File Access Methods in Operating System

- When a file is used, information is read and accessed into computer memory and there are several ways to access this information of the file.
- There are three ways to access a file into a computer system: ***Sequential-Access, Direct Access, Index sequential Method***.

1. Sequential Access

- Information in the file is processed in order, one record after the other.
- This mode of access is by far the most common; for example, the editor and compiler usually access the file in this fashion.
- Data is accessed one record right after another record in an order.
- When we use the read command, it moves ahead pointer by one.
- When we use the write command, it will allocate memory and move the pointer to the end of the file.

2. Direct Access

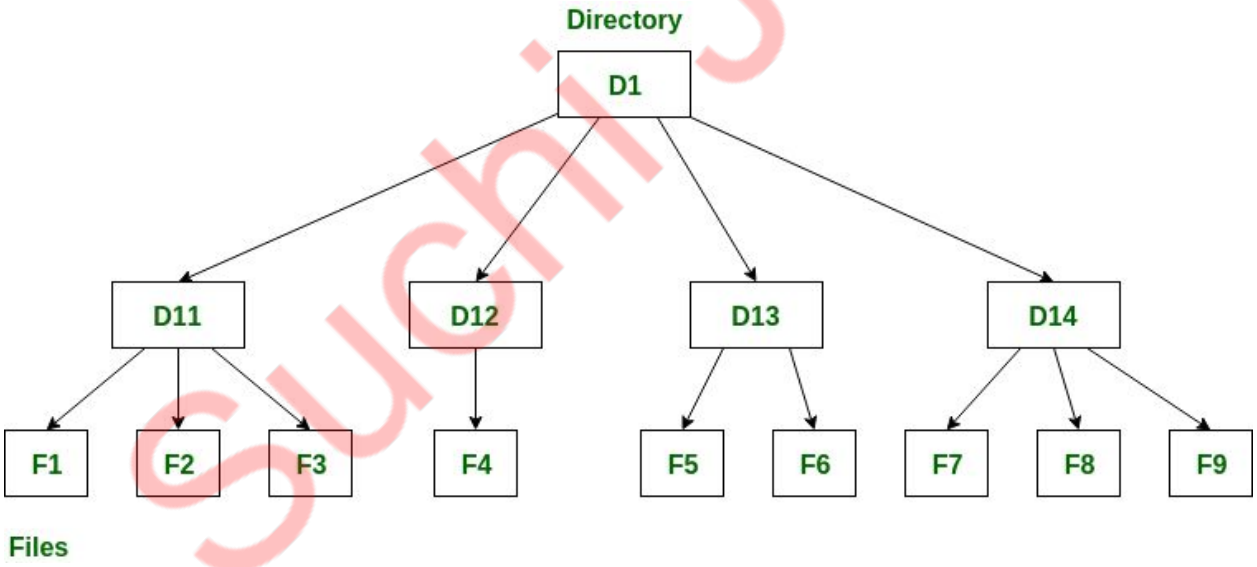
- Also known as relative access method.
- A fixed-length logical record that allows the program to read and write records rapidly in no particular order.
- The direct access is based on the disk model of a file since the disk allows random access to any file block.
- For direct access, the file is viewed as a numbered sequence of block or record.
- Thus, we may read block 14 then block 59 and then we can write block 17.
- There is no restriction on the order of reading and writing for a direct access file.
- A block number provided by the user to the operating system is normally a relative block number, the first relative block of the file is 0 and then 1 and so on.

3. **Index sequential method –**

- These methods construct an index for the file.
- The index, like an index in the back of a book, contains the pointer to the various blocks.
- To find a record in the file, we first search the index and then by the help of a pointer we access the file directly.

Directory Structure

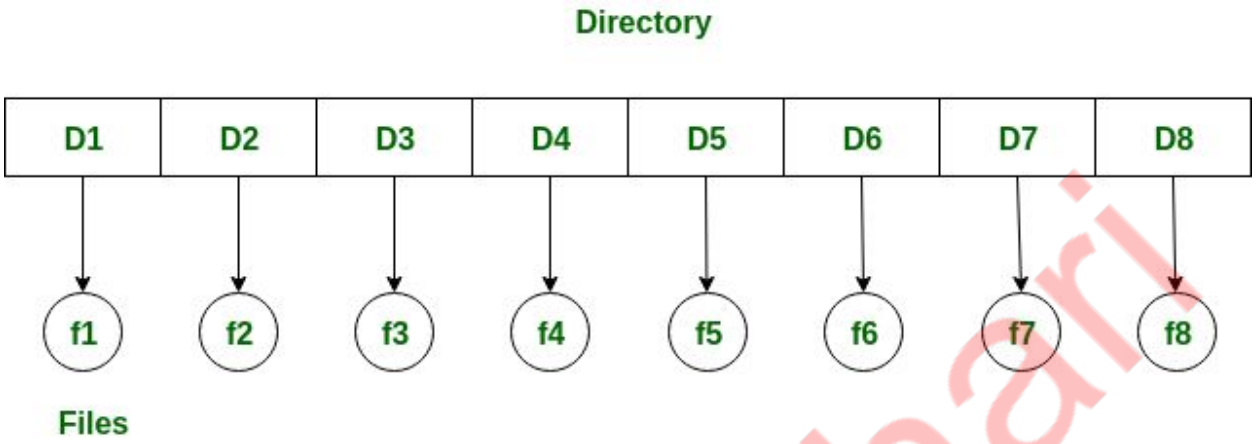
A **directory** is a container that is used to contain folders and files. It organizes files and folders into a hierarchical manner.



There are several logical structures of a directory:

1. **Single-level directory –**

- Single level directory is the simplest directory structure.
- In it all files are contained in the same directory which make it easy to support and understand.
- Since all the files are in the same directory, they must have the unique name.



Advantages:

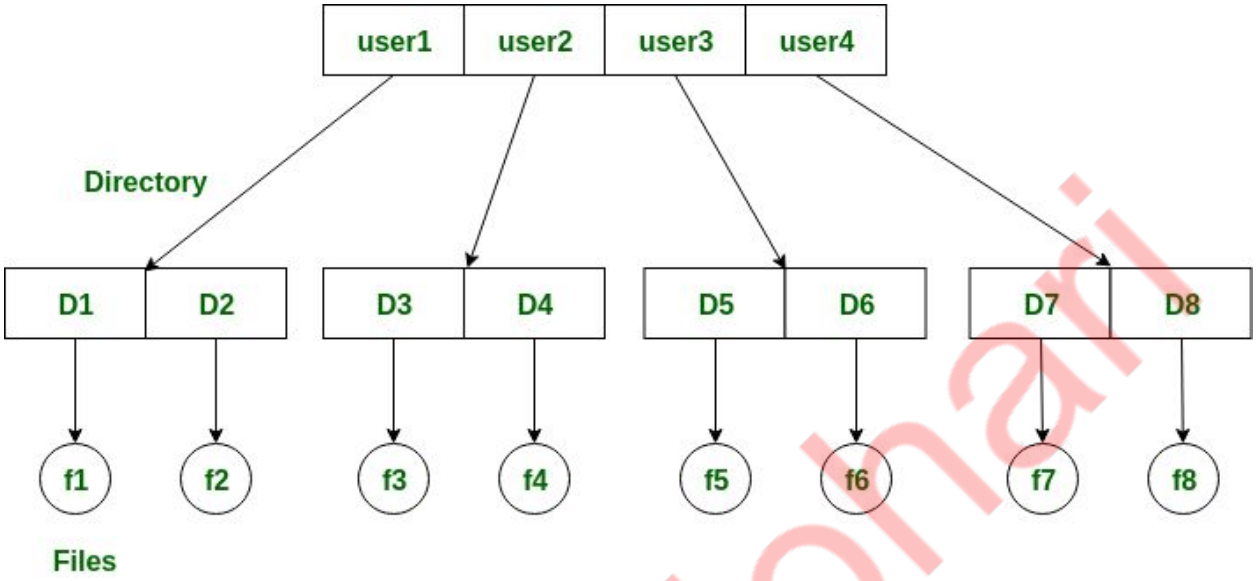
- Since it is a single directory, its implementation is very easy.
- Searching is faster.
- The operations like file creation, deletion, updating are very easy in such a directory structure.

Disadvantages:

- There may be a chance of name collision because two files can not have the same name.
- Searching will become time taking if the directory is large.
- This can not group the same type of files together.

2. **Two-level directory –**

- In the two-level directory structure, each user has there own *user files directory (UFD)*.
- The UFDs have similar structures, but each lists only the files of a single user.
- The system's *master file directory (MFD)* is searched whenever a new user is logged in. The MFD is indexed by username or account number, and each entry points to the UFD for that user.



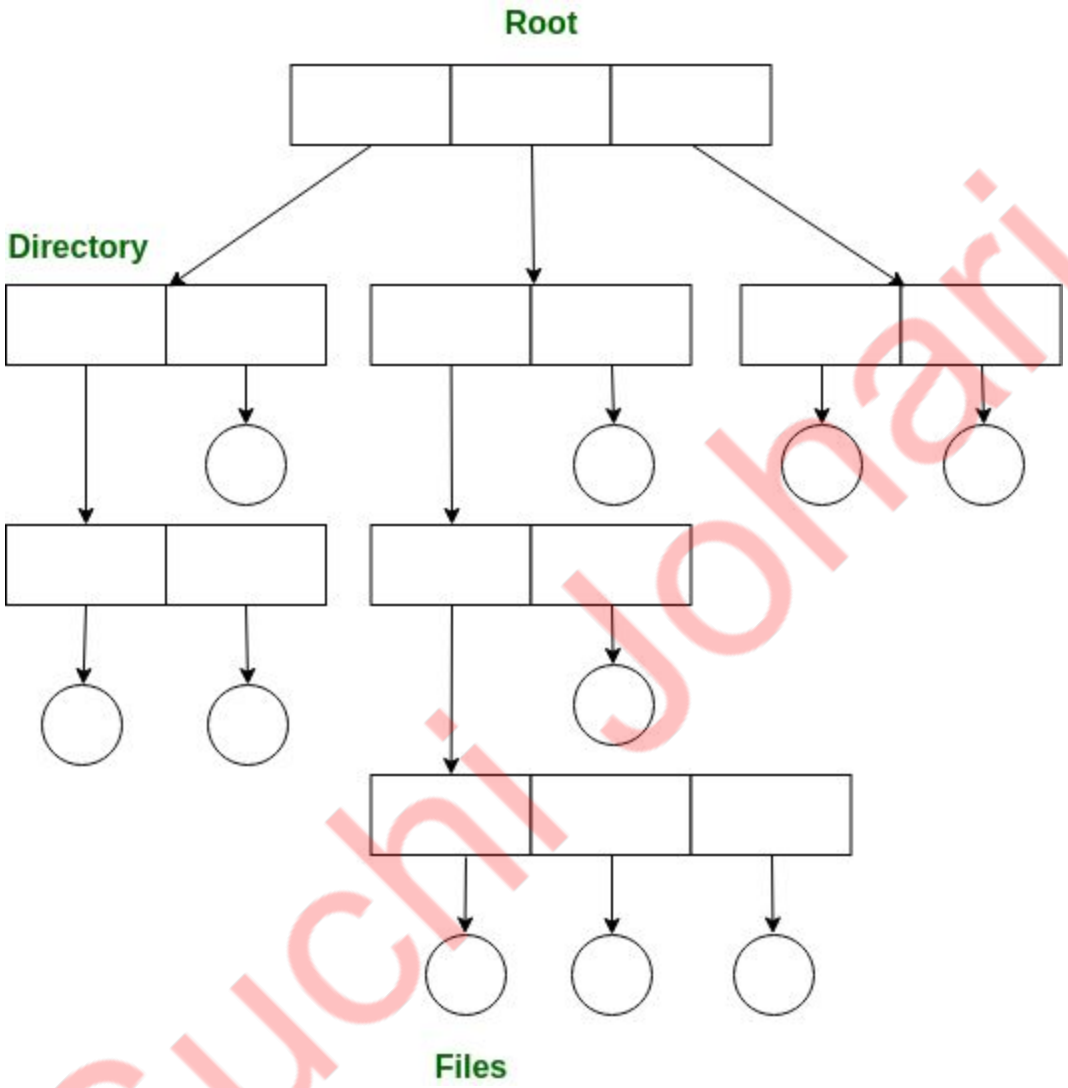
Advantages:

- We can give a full path like /User-name/directory-name/.
- Different users can have the same directory as well as file name.
- Searching for files becomes more easy due to path name and user-grouping.

Disadvantages:

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

3. **Tree-structured directory –**
   - A tree structure is the most common directory structure. The tree has a root directory, and every file in the system has a unique path.



Advantages:

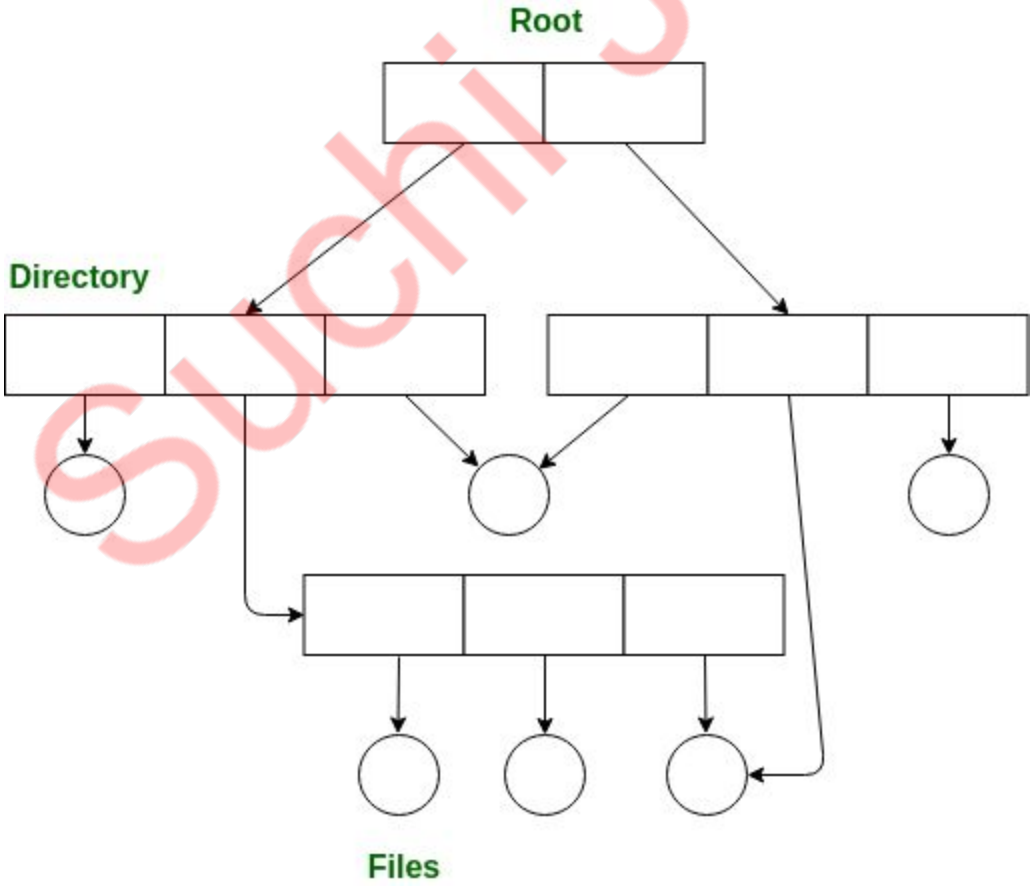
- Very generalize, since full path names can be given.
- Very scalable, the probability of name collision is less.
- Searching becomes very easy.

Disadvantages:

- Every file does not fit into the hierarchical model, files may be saved into multiple directories.
- We can not share files.
- It is inefficient, because accessing a file may go under multiple directories.

4. **Acyclic graph directory –**

- An acyclic graph is a graph with no cycle and allows to share subdirectories and files.
- The same file or subdirectories may be in two different directories.
- It is a natural generalization of the tree-structured directory.
- It is used in situations like when two programmers are working on a joint project and they need to access files. The associated files are stored in a subdirectory, separating them from other projects and files of other programmers, since they are working on a joint project so they want the subdirectories to be into their own directories. The common subdirectories should be shared. So here we use Acyclic directories.
- It is the point to note that a shared file is not the same as a copy file . If any programmer makes some changes in the subdirectory it will reflect in both subdirectories.



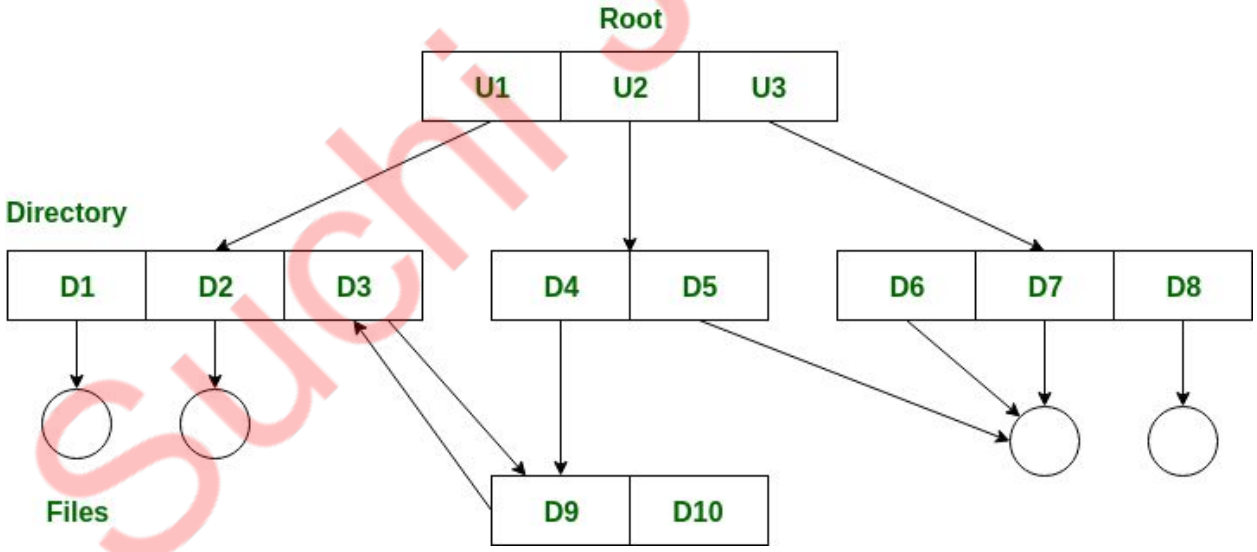
Advantages:

- We can share files.
- Searching is easy due to different-different paths.

2. **Disadvantages:**
    - We share the files via linking, in case of deleting it may create the problem,
    - If the link is softlink then after deleting the file we left with a dangling pointer.
    - In the case of hardlink, to delete a file we have to delete all the references associated with it.

5. General graph directory structure –

- In general graph directory structure, cycles are allowed within a directory structure where multiple directories can be derived from more than one parent directory.
- The main problem with this kind of directory structure is to calculate total size or space that has been taken by the files and directories.



Advantages:

- It allows cycles.
- It is more flexible than other directories structure.

3. **Disadvantages:**
    - It is more costly than others.
    - It needs garbage collection.

File Allocation Methods

The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

The main idea behind these methods is to provide:

- Efficient disk space utilization.
- Fast access to the file blocks.

A. Contiguous Allocation

- In this scheme, each file occupies a contiguous set of blocks on the disk.
- For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: *b, b+1, b+2,……b+n-1*.
- Starting block address and the length of the file is given.

The directory entry for a file with contiguous allocation contains

- Address of starting block
- Length of the allocated portion.

Directory

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

Advantages:

- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

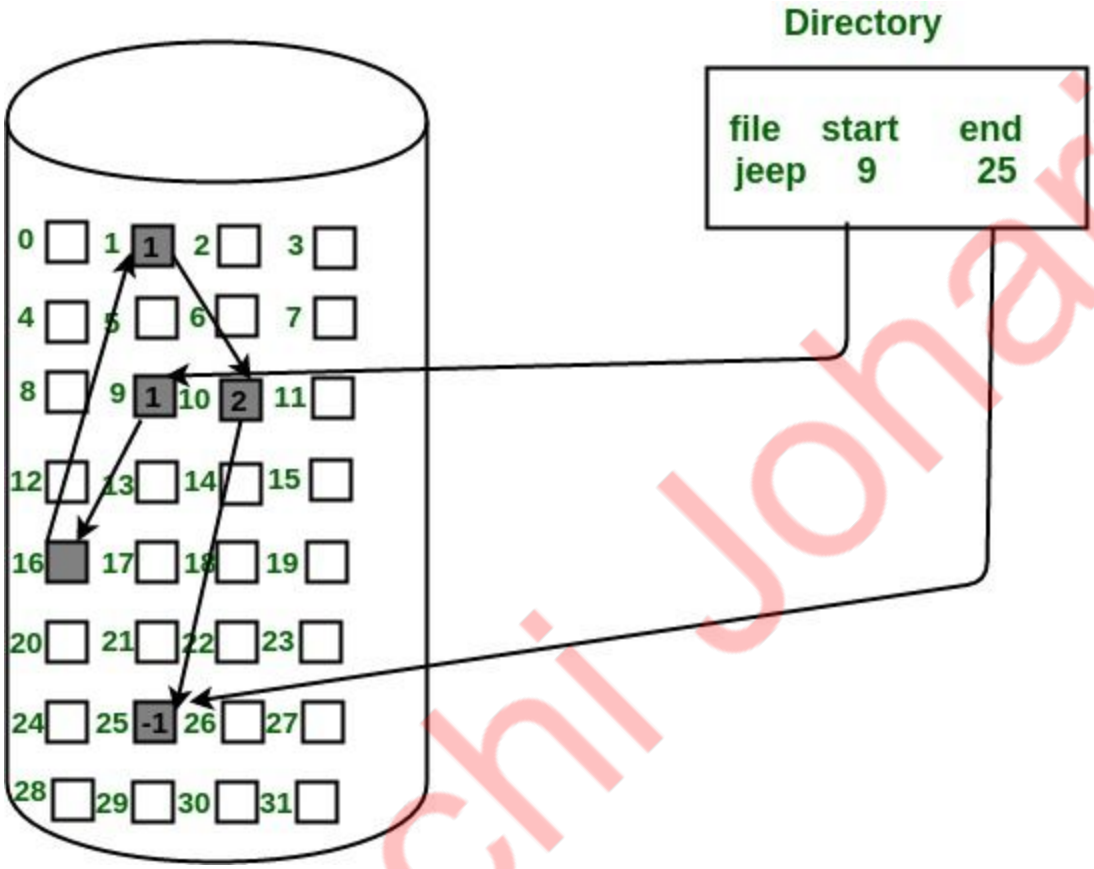
Disadvantages:

- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

B. Linked List Allocation

- In this scheme, each file is a linked list of disk blocks which **need not be** contiguous.
- The disk blocks can be scattered anywhere on the disk.

- The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.



Advantages:

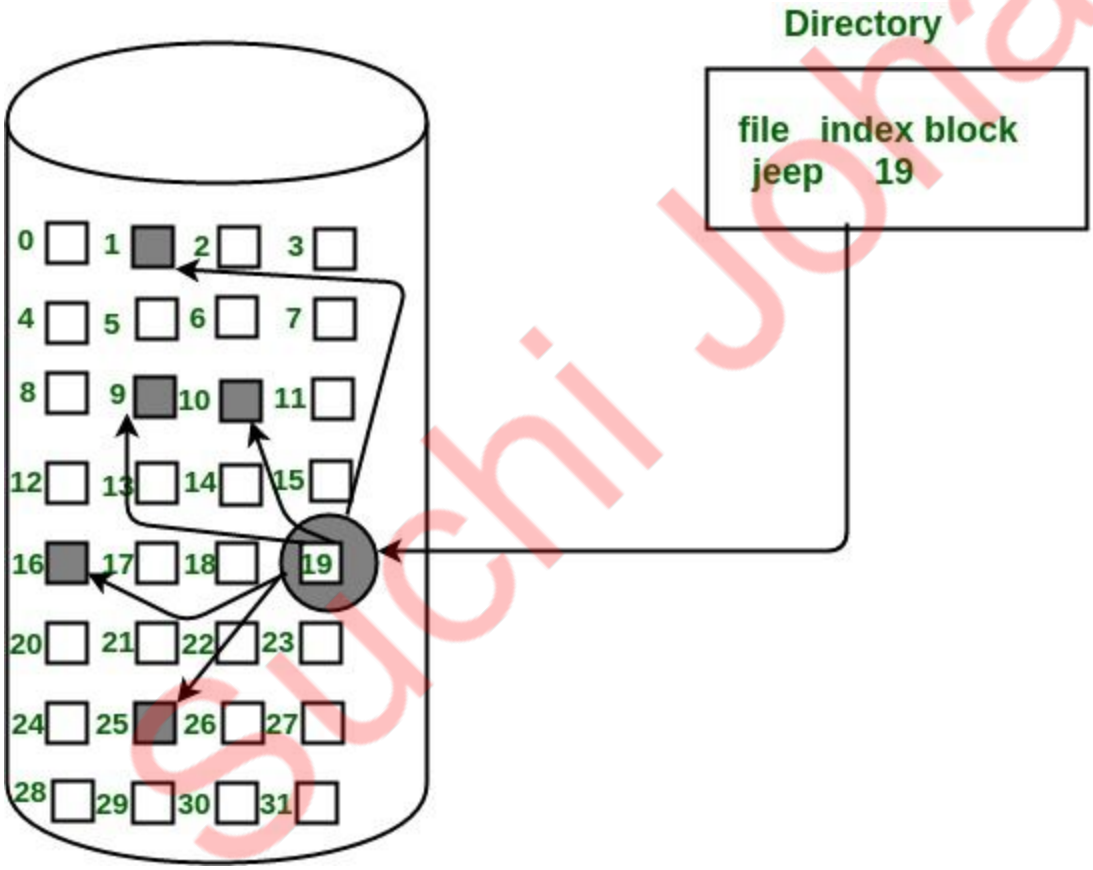
- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

Disadvantages:

- Because the file blocks are distributed randomly on the disk, 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. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead.

C. Indexed Allocation

- A special block known as the **Index block** contains the pointers to all the blocks occupied by a file.
- Each file has its own index block.
- The ith entry in the index block contains the disk address of the ith file block.
- The directory entry contains the address of the index block as shown in the image:



Advantages:

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

Disadvantages:

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which

is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

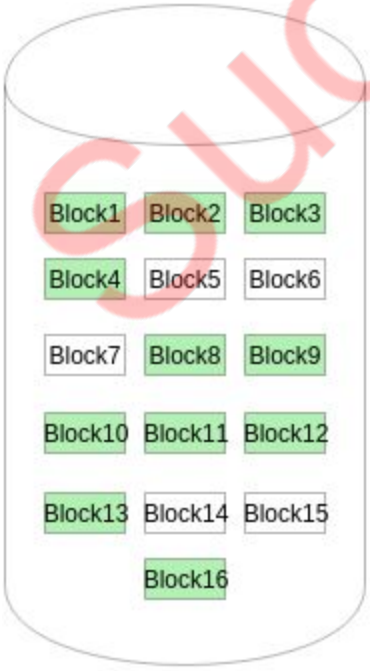
Free space management in Operating System

- The system keeps tracks of the free disk blocks for allocating space to files when they are created.
- To reuse the space released from deleting the files, free space management becomes crucial.
- The system maintains a free space list which keeps track of the disk blocks that are not allocated to some file or directory. The free space list can be implemented mainly as:

1. *Bitmap or Bit vector*

- A Bitmap or Bit Vector is a series or collection of bits where each bit corresponds to a disk block.
- The bit can take two values: 0 and 1: 0 indicates that the block is allocated and 1 indicates a free block.

Example: 0000111000000110



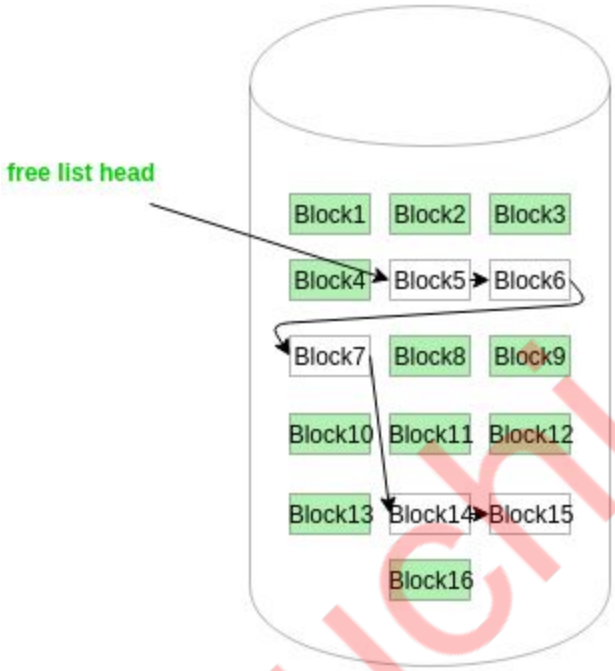
Advantages –

- Simple to understand.

- Finding the first free block is efficient.

2. ***Linked List –***

- The free disk blocks are linked together i.e. a free block contains a pointer to the next free block.
- The block number of the very first disk block is stored at a separate location on disk and is also cached in memory.



3. ***Grouping –***

- This approach stores the address of the free blocks in the first free block.
- The first free block stores the address of some, say n free blocks.
- Out of these n blocks, the first n-1 blocks are actually free and the last block contains the address of the next free n blocks.
- An advantage of this approach is that the addresses of a group of free disk blocks can be found easily.

4. ***Counting –***

- This approach stores the address of the first free disk block and a number n of free contiguous disk blocks that follow the first block.

Every entry in the list would contain:

1. Address of first free disk block
2. A number n