* File system Management. *

File concept :-

- A file is a named collection of related information that is recorded on secondary storage.
- From a user's perspective, a file is the smallest allotment of logical secondary storage; that is, data cannot be written to secondary storage unless they are within a file.
- Commonly, files represent programs (both source and object forms) and data.
  In general, a file is a sequence of bits, bytes, lines or records, the meaning of which is defined by the file's creator and user.
- The information in a file is defined by its creator.
- Many different types of information may be stored in a file - source programs, object programs, executable programs, numeric data, text, payroll records, graphic images, sound recordings and so on.
- A file has a certain defined structure according to its type:

e.g. A text file is a sequence of characters organized into lines (and possibly pages). A source file is a sequence of subroutines and functions, each of which is further organized as declarations followed by executable statements.

GOODLUCK Page No.
Date / /

Attributes :-

1. Name :- The symbolic file name is the only infⁿ kept in human readable form.
2. Identifier :- This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
3. Type :- This information is needed for those systems that support different types.
4. Location :- This information is a pointer to a device and to the location of the file on that device.
5. Size :- The current size of the file (in ~~big~~ bytes, words, or blocks), and possibly the maximum allowed size are included in this attribute.
6. Protection :- Access-control information determines who can do reading, ~~writing~~ writing, execution and so on.
7. Time, date and user identification :- This information may be kept for creation, last modification, and last user. These data can be useful for protection, security and usage monitoring.

GOODLUCK Page No.
Date

* File operations –

1. Creating a file :-
Two steps are neccessary to create a file.
First, space in the file system must be found for the file.
Second, an entry for the new file must be made in the directory.
The directory entry records the name of the file and the location in the file system, and possibly other information.

2. Writing a file :-
To write a file, we make a system call specifying both name of the file and the information to be written to the file.
Given the name of the file, the system searches the directory to find the location of the file.
The system must keep a write pointer to the location in the 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, 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.
Again, the directory is reached for the associated directory entry and the system ~~need~~ 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.
A given process is usually only reading or writing a given file, and the current operation location is kept as a per-process current-file-position pointer.
Both the read and write operations use this same pointer, saving space and reducing the system complexity.

4. Repositioning within a file :-
The directory is searched for the appropriate entry and the current-file-position is set to a given value.
Repositioning within a file does not need to involve any actual I/O.
This file operation is also known as a file seek.

5. Deleting a file :-
To delete a file, we search the directory for the named file.
Having found the associated directory entry, we release all ~~the~~ file space, so that it can be reused by other files, and erase the directory entry.

6 Truncating a file :-
The user may want to erase the contents of a file but keep its attributes.
Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged — ~~or~~ except for the length - but lets the file be reset to length zero and its file space released.

File type :-
It refers to the ability of the OS to differentiate various types of files like text files, binary and source files.

- A common technique for implementing file types is to include type as part of the file name.
- The name is split into two parts - a name and an extension, usually separated by a period character.
- In this way the user and the OS can tell from the name alone what the type of a file is.
eg. only a file with a .com, .exe or .bat extension can be executed.
The .com and .exe files are two forms of binary executable files, whereas a .bat file is a batch file containing, in ASCII format

commands to the OS.

File Type	Usual extension	function.
executable.	exe, com, bin or none	read to run m/c language program.
Object	obj, O	compiled, m/c language, not linked.
source code	c, cc, java, pas, asm, a	Source code in various language.
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing.
archive	arc, zip, tar	related files grouped into one file, sometimes compress for archiving or storage.
multimedia	mpeg, mov, rm	binary file containing audio or A/V information.

* Access Methods :-

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

* Sequential Access :-

- The simplest access method is sequential access.
- Infn in the file is processed in order, one record after the other.
- This mode of access is by far the most common. eg. editors and compilers usually access files in this fashion.
- The bulk of the operations on a file is reads and writes.
- A read operation reads the next portion of the file and automatically advances a file pointer which tracks the I/O location.
- Similarly, a write appends to the end of the file and advances to the end of the newly writte material (the new end of file).

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

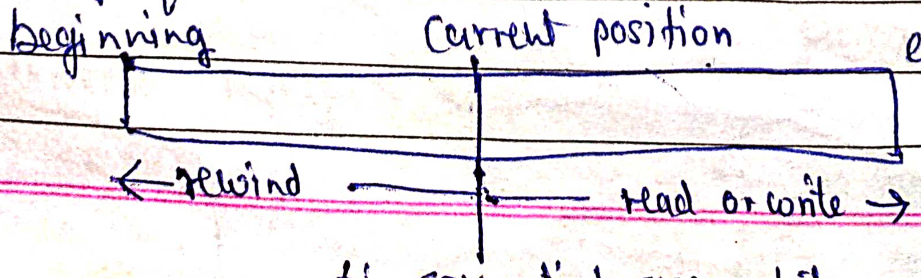


fig. sequential-access file.

GOODLUCK Page No.
Date / /

sequential accen is based on a tape model of a file, and works as well on sequential accen devices as it does on random accen ones.

* Direct Accen :-

- A file is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order.
- The direct accen method is based on a disk model of a file, since disks allow random accen to any file block.
- For direct accen, the file is viewed as a numbered sequence of blocks or records.
- A direct-accen file allows arbitrary blocks to be read or written.
- 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 writting for a direct-accen-file.
- Direct-accen files are of great use for immediate accen to large amounts of infⁿ.
- Database are often of this type.
- As a simple example, on an airline-reservation system, we might store all the infⁿ about a particular flight in the block identified by the flight number.

~~Allocation methods~~ :—

* File-system structure :—

The file system is generally composed of many different levels. The structure shown in figure is an example of a layered design.

Each ~~layer~~ level in the design uses the features of lower levels to create new features for use by higher levels.

$$\begin{tikzcd}
\text{application programs} \arrow[d, Rightarrow] \\
\text{logical file system} \arrow[d, Rightarrow] \\
\text{file-organization module} \arrow[d, Rightarrow] \\
\text{basic file system} \arrow[d, Rightarrow] \\
\text{I/O control} \arrow[d, Rightarrow] \\
\text{devices}
\end{tikzcd}$$

fig: layered file systems.

I/O control :—

The lowest level, the I/O control, consists of device drivers and interrupt handlers to transfer infⁿ betⁿ the main memory and the disk systems.

A device driver can be thought of as a translator.

Its i/p consists of high-level commands & its output consists of low level, h/w specific instrs that are used by the h/w controller, which interfaces the I/O device to the rest of the system.

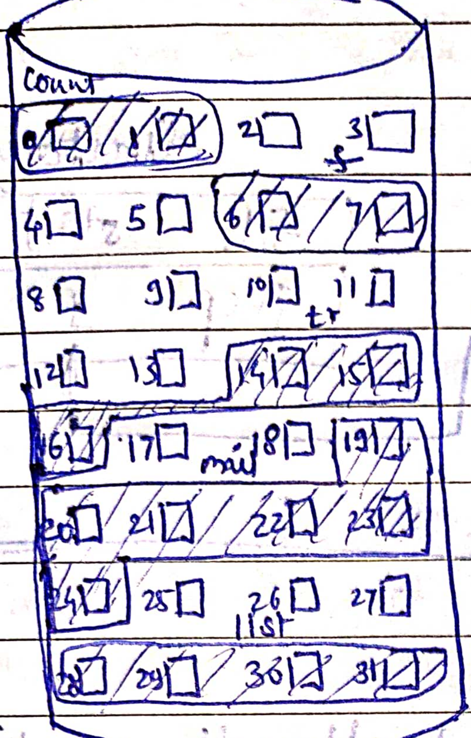
Basic file system –
It needs only to issue generic commands to the appropriate device driver to read and write physical blocks on the disk. Each physical block is identified by its numeric disk address (eg. drive 1, cylinder 73, track 2, sector 10).

File organization module :–
– It knows about files and their logical blocks, as well as physical blocks.
– By knowing the type of file allocation used and the location of the file, the file-organization module can translate logical block addresses to physical block addresses for the basic file system to transfer.

Logical file system –
– It manages metadata information.
– Metadata includes all of the file-system structure excluding the actual data (or content of file).
– The logical file system manages the directory structure to provide the file-organization

module with the information, the latter needs, given a symbolic file name.
- It maintains file structure via file control blocks.
- A file control block (FCB) contains infⁿ about the file, including ownership, permissions and location of the file contents.
- The logical file system is also responsible for protection and security.

Allocation Methods :-
1. Contiguous Allocation :-
- This method requires each file to occupy a set of contiguous blocks on the disks.
- Disk addresses define a linear ordering on the disk.
- 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.



directory

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

fig contiguous allocation of disk space.

- One difficulty is finding space for a new file.
- It also suffer from external fragmentation. As files are allocated and deleted, the free disk space is broken into little pieces.

2- Linked Allocation:—

- Linked allocation solves the all problems of contiguous allocation.
- With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk.
- The directory contains a pointer to the first and last blocks of the ~~disk~~ file.

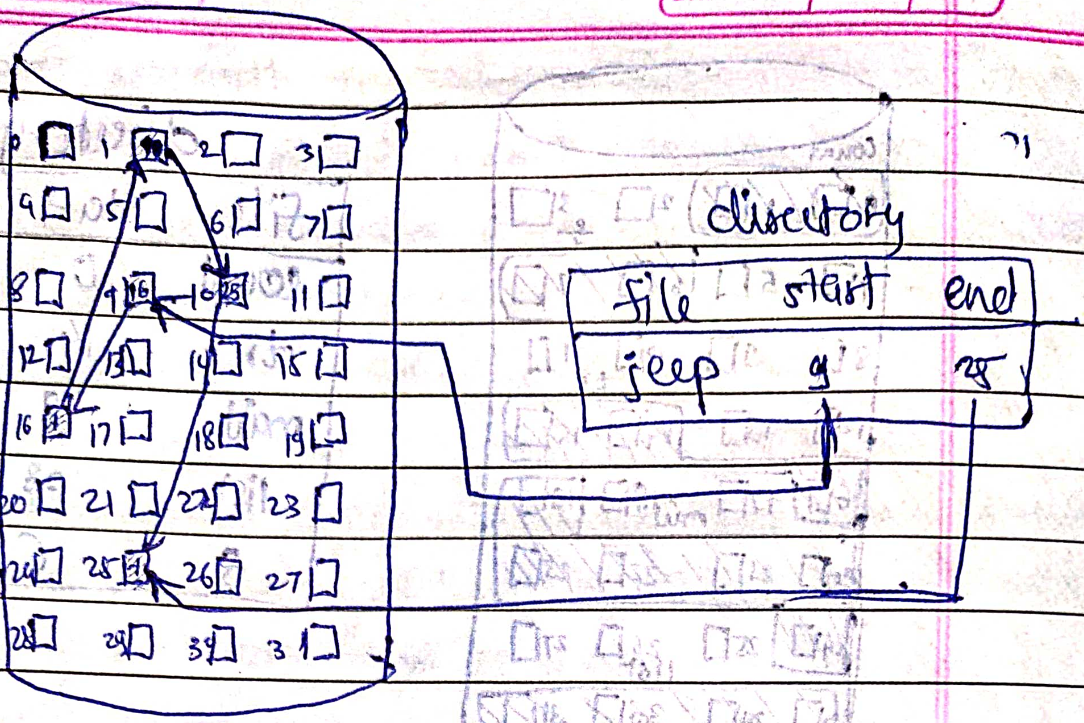


fig: linked allocation of disk space

eg: As shown in figure a file of five blocks might start at block 9, continue at block 16, then block 1, block 10, and finally block 25. Each block contains a pointer to the next block.

- To create a new file, we simply create a new entry in the directory.
- With linked allocation, each directory entry has a pointer to the first disk block of the file. This pointer is initialized to nil to signify an empty file.

- There is no external fragmentation with linked allocation, and any free block on the free-space list can be used to satisfy a request.
- The size of a file does not need to be declared when that file is created. A file can continue to grow as long as free blocks are available.

- Linked allocation does have ~~[illegible]~~ disadvantages, Because the file blocks are distributed randomly on the disks, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
- It does not support random or direct access. We can not directly access the blocks of a file.
- ~~Anoth~~ ~~In~~ linked ~~list~~ allocation suffer from the space required for the pointers. If a pointer requires 4 bytes out of a 512-bytes block, then 0.78 percent of the disk is being used for pointers, rather than for information.
- Another problem of linked allocation is reliability. Since the files are linked together by pointers scattered all over the disk, what would happen if a pointer were lost or damaged.

3. Indexed allocation :-
- In this method each file has its ~~ow~~ own index block, which is an array of disk-block addresses.
- The ith entry in the index block points to the ith block of the file.
- The directory contains the address of the index block.
- To read the ith block, we use the pointer in the ith index-block entry to find and read the desired block.

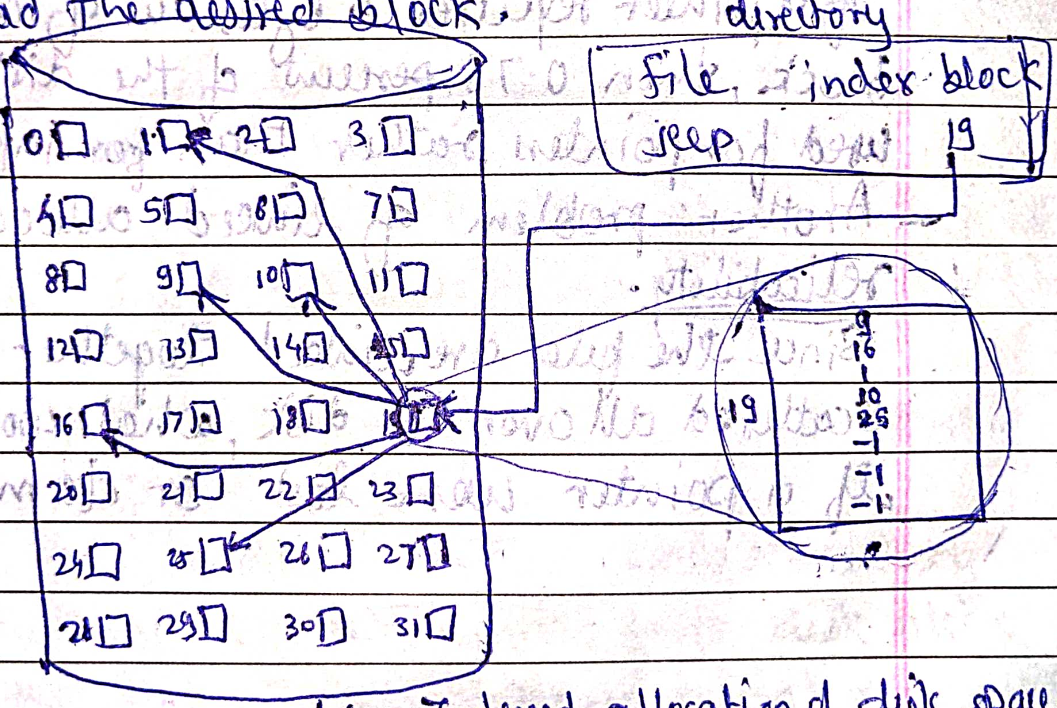


fig. Indexed allocation of disk space.

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

- Indexed allocation supports direct accen, without suffering from external fragmentation, because any free block on the disk may satisfy a request for more space.

- Indexed allocation does suffer from wasted space :
- The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.

- Indexed allocation supports direct accen to the blocks occupied by the file and therefore provides fast accen to the file blocks.
- It overcomes the problem of external fragmentation.

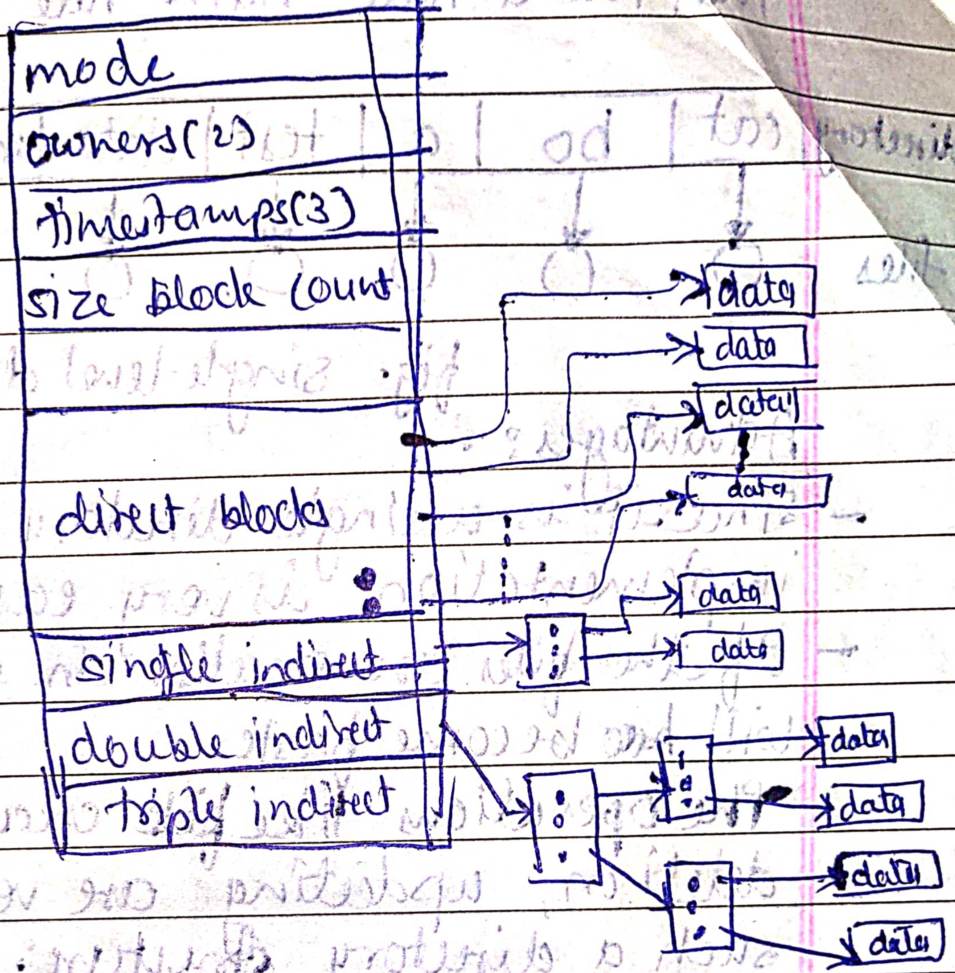
For files that are very large, single index block may not be able to hold all the pointers.
Following mechanisms can be used to resolve this :

1. Linked scheme :-
This scheme links two or more index blocks together for holding the pointers.
Every index block would then contain pointer or the addren to the next index block.

2. Multilevel index :-
In this policy, a first level index block is used to point to the second level index blocks which inturn points to the disk blocks occupied by the file. This can be extended to 3 or more level depending on the maximum file size.

3. Combined scheme :-
- In this scheme a special block called the Inode (Information Node) contains all the infⁿ about the file such as the name, size, authority etc and the remaining space of Inode is used to store the Disk Block addresses which contain the actual file.
- The first few of these pointers in Inode points to the direct blocks i.e. the pointers contain the addresses of the disk blocks that contain data of the file.
- The next few pointers points to indirect blocks. Indirect blocks may be single indirect, double indirect or triple indirect.
- Single indirect block is the disk block that does not contain the file data but the disk address of the blocks that contain the file data.
- Double indirect blocks do not contain the file data but the disk address of the blocks that contain the address of the blocks containing the file data.

The last pointer would contain the address of a triple indirect block.



Directory structure :-

1. Single-Level Directory :-

- The simplest directory structure is the single-level directory.
- All files are contained in the same directory which is easy to support and understand.
- A single-level directory has significant limitations, however, when the number of files increases or when the system has more than one user.
- Since all files are in the same directory, they

must have unique names.
- If two users call their data file test, then the unique-name rule is violated.

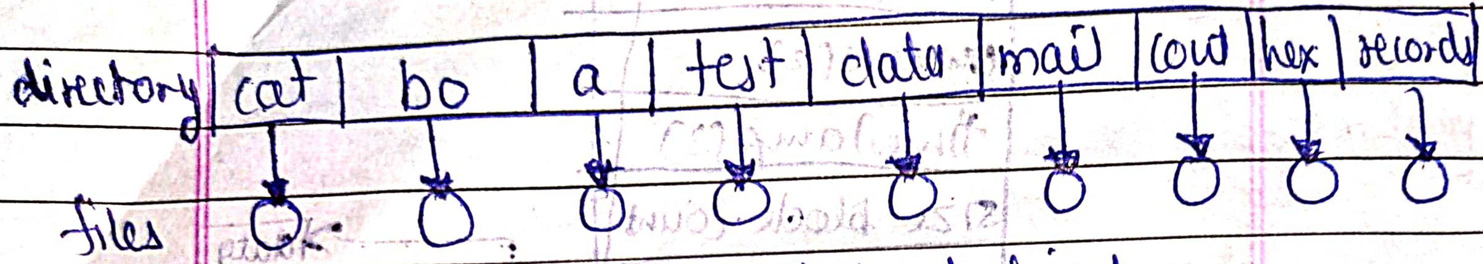


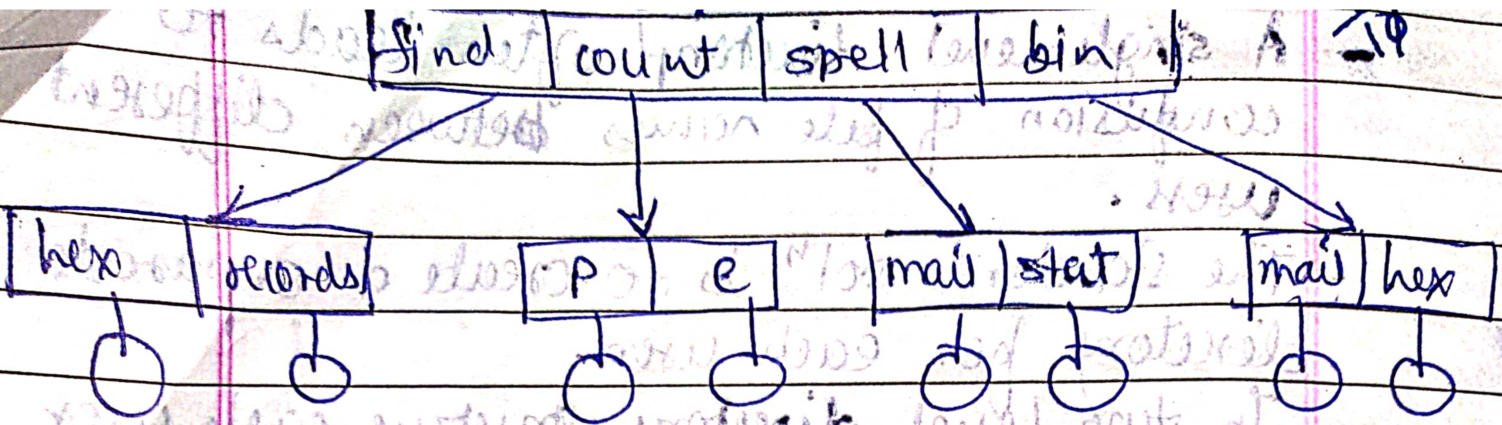
fig. single-level directory.

Advantages:-
- Since it is a single directory, so its implementation is very easy
- If the files are smaller in size searching will ~~bec~~ become faster.
- The operations like file creation, searching, deletion, updating are very easy in such a directory structure.

Disadvantages -
- There may chance of name ~~collects~~ collision because two files can not have the same name.
- Searching will became time taking if the directory is large.
- In this can not group the same type of files together.

2. Two-level directory :-

- A single-level directory often leads to confusion of file names between different users.
- The standard solⁿ is to create a separate directory for each user.
- In two-level directory structure each user has her own user file directory (UFD).
- Each UFD has a similar structure, but lists only the files of a single user.
- When a user 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 files names within each UFD are unique.

Advantages: →
- We can give full path like /username/directory-name/.
- Different users can have same directory as well as file name.
- Searching of files become more easy due to path name and user-grouping.

Disadvantages: —
- A user is not allowed to share files with other users.
- Still it not very scalable, two files of the same type cannot be grouped together in the same user.