directory.

(iv) Acyclic-Graph Directories:-

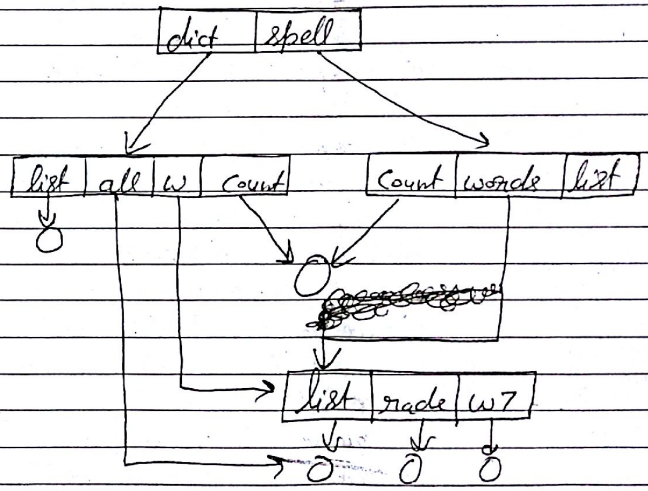
- Consider two programmers who are working on a joint project. The files associated with that project can be stored in a subdirectory, separating them from other projects and files of the two programmers. But since both programmers are equally responsible for the project, both want the subdirectory to be in their own directories. The common subdirectory

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

- 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. The same file or subdirectory may be in two different directories.

It is imp. to note that a shared file (or directory) is not the same as two copies of the file. With two copies, each programmer can view the copy, rather than the original, 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.

- Shared files and subdirectories can be implemented in several ways. A common way, is to create a new directory entry called a link. A link is effectively a pointer to another file or subdirectory. For ex:- a link may be implemented as an absolute or a relative path name. When a reference to a file is made, we search the directory. if the directory entry is marked as a link, then the name of the real file is included in the link information. We resolve that the link by using that path name to locate the real file. Links are easily identified by their format in the directory entry and are effectively named indirect pointers. The OS ignores these links when traversing directory trees to preserve acyclic structure of the system.



Acyclic-graph directory structure

A file may have more than one path but there is no cycle.

Implementation of File Systems:-

(i) Implementation of directory structure.
(ii) Mapping of file in blocks (or File Allocation Methods)
(iii) Free space Management.

File organization :-

- File organization refers to the manner in which the records of a file are arranged on secondary storage. Several file organization schemes have been implemented

- **Sequential** :- Records are placed in physical order, the "next" record is the one that physically follows the previous record. This organization is natural for files stored on magnetic tape, an inherently sequential medium. Disk files may also be sequentially organized, ~~but for various reasons~~

- **Direct** - Records are directly (randomly) accessed by their physical addresses on a direct access storage device (DASD). The application user places the records on DASD in any ~~other~~ order appropriate for a particular application

- **Indexed Sequential**:- Records on disk are arranged in logical sequence according to a key contained in each record. The system maintains an index containing the physical address of certain principal records. Indexed Sequential records may be accessed sequentially in key order or they may be accessed directly, by a search through the system-created index

- **Partitioned** - This is essentially a file of sequential subfiles. Each Sequential subfile is called a member. The starting address of each member is stored in the file's directory. Partitioned files have been used to store program libraries or macro libraries.

File Allocation Methods :-

An allocation method refers to how disk blocks are allocated for files:

An important function of the file system is to manage space on the secondary storage, which includes keeping track of both disk blocks allocated to files and free blocks available for allocation.

The main problems in allocating space to files are:
1. effective utilization of disk space
2. fast accessing of files

But secondary storage has some problems
1. slows disk access time and
2. large number of blocks to deal with.

There are three major methods of allocating disk space are in wide use: contiguous linked and indexed.

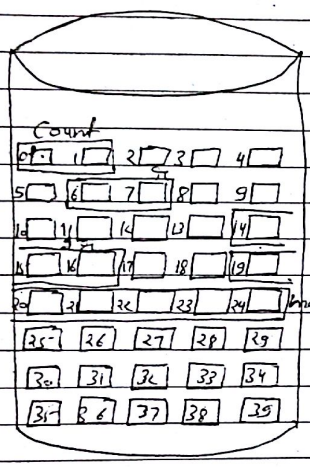
While discussing these allocation strategies, we will consider that a file is a sequence of blocks

(i) Contiguous Allocation :-

- Contiguous allocation requires that each file occupy a set of contiguous blocks on the disk. Disk addresses define a linear ordering on the disk: with this ordering, assuming that only one job is accessing the disk, accessing block b+1 after block b normally requires no head movement.
When head movement is needed (from the last sector of one cylinder to the first sector of the next cylinder), the head need only move from one track to the next. Thus, the number of disk seeks required for accessing contiguously allocated file is minimal, as is seek time when a seek is finally needed

- Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block. If the file is n blocks long and starts at location b, then it occupies blocks b, b+1, b+2, .., b+n-1. The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file.
- For sequential access, the file system remembers the disk address of the last block referenced and when necessary, reads the next block. For direct access to block i of a file that starts

at block 6, we can immediately access block 6+i. Thus, both sequential and direct access can be supported by contiguous allocation.



Directory

File	Start	length
Count	0	2
qr	6	2
tr	14	3
mail	19	6

The contiguous disk-space-allocation problem is how to satisfy a request of size n from a list of free holes. First-fit & Best-fit are the most common strategies used to select a free hole from the set of available holes.

These algorithms suffer from the problem of external fragmentation. As files are allocated and deleted, the free disk space is broken into little pieces. External fragmentation exists whenever free space is broken into chunks. It becomes a problem when the largest contiguous chunk is insufficient for a request; storage is fragmented into a number of holes, no one of which is large enough to store the data. Depending on the total amount of disk storage and the avg. file size, external fragmentation may be a minor or a major problem.

eg:- Block size = 512 Bytes
if File 1 — 1024 B — {0,1} blocks
File 2 — 770 B — {2,3}

Internal fragmentation is there, as in above block 3 is partially empty.
External fragmentation is there,
as if file1 — {0,1}, file2 — {2,3} file3 — {4,5,6,7,8}
file4 - {9,10}, file5 - {11,12,13,14,15,16}
& Now OS deletes file2 & file4 ∴ blocks 2,3,& 9,10 are free & there is now file6 req 4 blocks then those 4 free blocks can not be allocated to file6.

Drawback of contiguous allocation is that dynamic expansion of file is not possible

[last block allocated to a file contains the null pointer]

(ii) Linked Allocation :→

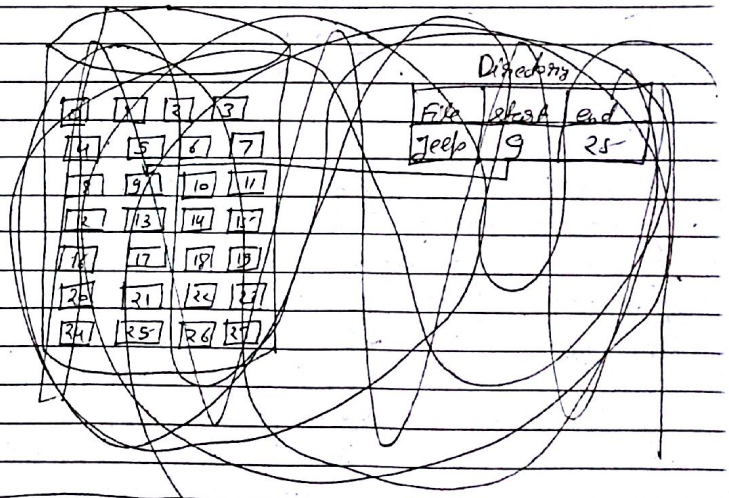
- Linked Allocation solves all problems of contiguous allocation. In this allocation, each file is a linked list of disk blocks which may be scattered at any place on the disk. The directory contains a pointer to the first block of file. For eg- a file of 5 blocks may start at block 9, continue at block 16 then block 1, block 10 and finally at block 13. Each block contains a pointer to the next block. ~~These pointers~~

- There is no external fragmentation with linked allocation. Any free block on the free space list can be used to satisfy a request since all blocks are linked together. Also, there is no need to declare the size of a file when it is created. A file can continue to grow as long as there are free blocks. Consequently, it is never necessary to compact the disk space.

- The major problem with linked allocation is that it can be used only for sequential access files. To find the i^{th} block of a file, we must start at the begining of that file, and follow the pointers until we get the i^{th} block. It is inefficient to support direct access capability for linked allocation files.

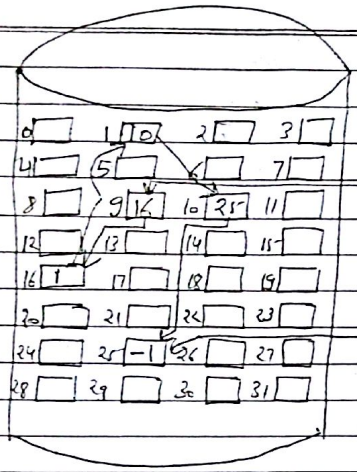
- Another problem of linked association is reliability, since the files are linked together by pointers scattered all over the disk. Consider what will happen if a pointer is lost or damaged.

- Another disadvantage is the space required for pointers. If a pointer requires 4 bytes out of a 512 byte block, then 0.78 percent of the disk is being used for pointers, rather than for information.



- The usual solution to this prob. is to collect blocks into multiples called clusters and to allocate clusters rather than blocks. But it increase ~~as~~ internal fragmentation bec. more space is wasted when a cluster is partially full than when a block is partially full.

[Direct Access]

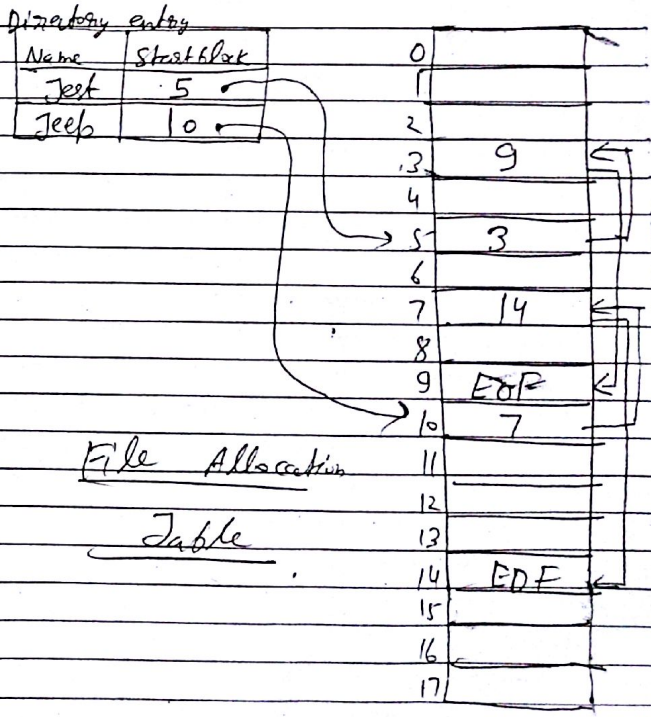


Directory		
File	start	end
Jeep	9	25

Linked allocation of disk space

FAT (File Allocation Table) :-

- A Table that the operating system uses to locate files on a disk.
  An implementation on linked allocation is FAT
- A section of disk at the beginning of each volume is set aside to contain the Table. The Table has one entry for each disk block and is indexed by block number. The directory entry contains the block number of the first block of the file. The Table entry indexed by that block number contains the block number of the next block in the file. This chain continues until the last block, which has a special end-of-file value as the Table entry. Unused blocks are indicated by a 0 Table value. Allocating a new block to a file is a simple matter of finding the first 0-valued Table entry and replacing the previous end-of-file value with the address of the new block. The 0 is then replaced with the end-of-file value.



File Allocation Table

FAT

No. of Entries in File Allocation Table

= No. of blocks in Sec. Memory partition

Size of FAT = No. of Entries × Size of one Entry

No. of Entries = No. of blocks

Size of Entry = No. of bits required to identify a block.

Indexed Allocation :-

→ Linked Allocation solves the external-fragmentation and size-declaration problems of contiguous allocation. ~~How~~

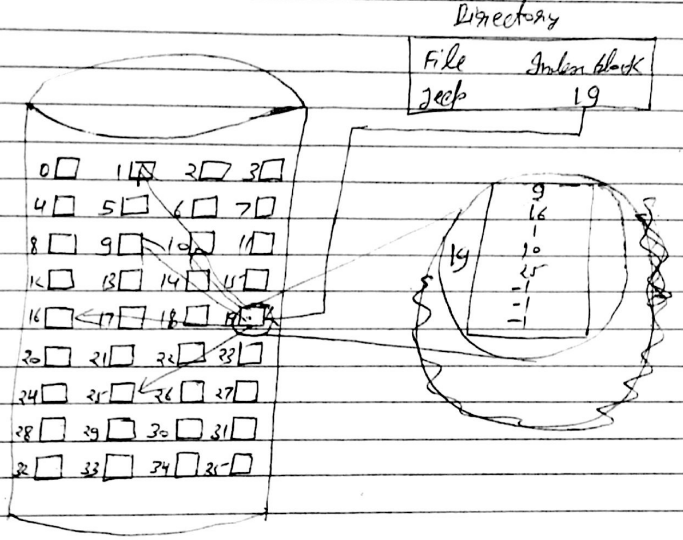
However, in the absence of a FAT, linked allocation cannot support efficient direct access, since the pointers to the block are scattered with the blocks themselves all over the disk and must be retrieved in order

- Indexed allocation solves the problem of linked allocation by bringing all the pointers together into one location; the index block

- Each file has its own index block, which is an array of disk-block addresses. The i^{th} entry in the index block points to the i^{th} block of the file. The directory contains the address of the index block. To find and read the i^{th} block, we use the pointer in the i^{th} index-block entry.

- Index allocation supports direct access, without suffering from external fragmentation, since any free block on the disk can satisfy a request for more space. Indexed allocation does suffer from wasted space, however. The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation. For ~~every~~ consider a common case in which we have a file of only one or two blocks. With

linked allocation, we lose the space of only one pointer per block. With indexed allocation, an entire ~~block~~ index block must be allocated, even if only one or two pointers will be non-nil.



Indexed - Allocation

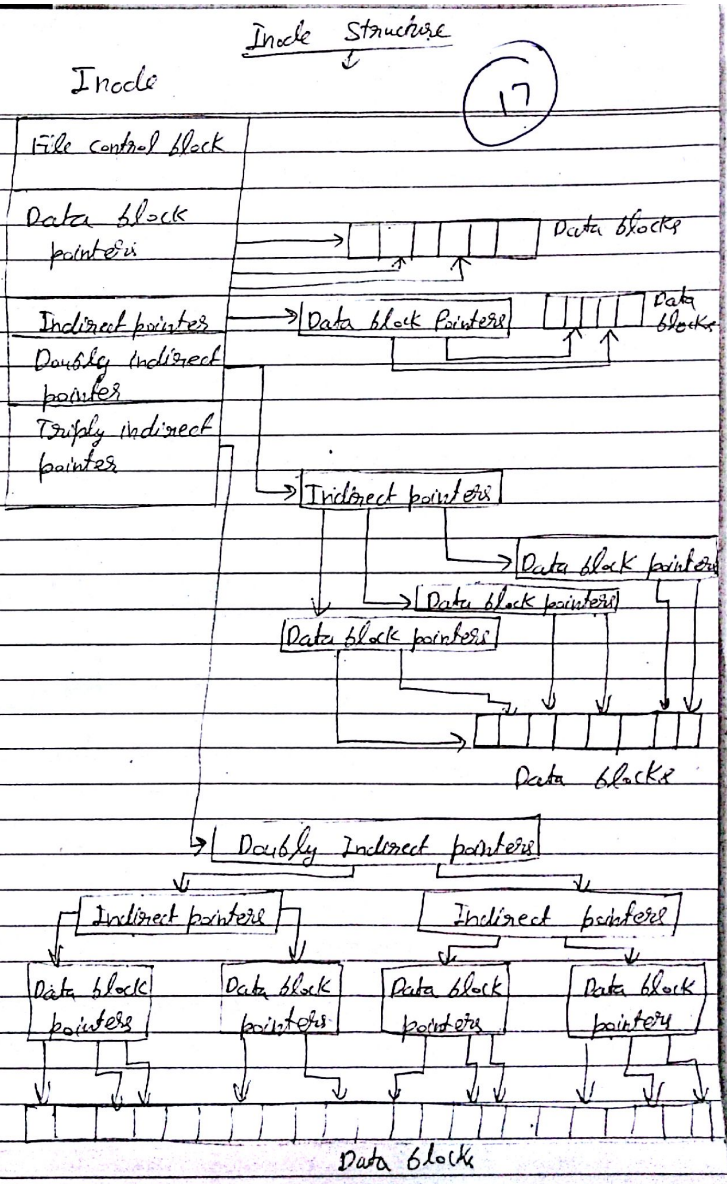
- This point raises the question of how large the index block should be. Mechanisms for this purpose are following:
  - Linked Scheme:- An index block is normally one disk block. Thus, it can be read and written directly by itself. To allow for files, we can link together several index blocks. For eg, an index block might contain a small header giving the name of the file and a set of the first 100 disk-block addresses. The next address (the last word in the index block) is nil (for a small file) or is a pointer to another index block (for a large file).
  - Multilevel Index. A variant of the linked representation is to use a first-level index block to point to a set of second-level index blocks, which in turn point to the file blocks.

⇒ I-Node:-

- Index blocks are called inodes (i.e. index nodes) in UNIX-base o.s..
  A file's inode stores the file's attributes, such as the its owner, size, time of creation and the time of last modification. It also stores the addresses of some of the file's data block and pointers to continuation index blocks called indirect blocks. Inode structures support upto three levels of indirect blocks. The first indirect block points to data block; these data blocks are singly indirect. The secondary indirect block contains pointers that reference only other

indirect blocks. These indirect blocks point to data blocks that are doubly indirect. The third indirect block points only to other indirect blocks that point only to more indirect blocks that point to data blocks; these data blocks are Triply indirect. The power of this hierarchical structure is that it places a relatively low limit on the maximum number of pointers that must be followed to locate file data - it enables inode to locate any data block by following at most four pointers (the inode and upto three levels of indirect blocks)

Inode Structure



FREE SPACE MANAGEMENT :→

- As files grow and shrink, file systems maintain a record of the location of blocks that are available to store new data (i.e free blocks). A file system maintains a free-list to keep track of free space. The free-space list records all free disk blocks - those not allocated to some file or directory. To create a new file, we search the free-space list for the required for the required amount of space and allocate that space to the new file.

- The free-space list may be implemented as following :-

(a) 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 ex,

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

0 0 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 0 0 0 1 1 1 1 0 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. Indeed, many computers supply bit-manipulation instructions that can be used effectively for that purpose.

(b) Linked List :→

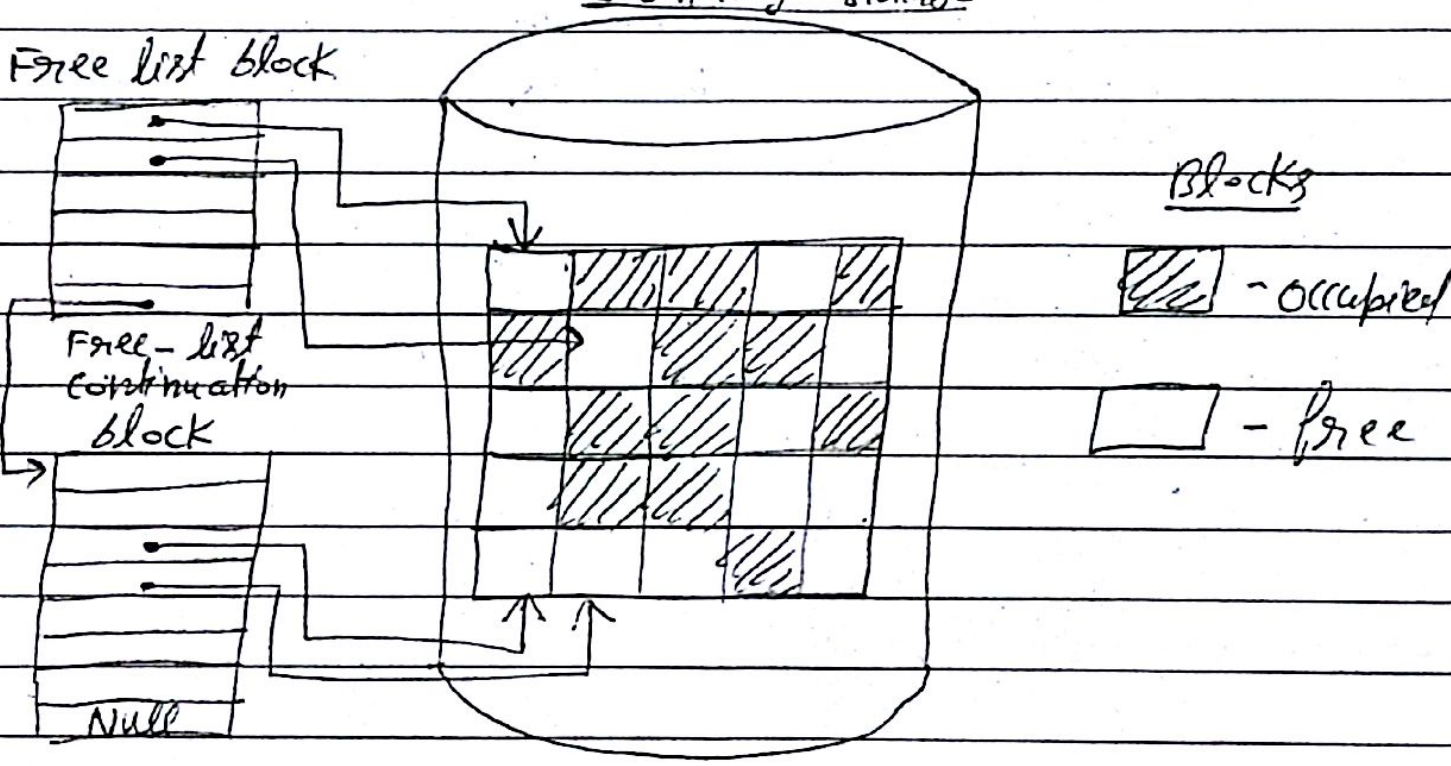
Here, we link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.

This first block contains a pointer to the next free disk block and so on. In our ex, 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. However, this scheme is not efficient; to traverse the list, we must read each block, which requires a substantial I/O time.

Linked free-space list on disk.

(C) 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 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.



Free-space management using a free list.